SOFTENG 2016

The Second International Conference on Advances and Trends in Software Engineering

ISBN: 978-1-61208-458-9

February 21 - 25, 2016

Lisbon, Portugal

SOFTENG 2016 Editors

Hermann Kaindl, Vienna University of Technology, Austria

Roberto Meli, DPO Srl, Italy

SOFTENG 2016

Forward

The Second International Conference on Advances and Trends in Software Engineering (SOFTENG 2016), held between February 21-25, 2016 in Lisbon, Portugal, continues a series of events focusing on challenging aspects for software development and deployment, across the whole life-cycle.

Software engineering exhibits challenging dimensions in the light of new applications, devices and services. Mobility, user-centric development, smart-devices, e-services, ambient environments, eHealth and wearable/implantable devices pose specific challenges for specifying software requirements and developing reliable and safe software. Specific software interfaces, agile organization and software dependability require particular approaches for software security, maintainability, and sustainability.

The conference had the following tracks:

- Software design and production
- Maintenance and life-cycle management
- Software requirements
- Software reuse
- Software testing and validation

We take here the opportunity to warmly thank all the members of the SOFTENG 2016 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors that dedicated much of their time and effort to contribute to SOFTENG 2016. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

Also, this event could not have been a reality without the support of many individuals, organizations and sponsors. We also gratefully thank the members of the SOFTENG 2016 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that SOFTENG 2016 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the field of software engineering. We also hope that Lisbon, Portugal provided a pleasant environment during the conference and everyone saved some time to enjoy the beauty of the city.

SOFTENG 2016 Advisory Committee

Alain Abran, University of Québec, Canada

Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden

Paolo Maresca, VERISIGN, Switzerland

Patricia McQuaid, California Polytechnic State University, USA

SOFTENG 2016 Research Liaison Committee
Veena Mendiratta, Bell Labs, Alcatel-Lucent, USA
Michael Perscheid, SAP Innovation Center Potsdam, Germany
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea
Fergal Mc Caffery, Dundalk Institute of Technology, Ireland
Ron Watro, BBN Technologies, USA

SOFTENG 2016 Industrial Liaison Committee
Tomas Schweigert, SQS Software Quality Systems AG, Germany
Janne Järvinen, F-Secure Corporation - Helsinki, Finland

SOFTENG 2016

Committee

SOFTENG 2016 Advisory Committee

Alain Abran, University of Québec, Canada
Mira Kajko-Mattsson, Stockholm University & Royal Institute of Technology, Sweden
Paolo Maresca, VERISIGN, Switzerland
Patricia McQuaid, California Polytechnic State University, USA

SOFTENG 2016 Research Liaison Committee

Veena Mendiratta, Bell Labs, Alcatel-Lucent, USA
Michael Perscheid, SAP Innovation Center Potsdam, Germany
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea
Fergal Mc Caffery, Dundalk Institute of Technology, Ireland
Ron Watro, BBN Technologies, USA

SOFTENG 2016 Industrial Liaison Committee

Tomas Schweigert, SQS Software Quality Systems AG, Germany
Janne Järvinen, F-Secure Corporation - Helsinki, Finland

SOFTENG 2016 Technical Program Committee

Haider Abbas, Center of Excellence in Information Assurance - King Saud University, Saudi Arabia
Alain Abran, University of Québec, Canada
İbrahim Akman, Atilim University, Turkey
Issam Al-Azzoni, King Saud University, Riyadh, Saudi Arabia
Magnus Almgren, Chalmers University of Technology, Sweden
Jocelyn Aubert, Luxembourg Institute of Science and Technology (LIST), Luxembourg
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea
Alessandra Bagnato, SOFTEAM R&D Department, France
Boyan Bontchev, Sofia University "St Kl. Ohridski", Bulgaria
Leonardo Bottaci, University of Hull, UK
Thomas Buchmann, Universität Bayreuth, Germany
Matthias Buechler, Technische Universität München, Germany
Azahara Camacho, Universidad Complutense de Madrid, Spain
Pablo Cerro Cañizares, Universidad Complutense de Madrid, Spain

Veena Mendiratta, Bell Labs, Alcatel-Lucent, USA
Andréa Mendonça, Instituto Federal do Amazonas (IFAM), Brazil
Akbar Siami Namin, Texas Tech University, USA
Erika Nazaruka (Asnina), Riga Technical University, Latvia
Risto Nevalainen, FiSMA Association, Finland
Cu Duy Nguyen, SnT Centre - University of Luxembourg, Luxembourg
Nicole Novielli, University of Bari "A. Moro", Italy
Antonio Pecchia, Federico II University of Naples, Italy
Michael Perscheid, SAP Innovation Center Potsdam, Germany
Pasqualina Potena, Fondazione Bruno Kessler, Italy
Wolfgang Reif, Institute for Software & Systems Engineering - University of Augsburg, Germany

Ella Roubtsova, Open University of the Netherlands, Netherlands
Alejandra Ruiz, TECNALIA, Spain
Gunter Saake, Otto-von-Guericke-University of Magdeburg, Germany
Kazi Muheymin Sakib, University of Dhaka, Bangladesh
Tomas Schweigert, SQS Software Quality Systems AG, Germany
Laura Semini, Università di Pisa, Italy
Michel S. Soares, Federal University of Sergipe, Brazil
Gunther Spork, Magna Powertrain AG & Co KG, Austria
Miroslaw Staron, University of Gothenburg, Sweden
Vinitha Hannah Subburaj, Baldwin Wallace University, Berea - Ohio, USA
Kumiko Tadano, NEC Corporation, Japan
Kunal Taneja, Accenture Technology Labs, San Jose, USA
Nguyen Anh Tuan, Boston Global Forum, USA
Tugkan Tuglular, Izmir Institute of Technology, Turkey
Sylvain Vauttier, LGI2P - Ecole des Mines d'Alès, France
Miroslav Velev, Aries Design Automation, USA
Gursimran S. Walia, North Dakota State University, USA
Hironori Washizaki, Waseda University, Japan
Gera Weiss, Ben-Gurion University of the Negev, Israel
Stephan Weissleder, Thales Transportation Systems, Germany
Ralf Wimmer, Institute of Computer Science - Albert-Ludwigs-University, Germany
Guowei Yang, Texas State University, USA
Cemal Yilmaz, Sabanci University, Turkey
Mansooreh Zahedi, IT University of Copenhagen, Denmark
Peter Zimmerer, Siemens AG, Germany
Alejandro Zunino, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Using the Event-B Formal Method for Disciplined Agile Delivery of Safety-critical Systems 1
Andrew Edmunds, Marta Olszewska, and Marina Walden

Unifying Modeling and Programming with ALF 10
Thomas Buchmann and Alexander Rimer

A Systematic Approach to Assist Designers in Security Pattern Integration 16
Loukmen Regainia, Cedric Bouhours, and Sebastien Salva

Function Point Analysis with Model Driven Architecture Applied on Frameworks of Partial Code Generation 23
Rodrigo Salvador Monteiro, Roque Pinel, Geraldo Zimbrao, and Jano Moreira de Souza

Developing Software for Mobile Devices: How to Do That Best 29
Hermann Kaindl, Roberto Meli, Andreas Kurtz, Bernhard Bauer, and Petre Dini

Developing a Quality Report for Software Maintainability Assessment: An Exploratory Survey 36
Pascal Giessler, Manuel Gerster, Michael Gebhart, Roland Steinegger, and Sebastian Abeck

A Tree-Based Approach to Support Refactoring in Multi-Language Software Applications 44
Hagen Schink, David Broneske, Reimar Schroter, and Wolfram Fenske

Collecting Product Usage Data Using a Transparent Logging Component 50
Thorvaldur Gautsson, Jacob Larsson, and Miroslaw Staron

The ICT Measurement System Definition, Components and a Maturity Evaluation Approach 56
Roberto Meli

Distributed Asynchronous Focus Group Interviews 65
Ulrike Hammerschall

Applying Privacy by Design in Software Engineering - An European Perspective 69
Karin Bernsmed

End User in Charge - Social Framework for Open Source Development 77
Kwabena Ebo Bennin, Mohammed Alqadhi, Shahid Hussain, Arif Ali Khan, Solomon Mensah, and Ernest Pobee

Sequence Data Mining Approach for Detecting Type-3 Clones 82
Yoshihisa Udagawa and Mitsuyoshi Kitamura

A Multi-Agent System for Expertise Localization in Software Development 89

Jose Ramon Martinez Garcia, Ramon Rene Palacio Cinco, Joel Antonio Trejo Sanchez, Luis Felipe Rodriguez, and Joaquin Cortez

Exploring the Scala Macro System for Compile Time Model-Based Generation of Statically Type-Safe REST Services 95
Filipe R. R. Oliveira, Hugo Sereno Ferreira, and Tiago Boldt Sousa

Migration from Annotation-Based to Composition-Based Product Lines: Towards a Tool-Driven Process 102
Fabian Benduhn, Reimar Schroter, Andy Kenner, Christopher Kruczek, Thomas Leich, and Gunter Saake

Executable Testing based on an Agnostic-Platform Modeling Language 110
Concepcion Sanz, Alejandro Salas, Miguel de Miguel, Alejandro Alonso, and Juan Antonio de la Puente

Software Based Test Automation Approach Using Integrated Signal Simulation 117
Andreas Kurtz, Bernhard Bauer, and Marcel Koeberl

Mobile Medical Apps Data Security Overview 123
Ceara Treacy and Fergal Mc Caffery

Reachability Games Revisited 129
Imran Khaliq and Gulshad Imran

Dynamic Symbolic Execution with Interpolation Based Path Merging 133
Andreas Ibing

Verification of Architectural Constraints on Interaction Protocols Among Modules 140
Stuart Siroky, Rodion Podorozhny, and Guowei Yang

Using the Event-B Formal Method for Disciplined Agile Delivery of Safety-critical Systems

Andrew Edmunds*, Marta Olszewska† and Marina Waldén‡
Distributed Systems Lab.
Abo Akademi University,
Turku, Finland
Email: *aedmunds@abo.fi, †mplaska@abo.fi, ‡mwalden@abo.fi

***Abstract*—In order to improve the development process of high-integrity systems, using formal methods, we consider how agile techniques may influence the Event-B formal method, and how Event-B may be used in a development that uses an agile approach. To examine the crossover between Event-B and agile methods we review the Disciplined Agile Delivery approach (DAD). The DAD approach is inspired by many state-of-the-art agile techniques, and we use it as a meta-analysis of current best-practice. In this paper, we propose an agile process for using Event-B and examine how agile techniques might influence the use of Event-B. We identify a number of areas in which Event-B could be improved and suggest that a different view of agile practices may be needed for an agile project involving formal development.**

Keywords–Agile; Formal Methods; Event-B; Critical Systems

I. INTRODUCTION

As part of the ADVICeS project [1], we have been investigating the crossover between agile and formal methods [2], [3]. In particular, we are focussing on the Event-B method [4] and DAD [5]. Event-B is a formal approach for the rigorous specification of safety-critical systems, based on set-theory and predicate logic. In contrast, DAD is a pragmatic, flexible approach to agile development, which seeks to guide teams towards a solution, rather than prescribe a list of tasks that should be adhered to. DAD's creators are experienced agile practitioners, taking inspiration from the many agile methods they have encountered during their careers, including XP [6], Scrum [7] and Lean approaches [8]. Since the DAD approach is so wide ranging, we use it for a meta-analysis, to assess how DAD and Event-B could be used together. The result provides an insight into the most relevant issues. We believe that this may be of interest to others in the formal methods community, since the questions we raise, and seek to answer, may apply to other methods. In Section II, we introduce Event-B. In Section III, we look at DAD and how it relates to Event-B. In Section IV, we examine how Event-B features may be used in a DAD project. In Section V, we discuss process goals. In Section VI, we discuss related work, and in Section VII, we summarize and reflect, asking the question "What's stopping Event-B from being used in an agile development?"

II. EVENT-B

Event-B is a specification language and methodology with tool support called Rodin [4], [9], [10]. Event-B has received interest from industry, for the development of railway, automotive, and other safety-critical systems [11]. In Event-B, system properties are specified using set-theory and predicate logic; proof and refinement are used to show that the properties hold as the development proceeds. It is recommended that developers add detail to the specification in a series of refinement steps. Proof obligations are automatically generated by the Rodin tool; the automatic prover is able to discharge many of them. Event-B is designed to reduce the amount of interactive proof needed, during the specification and refinement steps [12]. Complex systems can be handled using methods of composition and decomposition [13].

context C0
sets S ...
constants
c ...
axioms
c ∈ S

Figure 1. Example Event-B Context

A. Event-B's Core Technology

The core Event-B modelling artefacts are Machines and Contexts. Contexts describe the static elements of a system, using sets, constants and axioms. Machines describe the dynamic elements of a system; a machine can *see* contexts, and is made of variables, invariants, and guarded events. A simple context is shown in Figure 1. The context C0 declares a carrier set *S*, and a constant *c*, which is typed in the axioms as a member of *S*. Axioms can be used to type constants and specify assumptions. There are a number of built-in types such as $\mathbb{Z}$ and BOOL, and new types and operators can be added [14].

In Figure 2, we have a machine $M0$, which *sees* the context $C0$. In this way, $M0$ gains access to the sets and constants declared within $C0$. The machine's variables are typed in the invariants clause, which may also contain the description of other system properties.

Events describe atomic state updates; they are non-deterministically scheduled, occurring only when the guards (in the *where* clause) are true. The example shows a parameter *p1*, which is typed as an integer. It has a guard, which states that the updates can only occur when $x < p1$. In the action (the

 ISBN: 978-1-61208-458-9

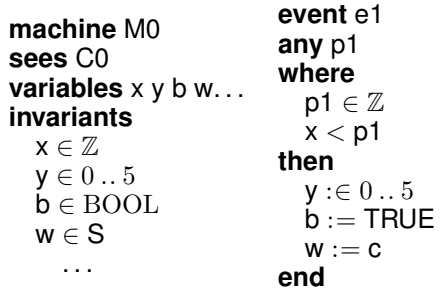


Figure 2. Example Event-B Machine

then clause) y is non-deterministically assigned a value in the range $0 \ldots 5$. Variables b and w are assigned (deterministic assignment) TRUE and c, respectively. All three assignment actions take place simultaneously, in parallel.

Refinement is used to add detail to a model. A refining machine is added to the development, then new variables and events can be added; or abstract events can be refined by strengthening the guards, and adding new actions. The new actions can modify the newly introduced variables.

B. Extensions to the Core Functionality

The Rodin tool is based on Eclipse [15], an open, extensible tool platform that allows plug-in developers to contribute new tools. There are many Rodin plug-ins available [16], we describe a few of them in this section.

One of the first steps in a project is to analyse the requirements. In Event-B, requirements can be recorded using **ProR** [17]. Elements in the Event-B model can be linked to references in the requirements for traceability throughout the development. Use case analysis can be done using a use-case modelling plug-in [18].

The **Theory** extension provides a way to add new data types, operators, inference and rewrite rules, and code generation translation rules [14], and theories can be reused in different projects. There are several extensions that provide **diagrammatic representations** with Event-B semantics, such as state-machines [19], class-diagrams [20], and progress diagrams [21].

When modelling, it is recommended that the behaviour of the model be explored using **ProB** [22], an animator and model-checker for Event-B. ProB allows developers to make sure that the expected behaviour, of a model, is observed. It can also be used to interrogate a model's state when proofs fail. ProB's model-checking facilities include a Linear Temporal Logic feature. An SMT solver is available in a separate plug-in [23]. Typically, ProB is used by technically-aware team members. There are additional animation tools for the non-technical stakeholders, to help them understand the system. Two such tools are **AnimB** [24] and **B-Motion Studio** [25]. Both perform similar roles, where the underlying elements of an Event-B model are linked to some graphical representations. As events occur, the state changes, together with the visual representation. This can be especially useful when validating requirements, and when showing the development to non-technical team members and stakeholders.

As models become more complex, **decomposition** techniques can be used to make the model of the system more tractable [13]. This also facilitates parallel development, since the sub-models can be refined independently. It can be used to introduce structure in the model, and ultimately, it provides a means to refine into implementation structures. The composed machine component provides the glue for linking the sub-models. This can also be used to compose independently produced components. Another approach is to use the **modularisation** plug-in [26], which takes a different view. It uses interfaces, and it introduces the concepts of operations, pre-conditions, and post-conditions into Event-B.

As the development progresses, it may be useful to perform simulations with a continuous representation of the environment, or with a continuous model of parts of the system. This may be achieved with **multi-simulation tools**, such as that based on the Functional Mock-up Interface (FMI), for simulation of cyber-physical systems [27]. This approach facilitates modelling, using a mix of continuous and discrete models, which can provide more confidence that non-functional requirements will be satisfied. A further step is to generate code from the Event-B models, which can then be included in an FMI simulation [28]. This simulation uses code that is very close to the actual software that will run in the system, and can easily be tailored for use in deployment. This approach is an extension of the work introduced in [29], where **code generation** tools use a scheduling language to describe implementations for multi-tasking, embedded systems. The target languages include Ada, Java and C. Another approach to automatic code generation includes that of Rivera et al. [30], who produce Java code, and Java Modelling Language (JML) annotations for downstream verification. Whether or not the code generator is certified, it can be desirable to perform testing. There is a **model-based testing** tool available for Event-B [31], which generates test-cases from the Event-B models based on ProB.

III. DAD

The Agile Manifesto [32] was presented in 2001, bringing together many new ideas that facilitate efficient, and timely, software development, in an ever changing environment. A number of concepts figured prominently. These include early delivery of "useful" resources; iterative development; collaboration between customers, and the development team; and collaboration within the development team itself. A number of approaches were advocated, such as XP [6], Scrum [7] and Lean [8]. These approaches consist of "guidelines for delivering high-quality software in an agile manner" [33]. Hence, it is apparent that agile has no single definition, and the approaches do not always cover all the necessary aspects of a development.

In [34], Ambler and Lines comment on the fact that agile practices were often adapted to the needs of the businesses in which they were applied. These authors advocate DAD [5], which provides a framework for this process. It includes a set of guidelines, drawn up as a result of analysing where agile approaches fall short. In the remainder of the paper, we make use of DAD, to understand how Event-B may be used in an agile project.

DAD is described by its authors as an adaptable process framework; they identify and discuss the many techniques and tools that may be used during agile development. Instead of building on radical, new technologies, the framework adopts

 ISBN: 978-1-61208-458-9

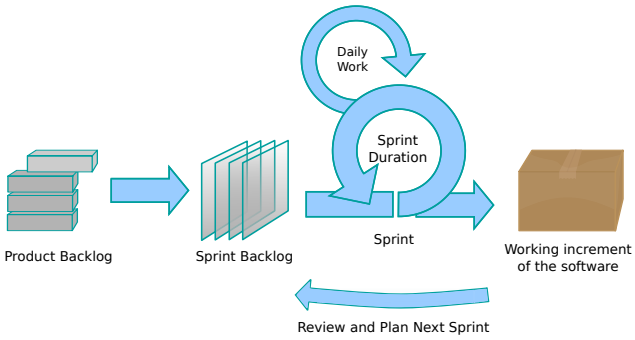


Figure 3. The Traditional Scrum Life-cycle

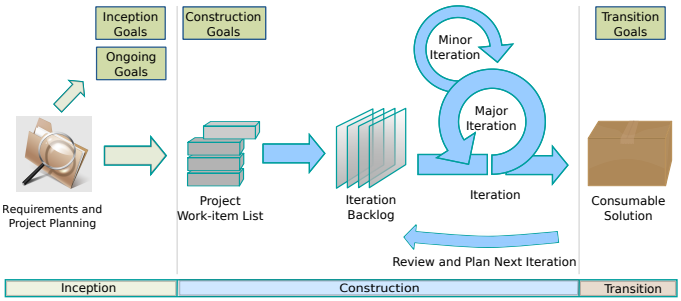


Figure 4. A Disciplined Agile Delivery Life-cycle

a pick-and-mix approach from existing technologies, based on the authors' practical experience in industry. Rather than being a step-by-step guide, there are a set of points to consider, and a set of guidelines that may be followed if appropriate. Many of the techniques that are part of DAD can also be used with Event-B. Much of the advice relates to project planning issues. One of the decisions includes the choice of which technologies to use, such as, whether to use Event-B. If the decision is yes, then a choice of modelling features is available; options include state-diagram [19] and class-diagram representations [20]. The scope of the formal model should also be defined.

1) Project management: Some of DAD's core concepts concern project management; others relate to individuals and team-working; and, others relate to tools and techniques. From the project management perspective, DAD does not impose a single life-cycle model. We understand it as a process goal-driven approach; as a side-note, the process goals in DAD should not be confused with goal-based requirements. In a process goal-driven approach, a semi-formal decision-making process is advocated, taking place at certain stages in the development, with the aid of process goal diagrams. The diagrams are tree-structured, and identify (at its root) a process decision point - with sub-branches showing the aspects (called process factors) to be considered. The leaves identify the choices that are available. We have identified this as one area which could be useful for Event-B, see Section V.

Scrum is considered to be a construction-centric approach focussed on producing working code at the end of each iteration, see Figure 3. With Scrum, code delivery is seen as the measure of progress. DAD can be viewed as an extension of Scrum, adding an inception phase, and a transition phase. It delivers consumable solutions, which can be seen as the more general notion of artefacts that are useful for stakeholders. In addition, DAD advocates the use of technology neutral terms; scrum's *sprints* are known as *iterations*, see Figure 4. We also note that there are goals for each development phase, and on-going goals that are considered throughout the project. Inception is the upfront consideration of design and architectural issues, and transition relates to the process of releasing a project.

In a safety-critical setting, there is likely to be more emphasis on gathering requirements early on, when compared to non-critical agile developments. Using DAD as a guide, it recommends that this stage is short relative to the construction phase of the project. For safety-critical parts of a system, this could involve a structured approach to requirements specification. We discuss how this might be done in Section V. But, the brevity is mitigated by the fact that the process will be iterative, so details evolve as the project progresses. In the DAD process, an important task of the inception phase is to scope the project and plan for the subsequent phases.

The activities performed during a DAD iteration are driven by a prioritised *work-item list*, which may contain requirements, change requests, feature requests, and so on. This expands on the scrum notion of a *product backlog* which may just contain requirements. The *sprint backlog* of scrum is known as an *iteration backlog* in DAD. The *work-item list* also takes into account the risks associated with the item. It is suggested that high-risk items are dealt with first, in order to minimise the risk to the project as a whole. It is recommended that Event-B is introduced to the process at an early stage of development, to begin modelling the critical parts of a system. Both approaches are compatible in this respect. Planning issues, such as which tasks should be undertaken during the next iteration, are discussed in the iteration planning phase. In safety-critical systems, we can choose to take a different view of *iterations*. When certification of the system is required, it is unlikely that this will take place more than three or four times a year, since it is a lengthy process. This means that there may be a different interpretation of iterations: *minor iterations* can be used for achieving and measuring project progress, with fully certified releases completed at the end of *major iterations*. The transition to a released, and supported product is of less interest in this paper. The life-cycle may be repeated from the inception phase, for each release of a product, including maintenance releases.

2) Team-working: DAD emphasises teamwork and the factors involving the individuals that make up the team. Motivated individuals are encouraged to learn, and make improvements, on an individual level, and at the team level. The lessons learned should be shared, too. New guidelines and patterns should be created where necessary. However, this needs to be done in a structured way, so that the patterns can be retrieved at an appropriate time in the future: there should be sufficient information available, to allow decision makers, and potential users, to make their choices. This can form part of the information that is available at the time of making process decisions, which we discuss in Section V.

3) Development Techniques: In the available literature, as in DAD, practitioners advocate the use of test-driven development. Test-driven development begins with the creation of tests that relate to some required feature. This creates a link between requirements, and tests, and provides traceability

 ISBN: 978-1-61208-458-9

between requirements and code. The test is first written so that it fails, then an implementation is coded so that the test no longer fails. The main advantage is that it ensures that the implementation passes the test, from a very early stage, and also, as features are introduced. However, there is no assurance that the test, itself, is correct. The test can be further used as part of a continuous integration process. "Test early and test often", is advice that is commonly given, and developers can generate several builds in a day. Also, since tests are only introduced when necessary, it is seen as a Just-In-Time (JIT) technique. When responding to changes, tests have to be changed, and the is code refactored, which helps to reduce technical debt [35].

We now consider how these advantages can be maintained, in a development that uses Event-B. We have highlighted that ProR can be used to link requirements to models, so that aids traceability. There are also tools available for automatic code generation [36], [29], and model-based testing [37], [38], that can be used to generate tests. This would make it amenable to continuous integration. Event-B can also be considered as a JIT approach since abstraction and underspecification are ideal ways of deferring the addition of specific details. Refactoring is also possible with Event-B, and improving the tool support for this is ongoing. Increased confidence in the correctness of code could be gained by generation of verification conditions, such as JML [39] for Java.

IV. Using Event-B in a DAD Project

We consider that DAD and Event-B may be of interest to organizations developing safety-critical systems. We discuss how Event-B might be used in an agile way, as part of a DAD-driven project. In DAD, emphasis is placed on the delivery of consumable solutions, which are a fully working revision of the system. However, we view models, diagrams, simulations, and animations, as consumable solutions; these are delivered at the end of each iteration. Therefore, we consider that working code is not the only measure of progress. This is different from the standard DAD approach where only a fully working revision of the system is considered to be a deliverable. It is often the case, that artefacts such as proven models, test results, and simulation results, are necessary to build the safety case that is required for certification of a critical system [40]. We believe that Event-B and DAD complement each other well, and we now present our view of why agile methods, and DAD, in particular, do so.

We expect it to be beneficial to start using Event-B early in the inception phase of development. Since the DAD life-cycle is flexible, it should be straightforward to incorporate Event-B into the process. However, a development iteration for a formal development may be longer than that of an informal development. This may then require the management of a two-speed development. If this is to be integrated into a process with a non-formal part, care will need to be taken to synchronise the iterations.

There are a number of ways to view, and animate Event-B models, and perform simulations. These should be useful in the inception phase onwards. Using these, teams can be kept informed of the latest developments, which is very important in a team-centric approach. In Figure 5 (which is inspired by the DAD life-cycle diagram (Figure 2) of [41]) we interpret Event-B artefacts as being consumable solutions

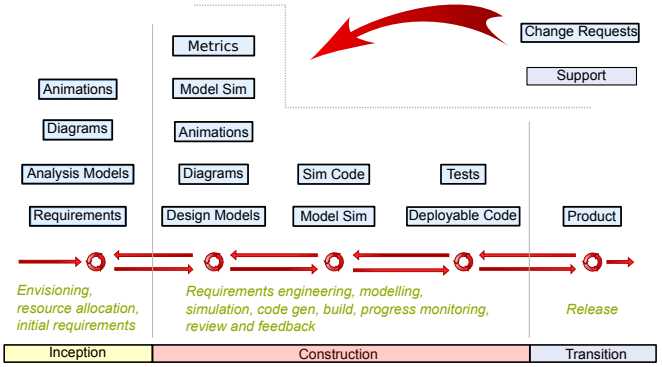


Figure 5. Artefacts Available During an Agile-Event-B Project

of the DAD process. We take the view that stakeholders will be consumers of the artefacts, and the consumables may vary as the project proceeds. When not contributing directly towards product deliverables, they should contribute to making progress throughout the development life-cycle. This includes supporting the process-decision making activity or building a safety-case for certification.

A. Event-B in the Inception and Construction Phases

During the inception stage, we could begin recording requirements using ProR [17]. Requirements can be linked to corresponding Event-B elements, which facilitates traceability. DAD guidelines recommend separating functional and non-functional requirements into different artefacts, in order to focus on the individual needs of each set of requirements. This would be possible in ProR, by generating two requirements specifications. Event-B is most useful for models involving functional, and safety, requirements; whereas non-functional requirements are more suited to simulation and testing. In the latter case, Event-B can assist with multi-simulation [42], [43].

The DAD approach advocates minimal requirements specification but recognises that a fuller treatment is needed for a critical system development, which is likely for projects using Event-B. In Figure 5, we show when various artefacts become available during the various phases. In the inception phase, the system's most important, high-level properties can be stated, in abstract models. These ideas can be communicated to non-technical stakeholders, using animation and other graphical techniques. B-Motion Studio [25] can be used to link images representing domain elements, to the Event-B model. An image's appearance and location can be linked to the state of a model. So, as the model's underlying state changes, then so does the graphical representation. For those requiring more technical details, the ProB animator and model-checker can be used [22]. It provides a simulation feature, where more precise details of the model can be examined. A choice of enabled events is presented to the user, allowing step-by-step analysis of the evolving system. The current state, state history, and event history, are presented. Details of the guard predicates are available, to help resolve event enablement issues. The frequent use of ProB is recommended during modelling and proving.

As the development progresses through the construction phase, state-diagrams [19] and class-diagrams [20] can be used

 ISBN: 978-1-61208-458-9

for specification, and can be used to communicate aspects of the design to other stakeholders. Throughout the development, we can assess more of the non-functional requirements, as the design matures, by simulating Event-B models with more accurate representations of continuous state. Such an approach was developed in the Advance project [42], which focussed on cyber-physical systems, using the Functional Mock-up Interface (FMI) approach [43]. In later stages, code can be generated from the Event-B models, and simulation models can be replaced with Functional Mock-up Units (FMUs) containing code. Then the simulation more closely represents what happens in the actual target platform. Deployable code [36], [29] can be generated automatically from the Event-B models, if desired.

Configuration for specific target platforms can be done during the transition phase, in this case, pre-configured templates, and code injection may help [44]. During the support phase, change requests can be generated. These will be fed back into a new development cycle, as seen in Figure 5.

B. Some Observations on Construction Patterns

During the DAD construction phase, there are a number of patterns (strategies) that could be used. We investigate how they relate to the use of Event-B, exploring a number of them individually.

"The team reliably demonstrates increments of software at the end of each iteration" :- This is the demonstration that an iteration has been successful, but how do we measure success when using Event-B? If code is being generated from the Event-B models, and this code passes all the unit and integration tests, then we could use the same measure of success (by demonstrating the increment). However, if we are not generating code, how is success to be measured?

In order to understand how this strategy would relate to a project using Event-B, it is important to understand why it is considered to be so important in DAD. On closer inspection we find that the important issues are: highlighting problems at an early stage, avoiding scope creep, and resisting perfectionism. The use of Event-B is considered to be a major advantage in helping with the first issue: identifying problems early. The second, limiting scope creep, seems to be independent of the use of Event-B. The last point may be more of an issue with formal development, due to the emphasis on discharging proof obligations. Guidelines could be drawn up about when it is practical to proceed without completing a proof, how to measure the confidence in its correctness, and recording and reporting a proof's status.

We consider the relationship between an incomplete proof and a failing test. Both relate to the failure to satisfy a requirement. On page 297 of [5] it is stated that *"All tests should be implemented before the end of the iteration, for the work to be considered to be potentially consumable"*. However, on page 262, a concession is made in the case-study. At a point in the project, at the end of an iteration, there are still some failing tests. Now, the end-of-iteration date is immutable, so it is accepted that some tests fail. So it would appear that passing all tests should be the goal, but it may not be achievable in the time allowed. A pragmatic approach is to have a two-speed process involving minor-, and major-iterations. All proofs should be completed for major-iterations (which may only happen a few times a year). Due to the weaker requirement associated with minor-iterations, the model, together with adequate documentation, and a measure of model integrity should be sufficient.

"Iteration dates are fixed" :- This point initially seems to be technology-neutral. However, in order to meet this goal, the authors state that it might be necessary to withdraw items from the work-item list. It is suggested that this may work best for larger projects. In small projects, or the small part of a large project that uses Event-B, removal of a work-item from the list may be a more significant part of the iteration deliverable, this could be more of a problem. However, it could be mitigated by flexible working patterns, as recommended in the pattern below.

"Team members who finish their tasks begin another task from the iteration work-list, or help others with their tasks" :- This encourages developers to make the best use of the time available. It would seem to be common sense, but for it to be useful in Event-B, it requires some notion of collaborative working (in Event-B). It would certainly be possible to perform some kind of pair modelling. However, for modellers to be working on independent, parallel tracks, existing composition techniques need to be improved, and this work is being undertaken as part of the ADVICeS project [1]. It includes construction of a useful notion of components, libraries, and interfaces. If such techniques, for parallel development, are not available for Event-B or are ineffective, then Event-B could not really be considered agile with respect to team-working.

"Stakeholders may request a demo (of working software) at any time" :- In a properly versioned Event-B project, using automatic code-generation tools and continuous-integration, this is theoretically possible. It should always be possible to demonstrate *a* build, but it might not be the most recent work. In a project using minor-, and major-iterations, it may be some time between stable builds since minor iterations may be incomplete. From a practical standpoint, formal modelling is not always seen as a process where certified, automatic code-generation is important. If there is no certified, automatic code-generation, then we can still satisfy this goal. But there is less assurance that the high integrity of a formal model has been transferred to the implementation. In addition, if we are using ProB animation, or executable models, as a form of validation, then there is, again, less assurance that what has been validated is transferred to the implementation.

C. Construction Anti-patterns

A number of anti-patterns are described by the authors of DAD. We consider those of interest to us, and describe them as patterns. **"Mitigate risk"**: We assume that the use of Event-B was a step towards achieving this. The authors' advice is, to make sure any concerns are flagged and are prominent in the working environment. **"Prove the architecture, with code, early"**: Would this be too complex a task for Event-B to model and generate code, so early in construction? Event-B may be useful for modelling the architecture early in the project, but this aspect requires investigation. **"Handle missing requirements"**: The discovery of a missing requirement can have repercussions due to dependencies on other requirements. The strategy, in a software-based project, is to provide a stub for the missing functionality or swap out the blocked requirement. In Event-B, requirements are related to model invariants, so,

the missing requirement and its related invariant could simply be added. However, some analysis of the requirement should be done before adding it to the model, to assess validity. We envisage that a process, using Event-B in an agile way, would be able to accommodate this.

V. Process Goals and Event-B

Once the decision to use Event-B has been made, one may wish to examine what development strategies are available, or one may need more detailed guidance on some technical aspects. We have been inspired by DAD's goal-oriented decision approach, to propose a goal-driven process for Event-B development. This could be useful when modelling as part of an agile process, and may be useful to the Event-B community in general.

We consider two categories of potential users: experienced teams and individuals who need a check-list of things to consider, and novices. However, to accommodate continuous improvement in the process, even experienced teams need to be able to record what they have learned, in the form of guidelines, patterns, and components. They then need a way of retrieving that information, so that they may be used to make process decisions, or design decisions, or for use in modelling. Inability to find useful knowledge at an early stage of a project is a barrier to the successful introduction of Event-B. In the case of reusable components, the deployment of components is being investigated [1]. The second category of users is novice teams or individuals. In either case, introducing Event-B into a company's existing development process can be difficult since Event-B advocates can struggle to get accepted into existing company culture.

There has been a great deal of knowledge recorded already, about how to proceed at various stages of the development process. However, this knowledge is widely dispersed on the web, much of it in an unstructured manner. We would like to reduce the number of entry points for obtaining guidance, for the use of Event-B, backed by a goal-driven process and a structured repository of data. However, we are aware that many companies may wish to keep their own private repository too since the information contained within it would be their intellectual property. We should, therefore, attempt to accommodate this in our approach. Another barrier that exists is that much of the knowledge resides behind an academic pay-wall which prevents industry from gaining free access to much of the available information. Our main motivation is to facilitate the creation of a goal-oriented decision-making process. The aim is to guide users, both experienced and inexperienced, providing them with the correct information to make informed decisions. Users need information upon which to base the decisions, and they may also need guidance/reminders about what issues they need to consider.

A. Monitoring mechanisms and metrics

Feedback to developers about the progress of a project is given through monitoring mechanisms, e.g., with the use of metrics and measurements (see Figure 4). They should be easily accessible throughout the duration of the project so that they can provide up to date information, and enable prompt action on any issues that are observed. Collecting product and process measurements early in the development stages is beneficial since it enables the analysis of the quality of the model, as well as improvement of modelling and development process. Furthermore, it contributes to empirically validate the developers perception about the model, and allow better estimation of a project. In [3] we proposed the placement of monitoring mechanisms in short and long iterations of Scrum process. Since the DAD framework extends Scrum, we can use the same metrics and measures, and collect them at the suggested times.

Collecting useful and meaningful measurements, for formal developments used within an agile development process, requires much more than just reusing the metrics commonly used in the programming setting. Not only does one need to consider metrics in terms of requirements and models, but also keep in mind the specifics of formal modelling and agile processes. There are several ground rules regarding the monitoring process: keep it simple and continuous; collect and combine several metrics and look for trends; and, use metrics as indicators rather than facts [5].

Some metrics for agile, and in particular, DAD developments are mentioned in [5]. Below we present the ones that are advocated for DAD and can be easily adapted for Event-B development:

- *Burn rate* depicts how resources are invested in a particular effort, measured by counting, e.g., money or time involved in such an effort. In the case of Event-B modelling, one can use work points assigned to an item (being an estimate of the man-hours necessary to model a requirement or prove a property) or according to complexity estimation (points assigned on the experience-basis). It can be displayed by, e.g., using a burnup chart, where the burnup chart would be comprised of modelled and proven items, and the total amount of work to do on these.
- *Delivered functionality* is the only true measure of progress of a software development project, it is measured by tracking items in a project. In Event-B development observing the modelled and proven items with respect to the ones left in the project backlog would give the full picture on the state of the project. Progress of modelling in Event-B can also be measured with statemachine-based metrics [45].
- *Velocity* describes how much functionality the team can deliver per iteration. It is team-specific and measured in the form of counting "use case points" or "story points". For our purpose, counting requirements modelled and proved, or work points per iteration would show the velocity of Event-B development.

Note that the term *item* stands for a requirement, and/or a property.

As for quality aspect of a development, it is proposed to measure the number and severity of defects. However, in the case of Event-B developments, this metric is not meaningful, as the measurement should be kept on the model level. We propose to use several product measurements, e.g., for the size and complexity of a model, based on Event-B syntax [46] for each refinement step. Moreover, the number of proof obligations, both automatic and interactive, will indicate the complexity of modelling (and proving). Furthermore, adherence to modelling conventions and styles can be collected via model inspections and displayed in a form of a histogram or line chart.

 ISBN: 978-1-61208-458-9

Very much advocated for agile developments, *lifecycle traceability* (to requirements, design, etc.) is enabled by following the refinement of the artefacts of interest. This measurement also shows that product quality is heavily impacted by the quality of the process.

There are several process-related, time-based, metrics that can be useful for Event-B developments. These are, for instance, *activity time* – the time it takes to model and prove an item (requirement or property); *time invested* – the amount of time spent on a project, where work can be categorised by activities like modelling, proving, re-modelling; and *change cycle time* – the period of time from when a requirement, a property or enhancement request is identified, until it is resolved (cancelled, or modelled and proved). All of these metrics can be displayed as a trend on a histogram or line chart.

B. Other Guidance for Modellers/Developers

There are several other approaches that could contribute towards the decision-making process, e.g., in preparation for the iteration planning activity of Figure 4. Kobayashi et al. propose the use of a method to guide refinement strategies [47]. The dependencies between domain elements (called phenomena) and artefacts of the model (which are Event-B invariants) are analysed. Domain elements are modelled, as usual in Event-B, using sets, variables constants, and events. The analysis involves computing the relationships between the domain elements in the invariants, and comparing the order in which domain elements can be added to the model, with the aim of simplifying the refinement. From the analysis, a weighted list of refinement plans is derived, which can help guide development. In further work [48], the authors propose a systematic approach to discount invalid refinement plan proposals.

Another approach is to use existing refinement patterns to propose alternative refinement plans when proof fails [49]. In this approach, a reasoned modelling critic is used to analyse the model, and failing proofs. Under the covers, a search is performed to find the closest match(es) of the current model to existing patterns. Alternative modelling strategies, in the form of high-level refinement plans, are suggested to solve the problem. This approach provides feedback to the user on a day-to-day basis; and can also provide feedback in the high-level, strategic, decision process.

VI. Related Work

Agile methods have established themselves in industry as a diverse range of practices, aimed at improving the software development process. The use of agile methods in critical-systems developments, in particular, with formal methods, has been investigated. For instance, the work of Eleftherakis and Cowling describes XFun [50], an agile approach to formal development using the X-machine formal method. Paige and Brooke [51] show how an integrated formal method (Eiffel [52] and Communicating Sequential Processes (CSP) [53]) can be developed in an agile manner. However, here the emphasis is on methods engineering. More generally, Gary et al. [54] discuss their experience of using agile methods in a critical-system development. They conclude that there is scope for more research in this area. The Safety Critical Systems Club discuss the practicalities of using agile development in the real world [55], with contributions from various developers from the safety-critical systems business. They conclude that some agile practices are already in use in a number of the companies involved, but they are used with a great deal of caution. Wolff describes how the Vienna Development Method (VDM) could be used in a project that uses Scrum [56]. Here, the formal approach is used in parallel with traditional software engineering methods. The formal approach can be used to communicate details about the critical parts of the system to the implementers. In other work, Black et al. posit a positive, but cautious, outlook for an agile/formal mix, in [57], as do Larson et al. in [58].

VII. Conclusion: what is stopping Event-B from being used in an agile development?

We believe that, in theory, Event-B is suitable for use in an agile project. We know that agile methods are being used (at least to some extent) in safety-critical systems development [55], so we would expect that Event-B might be useful here, too. In the technical report [2], a preliminary assessment was performed, to determine which aspects of Event-B could be considered to contribute to an agile approach. A preliminary case-study demonstrating the use of Event-B with agile methods is described in [59]. There are a number of improvements that could be made to make Event-B more agile. Some are tooling issues, like improving refactoring, a major feature of the agile approach. Other improvements require theoretical advancement, as well as tool support, such as the development of reusable components or new diagrammatic representations. In addition to improvements to theory and tools, we may also seek to improve the modelling process. When considering how Event-B could be used as part of an agile engineering process, we believe that a process-decision framework, similar to the one advocated by DAD, would be useful. The aim would be to provide developers (both beginners and experts) with structured assistance throughout the development life-cycle.

When considering how DAD and Event-B complement each other or conflict, we ask many questions. For instance, how do we interpret *consumable solutions* when developing critical-systems with a formal method? Do they have to involve working code, as required in a mainstream software development with DAD? This is particularly relevant for embedded systems, where the ultimate deliverable involves hardware. In this case, simulation is used to postpone commitment to hardware, until the latest possible stage. So we take the view that consumables are artefacts that contribute (in a timely manner) to the ultimate success of a project. They may be of use to different stakeholder audiences; for instance, ProB animation for technically-minded developers, B-Motion Studio for the non-technical, metrics for project planners, and so on. In critical-system development, it may be acceptable to defer code generation, and use animation to perform on-going validation, since it is so important that everybody fully understands the issues involved. Code can be generated automatically from Event-B, but the refinement process, down to the implementation-level, may take longer; and the generated code might provide a less clear result, than model animation. It should be easier to make changes to an abstract model too, since unnecessary detail is omitted from the model, making this more agile.

 ISBN: 978-1-61208-458-9

We also need to understand how formal modelling culture would work in an agile world. An example of this difference might be that between failed tests, and failed proofs. At the end of an iteration, in an agile development, it can be acceptable to have some tests that fail (with agreement from the client). It is understood that as the development proceeds more is learned, which will help resolve the issue, downstream. It would seem, therefore, that it should be acceptable too, to have failed proofs at the end of an iteration; but this sits much less comfortably in the formal world, where so much effort is invested in discharging proof obligations. It may be considered that it is enough to be reasonably confident that a proof will succeed, even though the proof remains to be completed. If this is the case, how do we estimate and record our confidence, and how do we measure progress in light of this? In safety-critical systems development, one would almost certainly need the minor- and major-iteration approach if it is decided that all proofs must succeed for the safety-case to be acceptable; without feedback from the minor-iterations, the development process would not be very agile.

Team-working is a fundamental aspect of an agile project. In an Event-B project, this means that models must be worked on concurrently. We hope to do this with component-based development, which also promotes reuse. Composite components are components made with machines and other components. It should be possible to independently refine at the component-level. However, independent refinement, at a finer granularity (i.e, the sub-models of a component) may be problematic, due to the dependencies introduced by properties specified over the state of the whole component. We are currently exploring the issues, and we are developing a new notion of interfaces for Event-B in the ADVICeS project [1].

The work to maintain and improve Event-B is ongoing. We have identified that we may make a contribution by designing a goal-driven process decision framework, backed by a structured repository for guidelines, and practical advice about development strategies. This would supplement the existing support approach, which is in the form of a wiki. This idea is inspired by the DAD notion of a goal-driven process. The framework could target beginners and experts separately. It should provide help and guidance for novices; for expert users, the approach could provide a checklist of things to consider. The repository could include information about development patterns, guidelines for development strategies, and have a component library. Ideally, for use in industry, there should be an option to set up a private repository too, which might be required to protect a company's IP.

VIII. ACKNOWLEDGEMENTS

This work was carried out within the project ADVICeS, funded by Academy of Finland, grant No. 266373.

REFERENCES

[1] The ADVICeS Team, "The ADVICeS Project," available at https://research.it.abo.fi/ADVICeS/.

[2] M. Olszewska and M. Walden, "FormAgi - A Concept for More Flexible Formal Developments," Tech. Rep. 1124, 2014.

[3] M. Olszewska and M. Walden., "DevOps Meets Formal Modelling in High-Criticality Complex Systems." in In Proceedings of QUDOS2015 workshop within European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 2015.

[4] J. Abrial, Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

[5] S. Ambler and M. Lines, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press, 2012.

[6] R. Jeffries, A. Anderson, and C. Hendrickson, Extreme Programming Installed. Addison-Wesley Professional, 2001.

[7] K. Schwaber and J. Sutherland, "The Scrum Guide, from scrum.org," 2010.

[8] M. Poppendieck, "Lean Software Development," in Companion to the proceedings of the 29th International Conference on Software Engineering. IEEE Computer Society, 2007, pp. 165–166.

[9] J.R. Abrial et al., "Rodin: An Open Toolset for Modelling and Reasoning in Event-B," Software Tools for Technology Transfer, vol. 12, no. 6, Nov. 2010, pp. 447–466. [Online]. Available: http://dx.doi.org/10.1007/s10009-010-0145-y

[10] "The Rodin User's Handbook," Available at http:// handbook.event-b.org/.

[11] A. Romanovsky and M. Thomas, Industrial Deployment of System Engineering Methods. Springer, 2013.

[12] S. Hallerstede, "Justifications for the Event-B Modelling Notation," in B, ser. Lecture Notes in Computer Science, J. Julliand and O. Kouchnarenko, Eds., vol. 4355. Springer, 2007, pp. 49–63.

[13] R. Silva and M. Butler, "Shared Event Composition/Decomposition in Event-B," in FMCO Formal Methods for Components and Objects, November 2010. [Online]. Available: http://eprints.soton.ac.uk/272178/

[14] M. Butler and I. Maamria, "Practical Theory Extension in Event-B," in Theories of Programming and Formal Methods, ser. Lecture Notes in Computer Science, Z. Liu, J. Woodcock, and H. Zhu, Eds. Springer Berlin Heidelberg, 2013, vol. 8051, pp. 67–81. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39698-4_5

[15] J. des Rivieres and W. Beaton, "Eclipse Platform Technical Overview," Available at http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.

[16] "The Rodin Plug-ins Wiki," at http://wiki.event-b.org/index.php/Rodin_Plug-ins.

[17] M. Jastram, "ProR, an Open Source Platform for Requirements Engineering Based on RIF," available at http://deploy-eprints.ecs.soton.ac.uk/245/1/seisconf.pdf, 2010.

[18] R. Murali, A. Ireland, and G. Grov, "A Rigorous Approach to Combining Use Case Modelling and Accident Scenarios," in NASA Formal Methods, ser. Lecture Notes in Computer Science, K. Havelund, G. Holzmann, and R. Joshi, Eds. Springer International Publishing, 2015, vol. 9058, pp. 263–278. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-17524-9_19

[19] V. Savicks and C. Snook, "A Framework for Diagrammatic Modelling Extensions in Rodin," in Rodin Workshop Proceedings, 2012, pp. 31–32.

[20] C. Snook and M. Butler, "UML-B and Event-B: An Integration of Languages and Tools," in The IASTED International Conference on Software Engineering - SE2008, February 2008. [Online]. Available: http://eprints.ecs.soton.ac.uk/14926/

[21] M. Plaska, M. Walden, and C. Snook, "Documenting the Progress of the System Development," in Methods, Models and Tools for Fault Tolerance. Springer, 2009, pp. 251–274.

[22] M. Leuschel and M. Butler, "ProB: An Automated Analysis Toolset for the B Method." STTT, vol. 10, no. 2, 2008, pp. 185–203. [Online]. Available: http://dblp.uni-trier.de/db/journals/sttt/sttt10.html#LeuschelB08

[23] D. Déharbe, "Integration of SMT-solvers in B and Event-B Development Environments," Science of Computer Programming, vol. 78, no. 3, 2013, pp. 310–326.

[24] C. Métayer, "AnimB Homepage," http://www.animb.org/index.xml.

[25] L. Ladenberger, J. Bendisposto, and M. Leuschel, "Visualising Event-B models with B-motion Studio," in Formal Methods for Industrial Critical Systems. Springer, 2009, pp. 202–204.

[26] A. Iliasov et al., "Supporting Reuse in Event B Development: Modularisation Approach," in Abstract State Machines, Alloy, B and Z. Springer, 2010, pp. 174–188.

 ISBN: 978-1-61208-458-9

[27] V. Savicks, M. Butler, J. Bendisposto, and J. Colley, "Co-simulation of Event-B and Continuous Models in Rodin," in 4th Rodin User and Developer Workshop, June 2013. [Online]. Available: http://eprints.soton.ac.uk/360400/

[28] J. Bendisposto et al., "ADVANCE Deliverable D4.2 (Issue 2) Methods and Tools for Simulation and Testing I," The ADVANCE Project, Tech. Rep., 2013.

[29] A. Edmunds and M. Butler, "Tasking Event-B: An Extension to Event-B for Generating Concurrent Code," in PLACES 2011, February 2011. [Online]. Available: http://eprints.ecs.soton.ac.uk/22006/

[30] V. Rivera and N. Cataño, "Translating Event-B to JML-Specified Java Programs," in 29th ACM Symposium on Applied Computing, Software Verification and Testing track (SAC-SVT), Gyeongju, Korea, March 24-28 2014, pp. 1264–1271.

[31] I. Dinca, F. Ipate, L. Mierla, and A. Stefanescu, "Learn and Test for Event-B - A Rodin Plugin," in ABZ, ser. Lecture Notes in Computer Science, J. Derrick, J. A. Fitzgerald, S. Gnesi, S. Khurshid, M. Leuschel, S. Reeves, and E. Riccobene, Eds., vol. 7316. Springer, 2012, pp. 361–364.

[32] M. Fowler and J. Highsmith, "The Agile Manifesto," Software Development, vol. 9, no. 8, 2001, pp. 28–35.

[33] T. Dingsoyr, S. Nerur, V. Balijepally, and N. Moe, "A Decade of Agile Methodologies: Towards Explaining Agile Software Development ," Journal of Systems and Software, vol. 85, no. 6, 2012, pp. 1213–1221.

[34] S. Ambler and M. Lines, "Going Beyond Scrum: Disciplined Agile Delivery," Disciplined Agile Consortium. White Paper Series, 2013.

[35] V. Krishna and A. Basu, "Software Engineering Practices for Minimizing Technical Debt," in Proceedings of the International Conference on Software Engineering Research and Practice (SERP). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013, p. 1.

[36] F. Degerlund, R. Gronblom, and K. Sere, "Code Generation and Scheduling of Event-B Models," Turku Centre for Computer Science, Tech. Rep. 1027, 2011.

[37] Q. Malik, J. Lilius, and L. Laibinis, "Scenario-based Test Case Generation using Event-B Models," in Advances in System Testing and Validation Lifecycle, 2009. VALID'09. First International Conference on. IEEE, 2009, pp. 31–37.

[38] A. Stefanescu, F. Ipate, R. Lefticaru, and C. Tudose, "Towards Search-based Testing for Event-B Models," in Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on. IEEE, 2011, pp. 194–197.

[39] N. Cataño, T. Wahls, C. Rueda, V. Rivera, and D. Yu, "Translating B Machines to JML Specifications," in Proceedings of the 27th Annual ACM Symposium on Applied Computing. ACM, 2012, pp. 1271–1277.

[40] P. Bishop and R. Bloomfield, "A Methodology for Safety Case Development," in Industrial Perspectives of Safety-Critical Systems. Springer, 1998, pp. 194–203.

[41] S. Ambler and M. Lines, "Scaling Agile Software Development: Disciplined Agility at Scale," available at http://disciplinedagileconsortium.org/Resources/Documents/ScalingAgileSoftwareDevelopment.pdf, May 2014.

[42] The Advance Project Team, "Advanced Design and Verification Environment for Cyber-physical System Engineering," Available at http://www.advance-ict.eu.

[43] The Modelica Association Project, "The Functional Mock-up Interface," Available at https://www.fmi-standard.org/.

[44] A. Edmunds, "Templates for Event-B Code Generation," in 4th International ABZ 2014 Conference, 2014. [Online]. Available: http://eprints.soton.ac.uk/364265/

[45] M. Olszewska and M. Walden, "Measuring the Progress of a System Development," Dependability and Computer Engineering: Concepts for Software-Intensive Systems: Concepts for Software-Intensive Systems, 2011, p. 417.

[46] M. Olszewska and K. Sere, "Specification Metrics for Event-B Developments," in Proceedings of the CONQUEST 2010: "Software Quality Improvement", I. Schieferdecker, R. Seidl, and S. Goericke, Eds. International Software Quality Institute, 2010, p. 112.

[47] T. Kobayashi and S. Honiden, "Towards Refinement Strategy Planning for Event-B," arXiv preprint arXiv:1210.7036, 2012.

[48] T. Kobayashi, F. Ishikawa, and S. Honiden, "Understanding and Planning Event-B Refinement through Primitive Rationales," in Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings, 2014, pp. 277–283.

[49] G. Grov, A. Ireland, and M. Llano, "Refinement Plans for Informed Formal Design," in Abstract State Machines, Alloy, B, VDM, and Z. Springer, 2012, pp. 208–222.

[50] G. Eleftherakis and A. Cowling, "An Agile Formal Development Methodology," in Proc. 1st South-East European Workshop on Formal Methods, 2003, pp. 36–47.

[51] R. Paige and P. Brooke, "Agile Formal Method Engineering," in Integrated Formal Methods. Springer, 2005, pp. 109–128.

[52] B. Meyer, "Design by Contract: The Eiffel Method," in TOOLS (26). IEEE Computer Society, 1998, p. 446.

[53] C. Hoare, Communicating Sequential Processes. Prentice Hall, 1985.

[54] K. Gary et al., "Agile Methods for Open Source Safety-critical Software," Software: Practice and Experience, vol. 41, no. 9, 2011, pp. 945–962. [Online]. Available: http://dx.doi.org/10.1002/spe.1075

[55] The Safety Critical Systems Club, "Agile Development for Safety Systems," https://scsc.org.uk/e346.

[56] S. Wolff, "Scrum Goes Formal: Agile Methods for Safety-critical Systems," in Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches. IEEE Press, 2012, pp. 23–29.

[57] S. Black, P. Boca, J. Bowen, J. Gorman, and M. Hinchey, "Formal versus Agile: Survival of the Fittest," Computer, vol. 42, no. 9, 2009, pp. 37–45.

[58] P. Larsen, J. Fitzgerald, and S. Wolff, "Are Formal Methods Ready for Agility? A Reality Check." in FM+ AM, 2010, pp. 13–25.

[59] M. Olszewska, S. Ostroumov, and M. Walden, "Synergising Event-B and Scrum - Experimentation on a Formal Development in an Agile Setting," Abo Akademi University, Tech. Rep. 1152, 2016.

 ISBN: 978-1-61208-458-9

Unifying Modeling and Programming with ALF

Thomas Buchmann and Alexander Rimer
University of Bayreuth
Chair of Applied Computer Science I
Bayreuth, Germany
email: {thomas.buchmann, alexander.rimer}@uni-bayreuth.de

***Abstract*—Model-driven software engineering has become more and more popular during the last decade. While modeling the static structure of a software system is almost state-of-the art nowadays, programming is still required to supply behavior, i.e., method bodies. Unified Modeling Language (UML) class diagrams constitute the standard in structural modeling. Behavioral modeling, on the other hand, may be achieved graphically with a set of UML diagrams or with textual languages. Unfortunately, not all UML diagrams come with a precisely defined execution semantics and thus, code generation is hindered. In this paper, an implementation of the Action Language for Foundational UML (Alf) standard is presented, which allows for textual modeling of software systems. Alf is defined for a subset of UML for which a precise execution semantics is provided. The modeler is empowered to specify both the static structure as well as the behavior with the Alf editor. This helps to blur the boundaries between modeling and programming. Furthermore, an approach to generate executable Java code from Alf programs is presented, which is already designed particularly with regard to round-trip engineering between Alf models and Java source code.**

Keywords–model-driven development; behavioral modeling; textual concrete syntax; code generation.

I. INTRODUCTION

Increasing the productivity of software engineers is the main goal of Model-driven Software Engineering (MDSE) [1]. To this end, MDSE puts strong emphasis on the development of high-level models rather than on the source code. Models are not considered as documentation or as informal guidelines on how to program the actual system. In contrast, models have a well-defined syntax and semantics. Moreover, MDSE aims at the development of *executable* models. Over the years, UML [2] has been established as the standard modeling language for model-driven development. A wide range of diagrams is provided to support both structural and behavioral modeling. Model-driven development is only supported in a full-fledged way, if executable code may be obtained from behavioral models. Generating executable code requires a precise and well-defined execution semantics for behavioral models. Unfortunately, this is only the case for some UML diagrams. As a consequence, software engineers nowadays need to manually supply method bodies in the code generated from structural models.

This leads to what used to be called "*the code generation dilemma*" [3]: Generated code from higher-level models is extended with hand-written code. Often, these different fragments of the software system evolve separately, which may lead to inconsistencies. Round-trip engineering [4] may help to keep the structural parts consistent, but the problem is the lack of an adequate representation of behavioral fragments.

The Eclipse Modeling Framework (EMF) [5] has been established as an extensible platform for the development of MDSE applications. It is based on the Ecore meta-model, which is compatible with the Object Management Group (OMG) Meta Object Facility (MOF) specification [6]. Ideally, software engineers operate only on the level of models such that there is no need to inspect or edit the actual source code, which is generated from the models automatically. However, practical experiences have shown that language-specific adaptations to the generated source code are frequently necessary. In EMF, for instance, only structure is modeled by means of class diagrams, whereas behavior is described by modifications to the generated source code. The OMG standard for the Action Language for Foundational UML (Alf) [7] provides the definition of a textual concrete syntax for a foundational subset of UML models (fUML) [8]. In the fUML standard, a precise definition of an execution semantics for a subset of UML is described. The subset includes UML class diagrams to describe the structural aspects of a software system and UML activity diagrams for the behavioral part.

In this paper, an implementation of the Alf standard is presented. To the best of our knowledge, there is no other realization of the Alf standard, which also allows to generate executable code from corresponding Alf scripts (c.f. Section IV). The currently realized features of the Alf editor are discussed, and some insights on the code generator, which is used to transform Alf scripts into executable Java programs are also given in this paper. The paper is structured as follows: In Section II, a brief overview of Alf is presented. The realization of the Alf editor and the corresponding code generation is discussed in detail in Section III before related work is discussed in the following section. Section V concludes the paper.

II. THE ACTION LANGUAGE FOR FOUNDATIONAL UML

A. Overview

As stated above, *Alf* [7] is an OMG standard, addressing a textual surface representation for UML modeling elements. Furthermore, it provides an execution semantics via a mapping of the Alf concrete syntax to the abstract syntax of the OMG standard of *Foundational Subset for Executable UML Models* also known as *Foundational UML* or just *fUML* [8]. The primary goal is to provide a concrete textual syntax allowing software engineers to specify executable behavior within a wider model, which is represented using the usual graphical notations of UML. A simple use case is the specification of method bodies for operations contained in class diagrams. To this end, it provides a language with a procedural character,

 ISBN: 978-1-61208-458-9

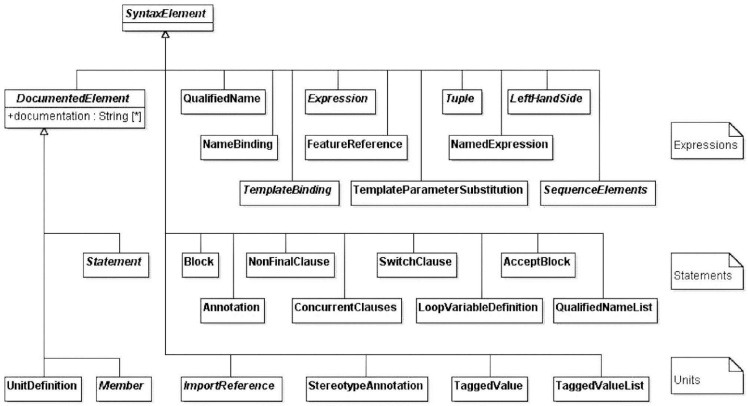


Figure 1: Cutout of the abstract syntax definition of Alf [7]

whose underlying data model is UML. However, Alf also provides a concrete syntax for structural modeling within the limits of the fUML subset. Please note that in case the execution semantics are not required, Alf is also usable in the context of models, which are not restricted to the fUML subset. The Alf specification comprises both the definition of a concrete and an abstract syntax, which are briefly presented in the subsequent subsections.

B. Concrete Syntax

The concrete syntax specification of the Alf standard is described using a context-free grammar in Enhanced-Backus-Naur-Form (EBNF)-like notation. In order to indicate how the abstract syntax tree is constructed from this context-free grammar during parsing, elements of the productions are further annotated.

```
1 ClassDeclaration(d: ClassDefinition)  = [ "abstract" (d.
      isAbstract=true) ] "class" ClassifierSignature(d)
```

Listing 1: Alf production rule for a class [7]

Listing 1 shows an example for an EBNF-like production rule, annotated with additional information. The rule produces an instance *d* of the class ClassDefinition. The production body (the right hand side of the rule) further details the ClassDefinition object: It consists of a ClassifierSignature and it may be abstract (indicated by the optional keyword "abstract").

C. Abstract Syntax

Alf's abstract syntax is represented by an UML class model of the tree of objects obtained from parsing an Alf text. The Alf grammar is context free and thus, parsing results in a strictly hierarchical parse tree, from which the so called abstract syntax tree (AST) is derived. Figure 1 gives an overview of the top-level syntax element classes of the Alf abstract syntax. Each syntax element class inherits (in)directly from the abstract base class SyntaxElement. Similar to other textual languages, the Alf abstract syntax tree contains important non-hierarchical relationships and constraints between Alf elements, even if the tree obtained from parsing still is strictly hierarchical with respect to containment relations. These cross-tree relationships may be solely determined from static analysis of the AST. Static semantic analysis is a common procedure in typical programming languages and it is used, e.g., for name resolving and type checking.

III. Realization

In this section, details of the implementation of the Alf standard are presented. As the UML modeling suite *Valkyrie* [9] is built upon Eclipse modeling technology, and the road map includes the integration of Alf into Valkyrie, EMF [5] and Xtext [10] have been used for the realization. In its current state, the Alf editor adopts most language features of the Alf standard. For the moment, language constructs, which are not directly needed to describe the behavior of method bodies contained in operations specified in class diagrams have been omitted (e.g., some statements like inline, accept or classify statements will be implemented in future work). The main focus has been put on the structural modeling capabilities and the description of behavior of activities plus the generation of executable Java code as these are the mandatory building blocks, which are required to integrate Valkyrie and Alf at a later stage.

A. Meta-model

According to the abstract syntax specification given in the Alf standard, a corresponding Ecore model was created. In its current state, the Alf meta-model comprises more than 100 meta classes and thus, only some relevant cutouts can be presented here due to space restrictions. Please note that the Alf specification provides a model for the abstract syntax of the language. However, due to the tools used to implement the specification, tool-specific adaptations had to be done. Furthermore, some language concepts have been omitted and will be added at a later stage, as described above. Figure 2 depicts the cutout of our realization of the Alf meta-model responsible for the structural modeling aspects of the language.

This part of the meta-model comprises all mandatory meta-classes for Packages, Classifiers, and Features, which are required for structural modeling. The root element of each Alf model is represented by the meta-class Model. A Model contains an arbitrary number of PackageableElements, i.e., Packages and Classifiers. Classifiers may establish an inheritance hierarchy using the meta-class Generalization. Like

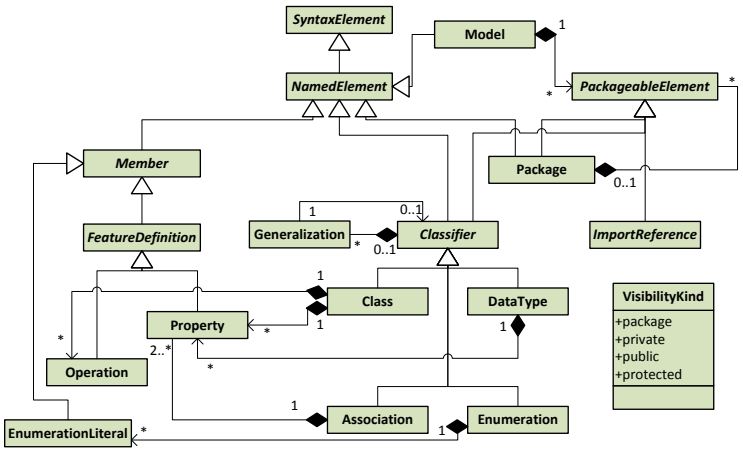


Figure 2: Relevant parts responsible for structural modeling aspects in our realization of the Alf abstract syntax.

UML, Alf supports multiple inheritance between Classifiers. Classifiers are further specialized by the subtypes Class, DataType, Enumeration, and Association. While classes, datatypes and associations contain attributes represented by the meta-class Property, Classes additionally contain Operations.

In Alf, Operations are used for behavioral modeling. Figure 3 depicts a simplified cutout of the Alf meta-model showing the relevant parts. Operations may be parameterized and may contain an "Operation Method". Parameters of an operation are typed and they possess a name. Additionally, the direction of the parameter is indicated by the enumeration ParameterDirectionKind. Possible values are in, out or inout. The type of an operation is determined by its return type. The method of an Operation contains the complete behavior realized by the operation. One possible way of realizing this method is using a Block [7], which represents the body of the operation. The block itself comprises an arbitrary number of Statements.

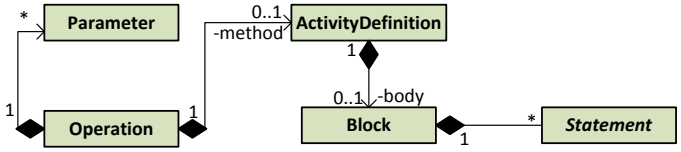


Figure 3: Simplified cutout of the Alf meta-model for Operations.

The Alf standard defines different types of Statements. Figure 4 shows the statements, which are currently realized in our implementation of the Alf standard as subtypes of the abstract class Statement.

Besides statements realizing the return values of an operation (ReturnStatement), several statements dealing with the control flow are included. Local variables may be expressed using the LocalNameDeclarationStatement. The initialization of local variables is done using Expressions, which are encapsulated by ExpressionStatements.

Expressions constitute the most fine-grained way of modeling in Alf and may be used in different contexts. For example, they are used for assignments, calculation, modeling of constraints or the access to operations and attributes. The current

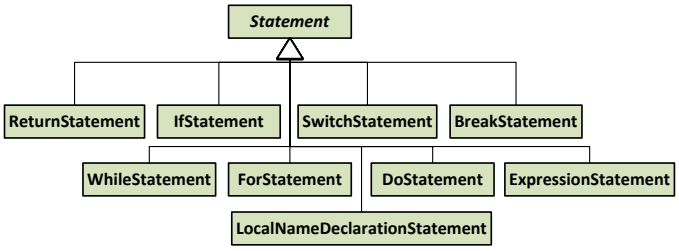


Figure 4: Simplified cutout of the Alf meta-model for Statements.

state of the Alf meta-model comprises various specializations of the meta-class Expression, as depicted in Figure 5.

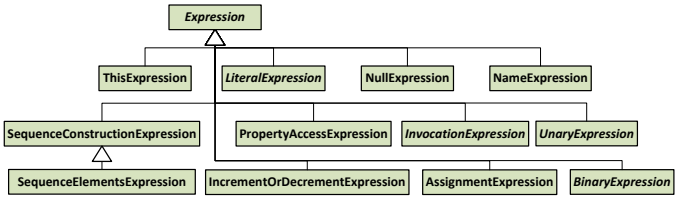


Figure 5: Simplified cutout of the Alf meta-model for Expressions.

LiteralExpression is the superclass for various kinds of literal expressions. They constitute the simplest kind of expressions and may represent, e.g., strings, boolean values or numbers. Literal expressions may be used in combination with assignment expressions or for comparison operations in conditional expressions. While UnaryExpressions, which are used for boolean or arithmetic negations or type queries (instanceof) only contain one operator and one operand, BinaryExpressions have one operator and two operands. Each of the operands may be an Expression as well and they may be used in conditions as well as in assignment expressions.

B. Editor

Xtext [10], an Eclipse framework aiding the development of programming languages and domain-specific languages

(DSLs), has been used to create a textual editor on top of the meta-model described in the previous subsection. Xtext allows for a rapid implementation of languages and it covers all aspects of a complete language infrastructure, like parsing, scoping, linking, validation, code generation plus a complete Eclipse IDE integration providing features like syntax highlighting, code completion, quick fixes and many more. Furthermore, it provides means to easily extend the default behavior of the IDE components using the Xtend [11] programming language. Since Xtext uses ANTLR [12] as a parser generator, there are some restrictions on the grammar, such as that there must not be any left-recursive rules. The Alf standard uses left-recursion in various places and thus, these rules had to be rewritten in order to be used with Xtext. For example this was needed when realizing various Expressions, e.g., expressions used for member access (dot notation).

In order to provide meaningful error messages and modern IDE feature like quickfixes for the end-users of the Alf editor, a set of validation rules has been implemented. Furthermore, validation rules are mandatory to check the static semantics given in the Alf standard. Among others, the validation rules comprise:

- Access restrictions on features with respect to their visibility
- Uniqueness of local variables in different contexts (bodies, control structures and operations)
- Validation of inheritance hierarchies (no cycles)

In order to provide an import mechanism for Alf elements, the scoping rules had to be specified. The default Xtext scoping mechanism calculates visibilities for each AST element, and the result is used, e.g., for linking and for validation of DSL programs. For the Alf editor the scope for different contexts had to be determined: A scope for blocks (in operations and control structures) is needed, furthermore a different scope for realizing feature access (attributes or operations) on classifiers is required (taking into account possible inheritance hierarchies). For example, inherited properties and operations are considered when the scope is determined. In addition, the correct scope for accessing link operations of associations and enumeration literals is also provided.

Figure 6 shows the running Alf editor. The code completion menu depicts possible values, which could be used at the current cursor position. The list of possible matches was computed using the implemented scoping rules as described above.

C. Type System

The Alf specification uses an implicit type system, which allows but does not necessarily require the explicit declaration of typing within an activity. However, static type checking is always provided based on the types specified in the structural model elements. In general, requirements for type systems comprise the following tasks:

- **Type definition:** Various model elements are defined as actual and fixed types (e.g. primitive types).

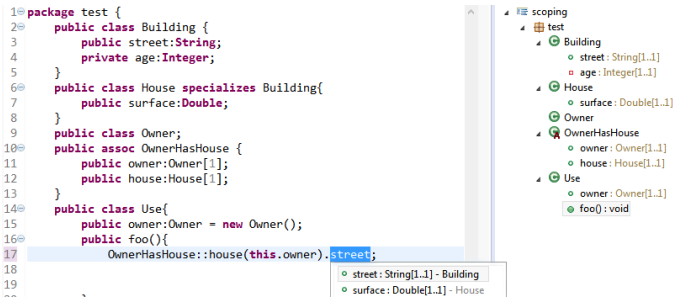


Figure 6: Running Alf editor, showing scoping applied in the code completion menu.

- **Type calculation** A type system should be able to calculate the type of an element and assign a type to an element respectively.
- **Type validation** A type system should provide a set of validation rules, which ensure the well-typedness of all model elements.

In the Xtext context, several frameworks exist, which assist DSL engineers when implementing a type system. For the Alf editor, the tool *Xtext Typesystem (XTS)* [13] has been used, which is optimized for expression-oriented DSLs. XTS provides a DSL, which allows to declaratively specify type system rules for Xtext DSLs. The validation of the type system rules integrates seamlessly into the Xtext validation engine. As an example, Listing 2 shows a simplified cutout of the type definition for a local variable using XTS.

```
1  public class AlfTypesystem extends DefaultTypesystem {
2  private AlfPackage lang = AlfPackage.eINSTANCE;
3  protected void initialize() {
4  //Type definition
5  useCloneAsType(lang.getIntegerType());
6  ... // do the same for all other primitive types
7
8  //Type assignment
9  useTypeOfFeature(lang.getLocalNameDeclarationStatement()
       , lang.getLocalNameDeclarationStatement_Type());
10 useFixedType(lang.getNaturalLiteralExpression(), lang.
       getIntegerType());
11 ... // define other fixed types
12
13 //Type validation
14 ensureOrderedCompatibility(lang.
       getLocalNameDeclarationStatement(),    lang.
       getLocalNameDeclarationStatement_Type(),
15 lang.getLocalNameDeclarationStatement_Expression());
16 }
17 }
```

Listing 2: Type definition of a local variable using XTS

When the type system is initialized, primitive types are defined as clones of their own type (c.f. line 5 in Listing 2). For the meta-class LocalNameDeclarationStatement the assignment of the type is depicted in line 9. The fixed type IntegerType is set for the meta-class NaturalLiteralExpression afterwards before rules for validating the types are specified.

D. Code Generation

In order to execute Alf specifications, they need to be translated into executable source code. In this case, the Alf model acts as platform independent model (PIM), and has

 ISBN: 978-1-61208-458-9

to be transformed into a platform specific one (PSM) first. As future plans for the Alf editor comprise the integration into the UML case tool Valkyrie [9], the MoDisco Java meta-model was chosen as platform specific model. To this end, a uni-directional model-to-model transformation from the Alf model to the MoDisco [14] Java model has been implemented. MoDisco is an Eclipse framework dedicated to software modernization projects. It provides, among others, an Ecore based Java meta-model (resembling the Java AST) and a corresponding discovery mechanism, which allows to create instances of the Java meta-model given on Java source code input. Furthermore, a model-to-text transformation is included allowing to create Java source code from Java model instances.

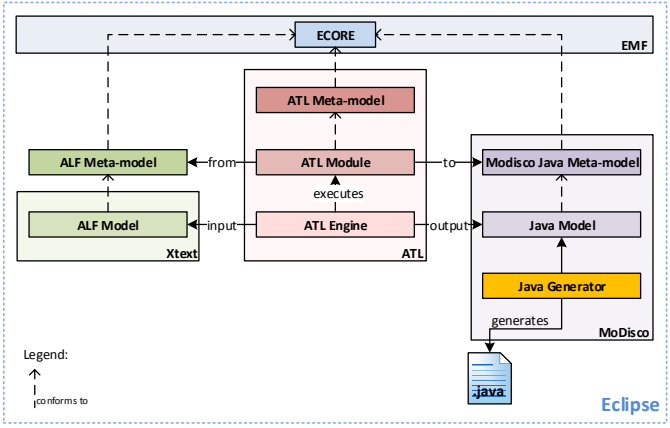


Figure 7: Conceptual design of the code generation using model transformations.

Figure 7 depicts the conceptual design of the code generation engine of the Alf editor. The Atlas Transformation Language (ATL) [15] was used as the model transformation tool, since it works very well for uni-directional model-to-model (M2M) transformations. ATL follows a hybrid approach, providing both declarative and imperative language constructs. Furthermore, ATL offers a concept for module superimposition, allowing to modularize and reuse transformation rules. The M2M transformation implemented for the Alf editor takes the Alf abstract syntax as an input and produces an instance of the MoDisco Java meta-model as an output (c.f. Figure 7). Afterwards, the model-to-text transformation provided by the MoDisco framework is invoked on the resulting output model to generate Java source code files. Please note that writing the ATL transformation rules was a tedious task, since the level of abstraction in Alf is much higher than in Java. For example, a property of an Alf class has to be transformed into a Java field declaration plus corresponding accessor methods. In case of an association, navigability also has to be taken into account and the resulting Java model must contain all (AST-) elements allowing to generate Java code, which ensures consistency of the association ends. In total, the ATL transformation implemented for the Alf editor comprises more than 9000 lines of ATL code distributed over 8 modules.

IV. Related Work

Many different tools and approaches have been published in the last few years, which address model-driven development and especially modeling behavior. The resulting tools rely on textual or graphical syntaxes, or a combination thereof. While some tools come with code generation capabilities, others only allow to create models and thus only serve as a visualization tool.

The graphical UML modeling tool **Papyrus** [16] allows to create UML, SysML and MARTE models using various diagram editors. Additionally, Papyrus offers dedicated support for UML profiles, which includes customizing the Papyrus UI to get a DSL-like look and feel. Papyrus is equipped with a code generation engine allowing for producing source code from class diagrams (currently Java and C++ is supported). Future versions of Papyrus will also come with an Alf editor. A preliminary version of the editor is available and allows a glimpse on its provided features. The textual Alf editor is integrated as a property view and may be used to textually describe elements of package or class diagrams. Furthermore, it allows to describe the behavior of activities. The primary goal of the Papyrus Alf integration is round-tripping between the textual and the graphical syntax and not executing behavioral specifications by generating source code. While Papyrus strictly focuses on a forward engineering process (from model to source code), the approach presented in this paper explicitly addresses round-trip engineering.

Xcore [17] recently gained more and more attention in the modeling community. It provides a textual concrete syntax for Ecore models allowing to express the structure as well as the behavior of the system. In contrast to Alf, the textual concrete syntax is not based on an official standard. Xcore relies on Xbase - a statically typed expression language built on Java - to model behavior. Executable Java code may be generated from Xcore models. Just like the realization of Alf presented in this paper, Xcore blurs the gap between Ecore modeling and Java programming. In contrast to Alf, the behavioral modeling part of Xcore has a strongly procedural character. As a consequence an object-oriented way of modeling is only possible to a limited extent. E.g. there is no way to define object constructors to describe the instantiation of objects of a class. Since Xcore reuses the EMF code generation mechanism [5], the factory pattern is used for object creation. Furthermore, Alf provides more expressive power, since it is based on fUML, while Xcore only addresses Ecore.

Another textual modeling language, designed for *model-oriented programming* is provided by **Umple** [18]. The language has been developed independently from the EMF context and may be used as an Eclipse plugin or via an online service. In its current state, Umple allows for structural modeling with UML class diagrams and describing behavior using state machines. A code generation engine allows to translate Umple specifications into Java, Ruby or PHP code. Umple scripts may also be visualized using a graphical notation. Unfortunately, the Eclipse based editor only offers basic functions like syntax highlighting and a simple validation of the parsed Umple model. Umple offers an interesting approach, which aims at assisting developers in rasing the level of abstraction ("umplification") in their programs [19]. Using this approach, a Java program may be stepwise translated into an Umple script. The level of abstraction is raised by using Umple syntax for associations.

PlantUML [20] is another tool, which offers a textual

 ISBN: 978-1-61208-458-9

concrete syntax for models. It allows to specify class diagrams, use case diagrams, activity diagrams and state charts. Unfortunately, a code generation engine, which allows to transform the PlantUML specifications into executable code is missing. PlantUML uses *Graphviz* [21] to generate a graphical representation of a PlantUML script.

Fujaba [22] is a graphical modeling language based on graph transformations, which allows to express both the structural and the behavioral part of a software system on the modeling level. Furthermore, Fujaba provides a code generation engine that is able to transform the Fujaba specifications into executable Java code. Behavior is specified using *Story Diagrams*. A story diagram resembles UML activity diagrams, where the activities are described using *Story Patterns*. A story pattern specifies a graph transformation rule where both the left hand side and the right hand side of the rule are displayed in a single graphical notation. While story patterns provide a declarative way to describe manipulations of the runtime object graph on a high level of abstraction, the control flow of a method is on a rather basic level as the control flow in activity diagrams is on the same level as data flow diagrams. As a case study [23] revealed, software systems only contain a low number of problems, which require complex story patterns. The resulting story diagrams nevertheless are big and look complex because of the limited capabilities to express the control flow.

V. Conclusion and Future Work

In this paper, an approach to providing tool support for unifying modeling and programming has been presented. To this end, an implementation of the OMG Alf specification [7], which describes a textual concrete syntax for a subset of UML (fUML) [8] has been created. Using the Alf editor, the software engineer may specify both the structure as well as the behavior of a software system on the model level. As a consequence, model transformations may directly be applied to Alf scripts. In order to execute Alf programs, a Java code generator is provided, which allows for the creation of fully executable Java programs and which is already designed particularly with regard to round-trip engineering.

Future work comprises the integration of the (currently) stand-alone Alf editor into the UML modeling tool suite *Valkryie* [9]. To this end, besides integrating textual and graphical modeling, also a mapping of the Alf abstract syntax to the fUML abstract syntax is required as proposed in the Alf standard [7]. Furthermore, a case study is performed in order to evaluate the modeling capabilities of the Alf editor.

References

[1] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[2] OMG, *Unified Modeling Language (UML)*, formal/15-03-01 ed., Object Management Group, Needham, MA, Mar. 2015.

[3] T. Buchmann and F. Schwgerl, "On A-posteriori Integration of Ecore Models and Hand-written Java Code," in *Proceedings of the 10th International Conference on Software Paradigm Trends*, M. v. S. Pascal Lorenz and J. Cardoso, Eds. SCITEPRESS, July 2015, pp. 95–102.

[4] T. Buchmann and B. Westfechtel, "Towards Incremental Round-Trip Engineering Using Model Transformations," in *Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2013)*, O. Demirors and O. Turetken, Eds. IEEE Conference Publishing Service, 2013, pp. 130–133.

[5] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF Eclipse Modeling Framework*, 2nd ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[6] OMG, *Meta Object Facility (MOF) Core*, formal/2011-08-07 ed., Object Management Group, Needham, MA, Aug. 2011.

[7] OMG, *Action Language for Foundational UML (ALF)*, formal/2013-09-01 ed., Object Management Group, Needham, MA, Sep. 2013.

[8] OMG, *Semantics of a Foundational Subset for Executable UML Models (fUML)*, formal/2013-08-06 ed., Object Management Group, Needham, MA, Aug. 2013.

[9] T. Buchmann, "Valkyrie: A UML-Based Model-Driven Environment for Model-Driven Software Engineering," in *Proceedings of the 7th International Conference on Software Paradigm Trends (ICSOFT 2012)*, Rome, Italy, 2012, pp. 147–157.

[10] "Xtext project," http://www.eclipse.org/Xtext, visited: 2015.09.30.

[11] "Xtend project," http://www.eclipse.org/xtend, visited: 2015.09.30.

[12] T. Parr and K. Fisher, "LL(*): the foundation of the ANTLR parser generator," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, M. W. Hall and D. A. Padua, Eds. ACM, 2011, pp. 425–436. [Online]. Available: http://doi.acm.org/10.1145/1993498.1993548

[13] L. Bettini, D. Stoll, M. Völter, and S. Colameo, "Approaches and tools for implementing type systems in xtext," in *Software Language Engineering, 5th International Conference, SLE 2012, Dresden, Germany, September 26-28, 2012, Revised Selected Papers*, ser. Lecture Notes in Computer Science, K. Czarnecki and G. Hedin, Eds., vol. 7745. Springer, 2012, pp. 392–412. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36089-3_22

[14] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: a generic and extensible framework for model driven reverse engineering," in *Proceedings of the IEEE/ACM International Conference on Automated software engineering (ASE 2010)*, Antwerp, Belgium, 2010, pp. 173–174.

[15] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Science of Computer Programming*, vol. 72, pp. 31–39, 2008, special Issue on Experimental Software and Toolkits (EST).

[16] A. Lanusse, Y. Tanguy, H. Espinoza, C. Mraidha, S. Gerard, P. Tessier, R. Schnekenburger, H. Dubois, and F. Terrier, "Papyrus UML: an open source toolset for MDA," in *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*. Citeseer, 2009, pp. 1–4.

[17] "Xcore," http://wiki.eclipse.org/Xcore, visited: 2015.09.30.

[18] "Umple Language," http://cruise.site.uottawa.ca/umple/, visited: 2015.09.30.

[19] T. C. Lethbridge, A. Forward, and O. Badreddin, "Umplification: Refactoring to incrementally add abstraction to a program," in *Reverse Engineering (WCRE), 2010 17th Working Conference on*. IEEE, 2010, pp. 220–224.

[20] "PlantUML," http://plantuml.com/, visited: 2015.09.30.

[21] "Graphviz," http://www.graphviz.org, visited: 2015.09.30.

[22] The Fujaba Developer Teams from Paderborn, Kassel, Darmstadt, Siegen and Bayreuth, "The Fujaba Tool Suite 2005: An Overview About the Development Efforts in Paderborn, Kassel, Darmstadt, Siegen and Bayreuth," in *Proceedings of the 3rd international Fujaba Days*, H. Giese and A. Zündorf, Eds., September 2005, pp. 1–13.

[23] T. Buchmann, B. Westfechtel, and S. Winetzhammer, "The added value of programmed graph transformations — a case study from software configuration management," in *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2011)*, A. Schürr, D. Varro, and G. Varro, Eds., Budapest, Hungary, 2012.

 ISBN: 978-1-61208-458-9

A Systematic Approach to Assist Designers in Security Pattern Integration

Loukmen Regainia*, Cédric Bouhours† and Sébastien Salva‡
LIMOS - UMR CNRS 6158
Auvergne University, France
Email: * loukmen.regainia@udamail.fr, † cedric.bouhours@udamail.fr, ‡ sebastien.salva@udamail.fr

***Abstract*—The last decade has witnessed significant contributions in software engineering to design more secure systems and applications. Software designers can now leverage specific patterns, called *security patterns* as reusable solutions to model more secure applications. But, despite the advantages offered by security patterns, these are rarely used in practice, because choosing and employing them for devising less vulnerable applications, is still a difficult and error-prone task. In this work, we propose an original approach to guide designers for checking whether a set of security patterns is correctly integrated into models and if vulnerabilities are yet exposed despite their use. This approach relies upon the analysis of the structural and behavioral properties of security patterns and on formal methods to check if these properties hold in the application model completed with patterns. We also provide a metric computation to assess the integration quality of patterns. Afterwards, we check whether the vulnerabilities, which should be removed by the use of patterns, are not exposed in the model. We illustrate this approach on an example of Web application, the *Moodle education platform*.**

***Keywords*—Model; UML; Security Patterns; Verification**

I. Introduction

Despite the indisputable improvements recently made in modeling, coding and testing, software engineering is still regarded as a complex field. One reason for this complexity is well-known: software engineering must not only address the functional aspects of an application, but also have to cover other aspects such as security. Indeed, providing secure models and code is recognized as an important factor of quality, but on the other hand, devising them is a difficult task.

To solve this issue, a large set of papers and tools have been proposed to help the integration of security in the software engineering steps [1][2]. Among them, the pattern community proposed the notion of security patterns as reusable solutions to security issues in the modeling stage [3]. Specifically, a pattern represents a structure, a behavior, or some intents that have to be applied in models to meet security properties or to prevent threats (sometimes partially). At present, the security pattern base holds hundreds of available patterns, more and less detailed and compatible with each other. Many of them are indeed described with text only, their contextualization (a.k.a. instantiation) being left to designers. Furthermore, the impact of their composition is often unknown. Hence, the choice of the relevant patterns and their inclusion in models is yet onerous and error-prone, even for experts. This paper focuses on these difficulties and proposes an approach for helping designers to integrate the appropriate patterns and to design more secure applications. This approach corresponds to a sequential process, whose main benefits are twofold: measuring the integration quality of security patterns in models, and checking if these models are yet vulnerable to attacks despite the use of patterns.

More precisely, the designer initially chooses a list of vulnerabilities that must not be found in the application model and a set of security patterns that should correct these vulnerabilities or prevent attackers from exploiting them. The proposed formal method-based process firstly aims at helping the designer to integrate each pattern in the application model: we check whether the structural and behavioral properties of the pattern hold in the model by means of a method based on OCL (Object Constraint Language) queries and a verification technique. The former tries to locate the pattern shape in the model and returns a coefficient of disclosure. The verification technique checks whether some behavioral properties of the security pattern, expressed with LTL (Linear Temporal Logic) formulas, hold in the model. Then, quality metrics are computed to evaluate the integration quality of each pattern and of all the patterns in the model. If these metrics are low, the model should be revisited. In a second stage, the process also checks whether each vulnerability can be detected in the model by means of a verification technique. Actually, despite the use of security patterns, a vulnerability may be still present on account of several reasons, e.g., the use of an incomplete or wrong list of security patterns, or the composition of several patterns in the model that may induce a vulnerability. Vulnerabilities are formally expressed with LTL properties that are given to a model-checker. If any property is satisfied in the model, the designer is then warned that the latter still includes vulnerabilities and requires modifications.

We illustrate the benefits of this approach by applying it on a case study, based on Moodle education platform and on the vulnerability *Code injection*, which is a well known flaw allowing to inject code that is then interpreted by the application [4].

The paper is structured as follows: we briefly present the background and motivations in Section II. The approach is described in Section III. Its illustration on a Web application example is given in Section IV. Finally, we draw conclusions and perspectives for future work in Section V.

II. Background

As the number of available security patterns is continuously growing, choosing the appropriate ones, to design more secure applications, is more and more tedious and error-prone. To help designers in this task, several papers proposed classifications and taxonomies to organize security patterns. Based on the STRIDE threat management methodology, Munawar et

 ISBN: 978-1-61208-458-9

al. presented an organization of 97 security patterns on three architectural layers (Core, Exterior, Perimeter) [5]. In addition, Alvi et al. proposed in [6] a natural scheme for classifying security patterns. They associated their classification to the phases of software life cycle, i.e., security objectives in the requirement phase, security properties in the design phase, and attack patterns in the implementation phase.

The above papers provide classifications to help designers in the choices of security patterns. But, they do not help check whether patterns are correctly included in models. They also do not ensure that models are indeed more secure.

For the first point, Konrad et al. introduced in [7] a security pattern template to ease pattern integration. They exposed the difficulties related to the lack of comprehensive and formal description and proposed a template composed of Unified Modeling Language (UML) diagrams and LTL properties. Actually, we assume in our approach, that security patterns are indeed described with this kind of template.

In the last decade, several papers also proposed methods to check if UML models meet security requirements, the latter being usually expressed with LTL properties [8]-[9]. For instance, Tanvir, et al. proposed an approach to verify the impact of using Role Based Access Control (RBAC) in a Computer Supported Cooperative Work (CSCW) [8]. Thereby, they showed how to formally check if an application meets security requirements.

Our approach proposes to consider both above aspects, i.e., pattern integration assistance and verification of security properties on UML models, inside a whole process. We check pattern integration in UML models by considering their structural and behavioral properties. We also define and assess their integration quality with metrics. If these are low, the designer is warned that the UML diagrams of the application should be revisited. Afterwards, we also check, by means of model-checking, that the chosen security patterns actually remove undesired vulnerabilities from the UML models. This step aims to attest that the chosen patterns are effective against undesired vulnerabilities, or to warn designers to chose other patterns.

III. Model securing with pattern integration aided by formal techniques

In this section, we present our approach to assist designers to integrate security patterns in models for devising more secure applications, as illustrated in Figure 1. In this paper, we assume having a UML model M of the application, a vulnerability set V, which must not be exploited by attackers in the application, and a security pattern set $Sp = \{Sp_1, ..., Sp_k\}$, which should prevent the vulnerabilities of $V = \{V_1, ..., V_l\}$ from being exploited. We also assume having a base of generic formal properties P_{V_i} and P_{Sp_j} describing V_i and Sp_j. The approach illustrated in Figure 2 aims at checking whether each pattern Sp_j is correctly integrated on M and if M still has the vulnerability V_i despite the use of Sp.

As the patterns are described in a generic, abstract form, the first step for the designer is to instantiate every pattern

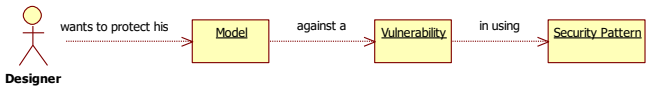


Figure. 1. Context of the proposed approach

to the context of his model. Instantiating a pattern consists in adapting each constituting element to the specific context of a model. It is a complex step, which requires expertise in order to not damage the pattern. Hence, this step has to be verified:

1) Given a pattern Sp_j, we check if the structural and behavioral properties of Sp_j hold in the model M. For the structural properties, we use the approach and tool we developed in [10]. Given the pattern Sp_j and its generic description, the tool automatically derives a set of OCL queries encoding the structure of the pattern. Then, we call an executor of procedural OCL queries [11]. After the execution, all micro-architectures, subsets of the model, looking like the pattern are listed and a coefficient of disclosure C_{Sp_j}, ranging between 0 and 1, is given. The one with the highest coefficient is taken into account to locate where the pattern has been integrated in the model. If the designer does not agree, two cases are possible. On one hand, the pattern instantiation is incorrect and should be modified. On the other hand, the context is specific enough to justify a change in the structure. In any case, if the designer does not consider that the pattern is structurally integrated, the next step cannot be reached.
2) This step consists in checking whether some behavioral properties of Sp_j hold in M. We provide, with the pattern, a set of generic behavioral properties P_{Sp_j}, described in LTL. These rules describe the sequential message exchanges between the methods and the temporal states of the objects. These properties are generic and should be manually instantiated by the designer. Once instantiated, M is automatically translated into a Promela (PROtocol MEta LAnguage) specification with the HugoRT tool [12] and we check if this specification satisfies the previous LTL formulas with the model-checker Spin [13]. This indicates whether the pattern integration into the model respects behavior imposed by the pattern.
3) Quality metrics are now computed to measure the integration quality of the pattern Sp_j of Sp into the model M with regard to the coefficient of disclosure C_{Sp_j} and the LTL property set P_{Sp_j} provided with Sp_j. Intuitively, the closer C_{Sp_j} is to 1 and the larger the set P_{Sp_j}, the more accurate the estimation of the pattern integration is. We define the mapping $m : P_{Sp} \rightarrow \{0, 1\}$ by $m(p) = 1$ iff $M \models p$, and $m(p) = 0$ otherwise. The estimation range of a pattern Sp_j integration is between 0 and $n = Card(P_{Sp_j}) + 1$ and this first integration metric is defined as:

 ISBN: 978-1-61208-458-9

$$0 \leq m(Sp_j) = \sum_{p_i \in P_{Sp_j}} m(p_i) + C_{Sp_j} \leq n$$

Afterwards, we compute the overall integration quality of the security pattern set $Sp = \{Sp_1, ..., Sp_k\}$ using an utility function U. With this aim, we take *Simple Additive Weighting* (SAW) [14] which allows to adjust the integration estimations of each pattern having different ranges by a weight representing user preferences and priorities. In our case U is defined as:

$$U = 0 \leq \sum_{i=1}^{k} m(Sp_i)/(Card(P_{Sp_i}) + 1).w_i \leq 1$$

with $w_i \in \mathbb{R}_0^+$ and $\sum_{i=1}^{k} w_i = 1$, w_i being the weight of Sp_i to represent designer preferences. The closer U is to 1, the greater the estimation that security patterns $Sp_1, ..., Sp_k$ are correctly integrated into M. A good integration quality U can be reached with few properties. With a small number of properties, the pattern is not well documented / defined. In contrast, if the pattern integration is defined with large property sets and U is close to 0, this means that a mistake has been made during the instantiation or that the context is specific enough to justify a change in the structure of the pattern.

4) The last step of our approach starts when the designer considers that the pattern is integrated with a sufficient quality. For the vulnerability $V_i \in V$, we have a set of generic LTL properties P_{V_i} describing V_i, i.e., undesired behavior that should never happens. Once more, the designer has to instantiate these properties in accordance with the model M. The Spin tool is now called to check that the Promela specification of M will never satisfy the undesired behavior expressed by P_{V_i}. If the presence of a vulnerability is detected, a counter-example is returned by Spin to detail the origin of the violation of the property. Hence, guided by the counter-example, the designer has to ensure that the chosen patterns are effective against the vulnerability or that the combination of patterns does not induce flaws.

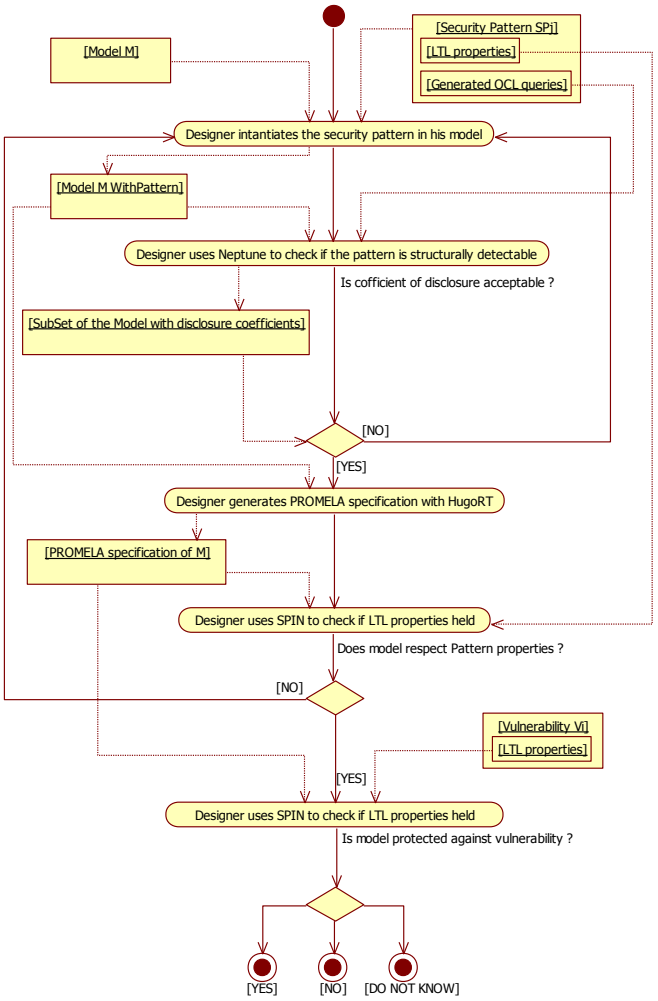


Figure. 2. An approach to assist designers to devise more secure applications

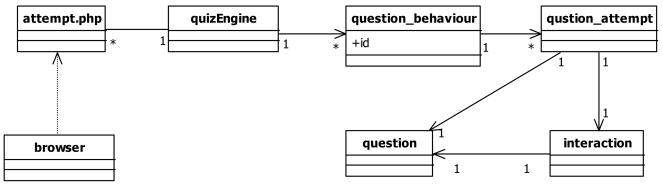


Figure. 3. Moodle Quiz engine class diagram

IV. Case Study

In this section, we describe, with more details, the steps of the proposed approach through a Web application example, the Moodle education platform [15], and precisely on its exam (quiz) functionality. The UML class diagram of this functionality is given in Figure 3 and its sequence diagram in Figure 4.

A user is identified with an *id* and accesses the service with a request of the form "GET /moodle/quiz/attempt.php?id=123". The identifier is used by the quizEngine as a parameter to get information about the user profile so that he/she has only one opportunity to pass an exam. An SQL database is used to get the user profile information and creates an exam session with regard to the requested id.

It is well-known that this kind of Web applications is usually exposed to threats related to *input ports 'passing illegal data'*, and especially injection attacks [4]. For instance, an attacker may exploit a vulnerability to pass an exam many times by spoofing or forging identities stored in database through SQL injection attacks. We have chosen to take as example here a familiar vulnerability called *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*, which is the main reason of SQL Injection attacks and one of the most recurrent vulnerabilities [16].

In order to secure the Web application, we have chosen to

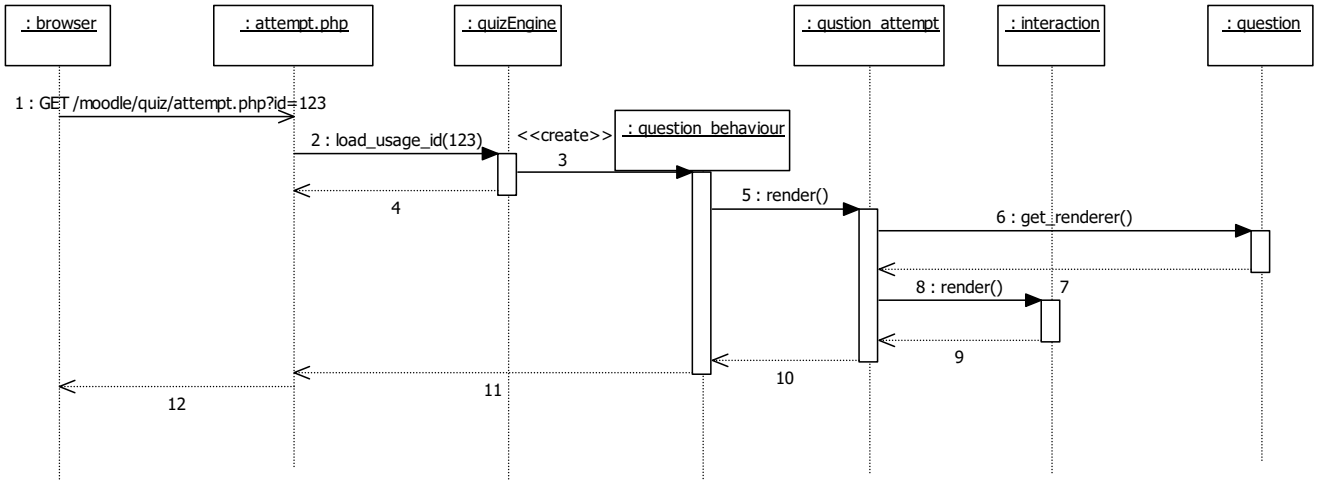


Figure. 4. Moodle Quiz engine sequence diagram

use the *Intercepting Validator* security pattern whose UML class diagram is illustrated in Figure 5. This pattern is indeed presented as a solution to prevent attackers from exploiting the above vulnerability [17]. The intent of this security pattern is to validate every user input request before using it as a parameter by a dynamically loadable validation logic [17].

According to the *Intercepting Validator* security pattern documentation, its behavior is highlighted most notably by the following properties:

1) a validation logic for every data-type used in the application,
2) a single mechanism to validate all data-types,
3) the separation of the validation logic from the presentation logic,

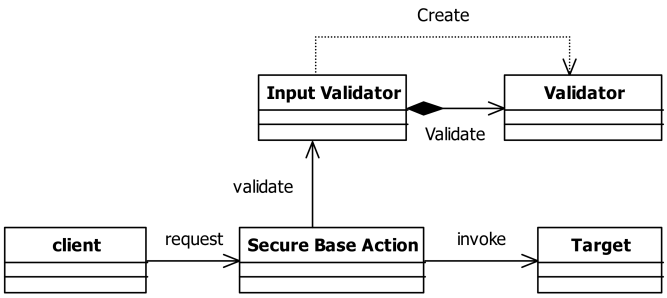


Figure. 5. Intercepting Validator class diagramm

A. Security pattern integration

We instantiated the *Intercepting Validator* security pattern on the model of the Moodle QuizEngine application by adapting its structural and behavioral properties in concordance with the diagrams in Figures 3 and 4. The pattern classes are firstly added between *attempt.php* and *quizEngine* in the QuizEngine class diagram to prevent from SQL Injections through *quizEngine*, which has access to the database. The resulting class diagram is depicted in Figure 6. The sequence diagram of the QuizEngine application is also extended to include the security pattern behavior. The resulting diagram, given in Figure 7, shows that the messages exchanged between *attempt.php* and *quizEngine* are now validated.

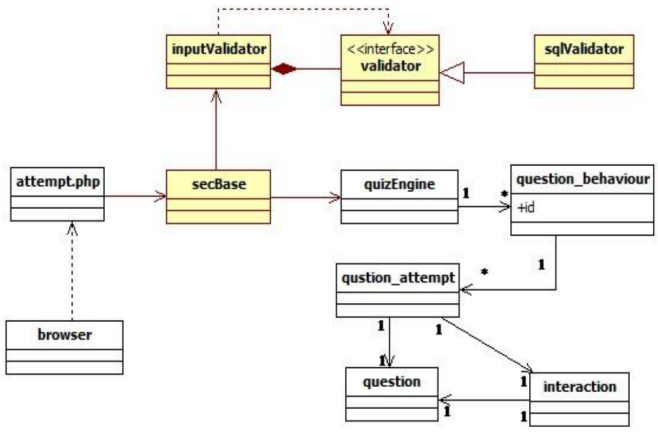


Figure. 6. Instantiation of the security pattern: class diagramm

B. Security pattern instantiation assessment

In this step, we check if the structural and behavioral properties of the security pattern hold in the application model (steps 1, 2 of the approach).

Firstly, we use the method we developed in [10] to extract the pattern structural properties, expressed with OCL queries. Then, we call the tool Neptune [11] to return a list of pairs $(v, coef)$ with v a vertex of the model that is also a vertex of the pattern and $coef$ a coefficient of disclosure. With the class diagram of Figure 6, the tool provides the class "SecBase"

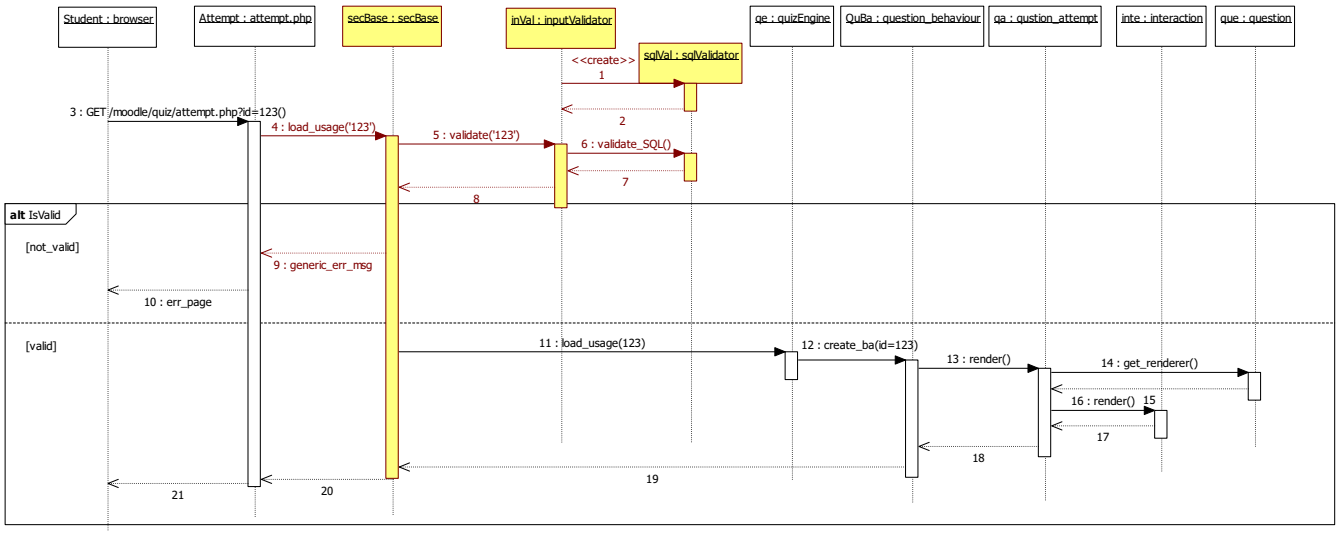


Figure. 7. Instantiation of the security pattern: sequence diagramm

and a coefficient equal to 1. This means that the pattern structure has been completely found only one time. With more complex diagrams, the designer may check if the tool has indeed recognized the pattern or has revealed a diagram part that looks like the pattern. He/she can change the class diagram if required.

For the behavioral properties, we assume having a set of generic LTL properties describing the *Intercepting Validator* pattern. We assume having three generic properties here, given in Table I col2 have to be manually instantiated to meet the QuizEngine application model. These ones are given in Table I col3.

For example, the property *"A validation logic for every data-type used in the application"* given above, is formalized with the LTL formula p_1, which means *"For every Client input, we do not validate data until the creation of the matching validator (SQL, XML, LDAP, etc.)"*.

$$p_1 : \Box\ (Clientinput(Data) \rightarrow !Validate(data)$$

$$\mathcal{U}\ createValidator(Data.type))$$

This generic formula is instantiated with respect to the QuizEngine application context, found in the sequence diagram of Figure 7. The instantiation of the property implies the good choice of the events matching the facts addressed by the generic LTL formula. For example, the fact $Clientinput(Data)$ corresponds to the state input (the arrival of the request "GET /moodle/quiz/attempt.php?id=123") of the object *attempt.php*. The LTL formula becomes :

$$p_1 : \Box\ (attempt.inState(input) \rightarrow$$

$$!secBase.inState(WaitingVal)\ \mathcal{U}\ inVal.inState\ (valCreated))$$

To check whether the LTL formula p_1, p_2, p_3 hold in the model of Figure 7, we then performed the two following steps:

1) The UML diagram is translated into a Promela specification, with the HugoRT tool [12],
2) The Spin model-checker [13] is called to check whether the Promela specification satisfies the LTL formulas. Namely, Spin checks that the Promela specification never ends in a state corresponding to a counterexample of one of the properties p_1, p_2, p_3.

In this example, all the properties of Table I hold. The pattern integration quality can now be straightforwardly estimated by combining the results obtained from the previous steps. Here, the estimation of the pattern integration is given by the metrics $0 \leq m(Sp_1) = 1+1+1+1 \leq 4$ and $0 \leq U = 1 \leq 1$. The latter shows that the security pattern is well integrated compared to the number of available behavioral properties. In contrast, the metric $m(Sp)$ also reveals that the upper bound of the metric range (n) is low. This means that the number of behavioral properties is modest, and perhaps insufficient to ensure that the pattern is really correctly integrated.

C. Vulnerability exposure assessment

The last step aims at confirming that the security vulnerability is no longer exposed in the application model. We also assume having a set of LTL generic properties expressing behaviors that should never happen. For the CWE-89 vulnerability, taken as example in the paper, its documentation provides the following properties [4]: v_1: No input validation, v_2: Bad input validation, v_3: Privilege escalation, v_4: Remote information inference. This undesired behavior is formalized with the LTL formulas given in Table II col.2. These formulas also have to be manually instantiated with respect to the context of the application. The resulting formulas are given in Table II col.3. With the UML diagram of Figure 6, one can deduce that the event *clientInput* corresponds to the arrival of information from *attempt.php*, and *invokeTarget* corresponds

 ISBN: 978-1-61208-458-9

TABLE I
INTERCEPTING VALIDATOR LTL PROPERTIES

P	LTL generic form	LTL instantiated form LTL
p_1	$\Box(Clientinput(Data) \rightarrow !Validate(Data)\ \mathcal{U}\ createValidator(data.type))$	$\Box\ (attempt.inState(input)) \rightarrow (!\ secBase : inState\ (WaintingValidation)\ \mathcal{U}\ inVal.inState\ (validators\ Created)))$
p_2	$\Box(inputValdiator.isUnique)$	$\Box(secBase.isUnique \wedge inVal.isUnique)$
p_3	$\Box(clientInput(data) \wedge !ServerValidate(data) \wedge \Diamond ServerValidate(data)) \rightarrow (!returnGeneric(message)\ \mathcal{U}\ ServerValidate(data))$	$\Box((attempt.inState(input) \wedge !secBase.inState(nonvalid) \wedge \Diamond secBase.inState(nonvalid)) \rightarrow !attempt.inState(err\_page)\ \mathcal{U}\ secBase.inState(nonvalid))$

TABLE II
CWE-89 VULNERABILITY PROPERTIES

Vulnerability property	LTL generic formula	LTL instantiated formula	Sat
v_1	$\Box(clientInput(data) \rightarrow \Diamond invokeTarget(data))$	$\Box(attempt.inState(input) \rightarrow \Diamond quizEngine.inState(loadUsage))$	No
v_2	$\Box(clientInput(data) \rightarrow \Box(!Valid(data) \rightarrow \Diamond invokeTarget(data)))$	$\Box(attempt.inState(input) \rightarrow \Box(secBase.inState(nonvalid) \rightarrow \Diamond(quizEngine.inState(loadUsage))))$	No
v_3	$\Box(clientInput(data) \wedge client.right(Min) \rightarrow \Diamond client.right(Max))$	?	?
v_4	$\Box(!valid(data) \rightarrow \Diamond(!genMessage))$	$\Box(secBase.inState(nonvalid) \rightarrow \Diamond\ (!attempt.inState(err\_page)))$	Yes

to the use of the input data in the object *quizEngine*. With the sequence diagram of Figure 6, one deduces that the fact *invokeTarget(Data)* of the generic formulas has to be replaced by the state *loadUsage* of the object *attempt.php*.

For example, the property *No input validation* is formulated with the generic LTL formula v_1 in Table II, which intuitively means "for every client input (data) the target is eventually invoked with (data) as parameter". This formula reflects undesired behavior because client inputs always have to be validated before any invocation. The instantiation of the generic formula v_1 gives:

$$v_1 : \Box(attempt.inState(input) \rightarrow \Diamond\ quizEngine.inState(loadUsage))$$

During the generic formula instantiation, we observed that the third property cannot be deduced. Indeed, this property, which is related to privilege escalation through SQL injection cannot be expressed with the events found in the diagram of Figures 6 and 7. This means that the application model does not include the required features for exposing this property. Here, the notion of level of access is indeed not represented.

We now call the Spin model-checker to check the absence of vulnerabilities in the QuizEngine application model. We obtain the results listed in Table II col.4. Spin detects the presence of the vulnerability property v_4, and provides a counter-example. This property means that in the case when the client input is not valid, the generic error messages (whose content is minimal) is not sent to the client. Thus, the client may get a more detailed error message with sensitive information about the application. This information may be used to deduce attack vectors with a remote information inference.

As a consequence, the QuizEngine application still exposes the CWE-89 vulnerability and this step reveals that the *Intercepting Validator* pattern is insufficient to not expose this vulnerability. After analysis of the counter-example, we deduced that the vulnerability was detected because there is no mechanism, expressed in the security pattern, to generate generic error messages in the case of invalid input messages. Indeed, the *Intercepting Validator* pattern validates and filters input messages only. Another security pattern is therefore required, for instance the *Exception Shielding* pattern [5].

In conclusion, all this process helped integrate the security pattern and showed that this one was not sufficient to secure the application against all the threats related to the CWE-89 vulnerability. Either a more appropriate pattern has to be taken, or another pattern has to be combined with *Intercepting Validator*.

V. CONCLUSION

This paper presents a model-based process for helping designers to devise more secure applications by checking the appropriate use and contextualization of security patterns within UML diagrams. In an initial step, we assume that the designer chooses a list of vulnerabilities that must not appear in the application and a list of security patterns, which should prevent these vulnerabilities from being exploited. The proposed approach then provides several automatic or manual steps to ensure whether security patterns are correctly integrated and if vulnerabilities are still exposed despite the use of security patterns. A coefficient of disclosure is computed for every pattern and assesses if its structure can be found in the application model. Then, a quality metric is

 ISBN: 978-1-61208-458-9

computed to estimate the integration quality of the patterns with regard to a set of available properties expressing the pattern behaviors. Theses metrics guide the designer in the correct pattern integration. The last step of the process aims at warning the designer if a vulnerability is still exposed. We have illustrated this approach with an example of Web application, which has to be protected against the CWE-89 vulnerability related to *Code Injection*.

In the near future, we intend to automate more this process to make its application easier to use for designers. Automation sounds typically not applicable to the entire process but to specific steps, e.g., the choice of the security patterns from vulnerabilities, or the instantiation of LTL properties from the application model. Hence, a designer having a very limited knowledge about security patterns and formal verification could improve the security of its applications anyway. To achieve such an automatic process, an initial step will consist of building an exhaustive base of formal vulnerabilities and security patterns providing the relationship between them.

VI. Acknowledgment

Research supported by the industrial chair on Digital Confidence http://confiance-numerique.clermont-universite.fr/index-en.html

References

[1] H. Mouratidis, P. Giorgini, and G. Manson, "When security meets software engineering," *Inf. Syst.*, vol. 30, no. 8, pp. 609–629, Dec. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.is.2004.06.002

[2] I. Flechais, C. Mascolo, and M. A. Sasse, "Integrating security and usability into the requirements and design process," *Int. J. Electron. Secur. Digit. Forensic*, vol. 1, no. 1, pp. 12–26, May 2007. [Online]. Available: http://dx.doi.org/10.1504/IJESDF.2007.013589

[3] N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," *Progress in Informatics*, vol. 5, pp. 35–47, Mar. 2008.

[4] Common weakness enumeration. [Online]. Available: https://cwe.mitre.org/

[5] Security pattern catalog. [Online]. Available: http://www.munawarhafiz.com/securitypatterncatalog/

[6] A. K. Alvi and M. Zulkernine, "A Natural Classification Scheme for Software Security Patterns," *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 113–120, 2011. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6118361

[7] S. Konrad, B. H. Cheng, L. a. Campbell, and R. Wassermann, "Using Security Patterns to Model and Analyze Security Requirements," *2nd International Workshop on Requirements Engineering for High Assurance Systems*, pp. 13–22, 2003.

[8] T. Ahmed and A. R. Tripathi, "Static verification of security requirements in role based CSCW systems," *Proceedings of the eighth ACM symposium on Access control models and technologies - SACMAT '03*, p. 196, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=775412.775438

[9] M. Al-lail, R. Abdunabi, R. B. France, and I. Ray, "An Approach to Analyzing Temporal Properties in UML Class Models," pp. 77–86, 2013.

[10] C. Bouhours, H. Leblanc, C. Percebois, and T. Millan, "Detection of generic micro-architectures on models," in *Proceedings of PATTERNS 2010, The Second International Conferences on Pervasive Patterns and Applications*, Lisbon, Portugal, 21st - 26th November 2010, pp. 34–41.

[11] T. Millan, L. Sabatier, T. T. Le Thi, P. Bazex, and C. Percebois, "An ocl extension for checking and transforming uml models," in *proceedings of the 8th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS)*. http://www.wseas.org/: WSEAS Press, 2009, pp. 144–150, (Invited speaker).

[12] S. Merz and C. Rauh, "Model checking timed uml state machines and collaborations," in *7th Intl. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 2002*, 2002, pp. 395–414.

[13] G. Holzmann, *Spin Model Checker, the: Primer and Reference Manual*, 1st ed. Addison-Wesley Professional, 2003.

[14] K. P. Yoon and C.-L. Hwang, "Multiple attribute decision making: An introduction (quantitative applications in the social sciences)," 1995.

[15] Overview of the moodle question engine. [Online]. Available: https://docs.moodle.org/dev/Overview_of_the_Moodle_question_engine

[16] OWASP, "Owasp testing guide v3.0 project," in *http://www.owasp.org/index.php/Category:OWASP_Testing_Project#OWASP_Testing_Guide_v3*, 2003.

[17] C. Steel, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall PTR, 2005.

 ISBN: 978-1-61208-458-9

Function Point Analysis with Model Driven Architecture Applied on Frameworks of Partial Code Generation

Rodrigo Salvador Monteiro
Instituto de Computação
Universidade Federal Fluminense, UFF
Niterói, Brasil
e-mail: salvador@ic.uff.br

Roque Pinel, Geraldo Zimbrão
and Jano Moreira de Souza
COPPE / UFRJ
Rio de Janeiro, Brasil
e-mail: {repinel,zimbrao,jano}@cos.ufrj.br

***Abstract*— Software measurement is a crucial task for the planning and the developing of information systems. The Function Point Analysis (FPA) was developed to measure the complexity of the functionality of systems. Its methods are independent of technology and can be applied directly to the specification of features and the domain. However, the counting should be performed by a metrics analyst, being under subjectivity, wasting time and a large number of resources. This article describes the proposal of automation of function point counting performed using Unified Modeling Language (UML) models and Model Driven Architecture (MDA) methodology. Our approach provides a standard method for counting based on the International Function Point Users Group (IFPUG), eliminating the subjectivity present in traditional procedures. The work counts the number of realized function points, based on the information system already developed. The counting of function points achieved allows for transparency to the client receiving the product besides the construction of an important historical base for the refinement of future estimates.**

Keywords-MDA; Metric; Function Point Analysis; AndroMDA; MDArte.

I. INTRODUCTION

With the increasing complexity of information systems, measuring their features is a crucial task for software projects. Estimation reports become common documents of the customer relationship, essential to development planning and organizing tasks. The Function Point Analysis (FPA) was defined in 1979 as a procedure capable of measuring the functionality and complexity of information systems [1]. It is performed based on the specifications of features and the domain, defined as independent of technology, unlike other metrics, such as Lines of Code (LOC), that depends on the programming language used. The method of function points was developed in order to deliver the customer a measure on the logic functions in the system, based on specifications. Therefore, the metrics analyst should study the documentation and count the number of points. Despite the efforts of the International Function Point Users Group (IFPUG) [2] to establish a standard for counting, this value is still under the subjectivity of the analyst. Moreover, this process is known to require hours of hard work and dedication, being a large consumer of resources.

Aligned with the independence of technology, we have the Unified Modeling Language (UML) [3], which allows projects to have a standard graphical representation. It has been widely used in information systems specifications, being the main source for the establishment of the Function Point Analysis. However, although it is helpful, the analysis of documents continues to be cumbersome and time-consuming.

In 2001, the Object Management Group (OMG) released a guide of definitions about code generation based on models, the Model Driven Architecture (MDA) [4]. This methodology uses, among other standards, UML as a modeling language. Its methods allow the automation of the life cycle of projects based on UML models, reducing development time and allowing the standardization of the system code. The MDA approach provides an environment ripe for introduction of automatic FPA.

In particular, we explore the use of the framework MDArte [5], an extension of the AndroMDA framework [6], to automate the process, extract values from models and generate artifacts useful in the FPA, such as the classification of elements of the information system generated, e.g., Entities, Services and Use Cases. This choice was based on the maturity of the tool and the number of information systems that are in production and use [7]. In addition to automation, its use allows access to systems that can benefit directly from this proposal. Thus, we aim to count the number of function points from what was already realized, i.e., based on functionalities already developed. Providing an automated procedure for counting the realized effort aims at delivering transparency to the client receiving the product. This way, the effort effectively realized can be confronted with the initial estimates. Moreover, although the counting is applied on developed functionalities, the results produced can be used to analyze and understand the estimation errors. This knowledge must be used in order to improve the accuracy of the estimations of complexity for functionalities still under planning.

This paper is organized as follow. Section 2 presents some related works and our approach to the problem. Section 3 describes the concepts of Function Point Analysis. Section 4 explains in detail the proposed automatic counting. Section 5 discusses the prototype developed. Section 6 explores the Case Study. Finally, Section 7 concludes the paper.

 ISBN: 978-1-61208-458-9

II. RELATED WORK

Automation of FPA has been discussed for some time. The works [8] and [9] approach the process based on UML models, using Class Diagrams, Sequence Diagrams and Use Cases as inputs. In general, the models are read and interpreted by the application responsible for the counting, with some exceptions in which the user must interact with the application. This is because, despite the fact that the UML and the FPA are independent of technology, in general, the UML models do not have all the information necessary for the counting.

In order to aggregate automatic counting to models that are actually used in the development, i.e., that are synchronized with the current stage of information system, the work [10] described the process in an MDA framework. The main difference between the work in [10] and that is [8] and [9], is that it explores the fact that the system code and the counting are processed by the same tool, the code generation framework, not requiring any additional effort.

However, even the models used for code generation following the MDA approach may not be sufficient for the automation of the FPA. When working with a specific group of MDA frameworks that apply the partial generation of implementation code [11], the model does not fully represent the business rules of the information system. In this type of development, a portion of the system is implemented manually by the developer. Thus, the code will contain information relevant to the counting, e.g., entities that are changed by the system or not.

In our work, we chose a different approach to the problem. We use the MDA to extract information from models, ensuring its accuracy, and a tool to extract information from the system code. Since the information was only extracted from the code, if the technology is changed, it is necessary to change only the extractor.

By using not only models but part of the code, our work and [10] count the number of function points from what was already realized, i.e., based on information systems already developed. Although the counting is applied on a developed project, the result produced can be used to create a historical base and to adjust and improve accuracy of the FPA rules application. Further, the proposal will be described and exemplified with the Case Study.

III. FUNCTION POINT ANALYSIS

The FPA measures the functionality provided by a single information system [2]. It is a recognized ISO (International Organization for Standardization) standard for measuring software and can be determined from the requirements specification, considered as independent of technology. As proposed in [1], it counts the following system characteristics: files used by the system, external inputs and outputs, user interactions and interfaces. Each feature is considered individually and counted as the weights assigned.

The version proposed by IFPUG FPA, used in this work, provides some modifications to the original rules. It is described in seven steps [2].

1) Determine the type of function point counting.
2) Identify the system boundary.
3) Count the Data Functions.
4) Count the Transaction Functions.
5) Determine the value of unadjusted function points.
6) Determine the adjustment factor.
7) Calculate the adjusted value.

In our work, the type of count used (step 1) will be Development Project. It measures the functions provided to the user with the first installation of the system being delivered. Our work follows steps 2 to 5. Steps 6 and 7 do not fall within the scope of this work, as they use system specific features that must be manually adjusted.

The next two subsections describe the two function types related to the steps 3 and 4, respectively: data function and transaction function.

A. Data Function

The Data Functions are functions that deal with stored data. They are classified as *Internal Logical File* (ILF) or *External Interface File* (EIF). ILFs are related to data that are created or maintained by the system, while EIFs deal with external data.

B. Transaction Function

The Transaction Functions are functions that interact with some user or with external agents. They are classified as *External Input* (EI), *External Output* (EO) or *External Inquiry* (EQ).

a) EI: controls information or processes data. Its main objective is to keep one or more ILFs or to change the system behavior.

b) EO: sends data or controls information outside the system boundary. Its main objective is to provide information to the user, as in reports. They should contain some processing, for example, mathematical formulas, maintain an ILF or alter the system behavior.

c) EQ: sends data or controls information outside the system boundary. Its main objective is to retrieve information from data items. Unlike EO, they should not contain processing, maintaining an ILF or alter the system behavior.

Each Transaction Function also has a number of *Data Element Types* (DETs), the smallest meaningful data items presented (or requested) to (by) the user. Beside DETs, each Transaction Function has a number of *File Type References* (FTRs), the number of Data Functions accessed by the Transaction Function.

IV. PROPOSAL

Although FPA is independent of technology, when performed without the use of models, the automation of counting proposed becomes specific of technology. As an example, we have [12] where only the COBOL code is considered during analysis.

Figure 1 illustrates the scheme proposed [14], where the system code, represented by the points of implementation, and the artifacts with the characteristics of the system are generated from UML models. Particularly, Figure 1 shows an example of the MDA methodology using partial generation of implementation code. These points are the spots that actually contain the business rules of the generated systems.

 ISBN: 978-1-61208-458-9

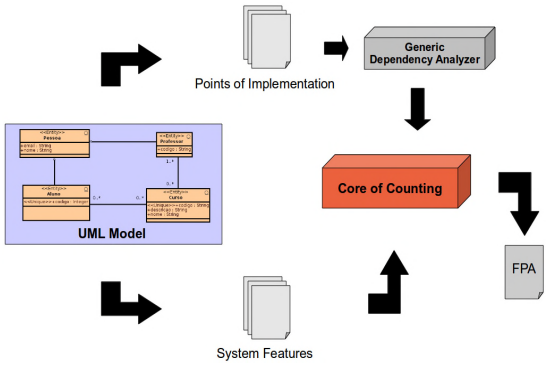


Figure 1. Schema of the proposal.

In our proposal [14], the system code is also used. However, only the dependence between the elements is considered, e.g., a Service that handles Entities. Thus, the points of implementation must pass through a dependency analysis before being used in the count. This allows the decoupling between the language used and the Core of Counting component.

Together with the dependencies, the system features are also used by the Core. Such features represent information that can be extracted from models, to classify elements as, e.g., Entities or Use Cases. Provided with this information, the Core can extract the dependencies that actually have value to count and identify the type of function: data or transaction. Considering the need to recognize and correctly count the types of functions, we developed conditions similar to those described in [8] and adapted for Web systems.

A. Counting Data Functions

As each Data Function has a number of DETs and RETs, they are counted as follows:

a) DET: defined as the number of attributes of the entity plus the number of inherited attributes, recursively, disregarding the identifying attributes.

b) RET: assumed to be 1 (one), since this value is used in most situations and has achieved good results [8].

TABLE I. DATA FUNCTIONS – UNADJUSTED VALUE [2]

RET	Data Element Type (DET)			Complexity	ILF	EIF
	1 ~ 19	*20 ~ 50*	*> 50*	***Low***	7	5
0 ~ 1	Low	Low	Average	***Average***	10	7
2 ~ 5	Low	Average	High	***High***	15	10
> 5	Average	High	High			

Through the combination of the number of DETs and RETs, it is possible to assign a complexity to the Data Function using the left side of Table 1. To each complexity is assigned a value of unadjusted function points, as shown on the right side of Table 1.

B. Counting Transaction Functions

As each Transaction Function has a number of DETs and FTRs. They are counted as follows:

a) DET: for an EI, it represents the number of arguments of the transaction. For an EO, it represents the number of output parameters. Finally, for an EQ, it represents the number of arguments of the transaction plus the number of output parameters.

b) FTR: analogous to the number of RET for Data Functions, it is assumed to be 1 (one) due the achievement of good results [8].

TABLE II. TRANSACTION FUNCTION – COMPLEXITY [2]

FTR	EI Data Element Type (DET)			EO and EQ Data Element Type (DET)		
	1 ~ 4	*5 ~ 15*	*> 15*	*1 ~ 5*	*6 ~ 19*	*> 19*
0 ~ 1	Low	Low	Average	Low	Low	Average
2 ~ 3	Low	Average	High	Low	Average	High
> 3	Average	High	High	Average	High	High

TABLE III. TRANSACTION FUNCTION – UNADJUSTED VALUE [2]

Complexity	EI	EO	EQ
Low	3	4	3
Average	4	5	4
High	6	7	6

Similar to the complexity assigned to Data Functions, the complexity of the Transaction Functions is realized based on the values of DET and FTR. Table 2 is used to determine the complexity of EIs, EOs and EQs. Then, the unadjusted value of function points can be obtained from Table 3, for all three types of Transaction Functions.

V. PROTOTYPE

In this section, we describe how the information necessary to perform the automatic counting of function points is obtained and processed. In order to achieve this goal, we developed a Prototype to demonstrate in practice how the automatic counting of function points is made. Figure 2 shows its operating model.

According to the model, the process begins with framework MDArte [5] to generate the artifacts used as input for the Prototype. The MDArte is a tool that receives UML models and generates the corresponding codes. It is one example of MDA framework with partial generation of implementation code, in which business rules should be described directly in the code at specific locations called points of implementation.

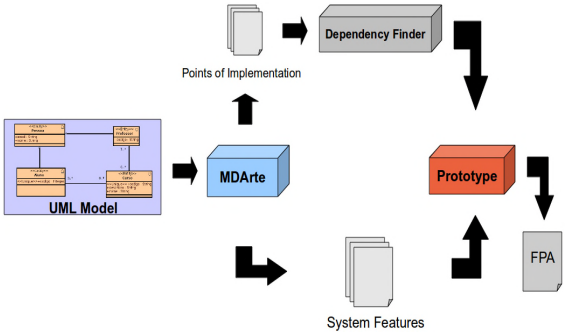


Figure 2. Schema of the prototype.

 ISBN: 978-1-61208-458-9

In our example, we use specific MDArte cartridges to generate information systems written in Java. Thus, we can add the tool Dependency Finder [13] to the proposed model. The Dependency Finder is capable of analyzing compiled Java code and extracting the dependency list of the elements. One of the benefits of its use is the generation of the list of dependencies in XML, as shown in Figure 3, following a pattern easily reproducible, which could be generated by other tools to analyze other programming languages. Thus, the Prototype does not depend directly on a technology, being flexible to deal with other cartridges generation, or even systems coded manually.

```
<?xml version="1.0" encoding="utf-8" ?>

<dependencies>
 <package confirmed="yes">
  <name>br.ufrj.coppe.system</name>
  <class confirmed="yes">
   <name>br.ufrj.coppe.system.Student</name>
   <outbound type="class" confirmed="yes">br.ufrj.coppe.system.Person</outbound>
   <feature confirmed="yes">
    <name>br.ufrj.coppe.system.Student.Student()</name>
    <outbound type="feature" confirmed="yes">br.ufrj.coppe.system.Person.Person()</outbound>
   </feature>
   <feature confirmed="no">
    <name>br.ufrj.coppe.system.Student.getName()</name>
     <outbound type="class" confirmed="no">java.lang.String</outbound>
   </feature>
  </class>
 </package>
</dependencies>
```

Figure 3. Example of XML produced by the Dependency Finder.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<statistics>
 <application>AcademicSystem</application>
 <type>entities</type>
 <entities>
  <entity>
   <name>br.ufrj.coppe.system.Person</name>
   <attributes>
    <size>1</size>
    <attribute identifier="false">
     <name>name</name>
     <type>java.lang.String</type>
    </attribute>
   </attributes>
   <methods>
    <size>1</size>
    <method modifier="false">
     <name>getName</name>
     <return>
      <type>java.lang.String</type>
     </return>
     <parameters>
      <size>0</size>
     </parameters>
    </method>
   </methods>
  </entity>
  <entity>
   <name>br.ufrj.coppe.system.Student</name>
   <extends>br.ufrj.coppe.system.Person</extends>
```

Figure 4. Example of XML generated by MDArte with system features.

Although the list of dependencies has an important role in the process, it is not enough for the FPA to be performed. As seen in Figure 3, the XML produced does not allow for the classification of elements. So, we use the characteristics of the information system as auxiliary entry, illustrated by the XML from Figure 4. This XML excerpt describes some characteristics of Entity elements.

From Figure 4, we can see that "Person" is an entity and has a "name" attribute. You may also notice that the entity "Student" inherits information from "Person", which explains the presence of method “getName” in Figure 3, unconfirmed, since it belongs to "Person" and not to "Student". The information from both types of entries is related and processed by the Prototype and used to identify and count the two types of functions: data and transaction.

The Prototype identifies the Data Functions checking which Entities are inside (ILF) or outside (EIF) of the boundary of the information system. To do so, it verifies which Entities have their set methods accessed. This is done through the attribute "modifier" tag "method", present in the XML with the characteristics of the system as shown in Figure 4. This attribute indicates methods that can be used to change an Entity, which allows its search on the list of dependencies.

After having classified the Data Function as ILF or EIF, the process of counting the number of DETs and the number of RETs begins. Both values are calculated as described in the previous section. The number of DETs is defined by the number of attributes of an entity, considering the inherited attributes and disregarding the identifying attributes. The number of RETs has been assumed to be 1 (one), as stated in the proposal.

The Transaction Functions are identified and counted according to the proposed rules. However, the concepts have been adapted to Use Cases, particularly the Activity Diagrams that describe their flow. Figure 5 illustrates how the Activity Diagrams used by the MDArte are modeled. The activities with the stereotype *<<FrontEndView>>* represent screens and the values associated to the outgoing transitions its parameters.

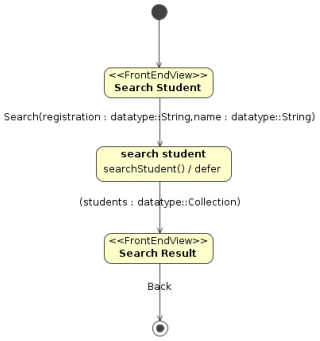


Figure 5. Example of Activity Diagram read by the MDArte.

The identification process is represented by the flowchart in Figure 6. As described in the flow, the functions are classified into EI or EQ. This flowchart represents the first stage of the process, remaining to deal with the EOs.

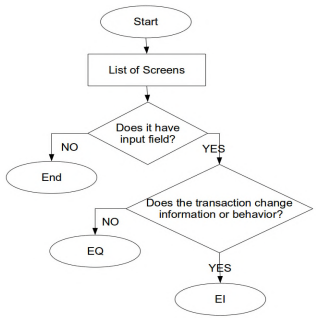


Figure 6. Flowchart of Transaction Functions identification process. First Stage.

 ISBN: 978-1-61208-458-9

After having identified the EIs and the EQs, the process continues with the second stage described by the flowchart in Figure 7. Now, the screens not yet identified are checked looking for a complementary screen of an EQ, e.g., the screen that shows the results of a query. If it is not the case, then the screen is classified as EO.

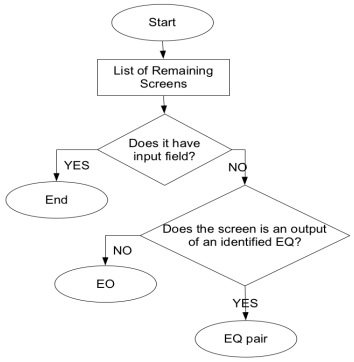


Figure 7. Flowchart of Transaction Functions identification process. Second Stage.

Next, we describe how the number of DETs for each type of Transaction Function is counted based on information provided to the Prototype. Remembering that the number of FTRs has been assumed to be 1 (one), as stated in the proposal.

a) External Input (EI): its number of DETs is calculated by adding the number of input attributes, present on the screen, including buttons.

b) External Output (EO): its number of DETs is calculated by adding the number of attributes in the results screen. Emphasizing that when displaying information as table, only the columns are counted, not lines.

c) External Inquiry (EQ): its number of DETs is calculated by adding the number of the input attributes, as for EIs, and the number of output attributes, as for EOs.

Having counted the number of DETs and RETs for Data Functions, and the number of DETs and FTRs for Transaction Functions, the Prototype performs the assignment of complexity for each function. Afterwards, it also determines the value of unadjusted function points based on Tables 1 to 3.

VI. CASE STUDY

The Case Study was prepared following the proposal described in this paper as well as the rules used by the Prototype. Its goal is to demonstrate how automatic FPA is made for real examples.

Therefore, we chose an information system of an academic environment as example, limited to a few Entities and Use Cases for better understanding. Thus, our example is only responsible for keeping the information of *Students* and allows the *User* to change some of its information, like the *password*.

First, we analyze the Data Functions and then the Transaction Functions.

A. Data Functions

Figure 8 illustrates the class diagram of the Case Study. You may notice the five Entities, separated into two symbolic groups: *academic system* and *access control*.

a) *Person*: person information.

b) *Student*: student information.

c) *User*: system user information.

d) *Group*: user groups information.

e) *Action*: information of actions that can be done through the system related to the group permission.

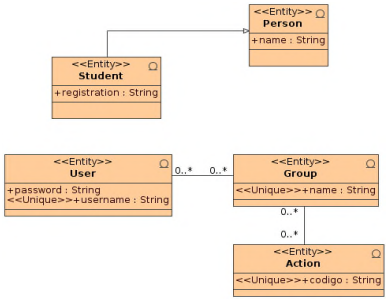


Figure 8. Example of Class Diagram read by the MDArte.

The entities *Group* and *Action* are kept outside the system boundary, as they represent a part of access control based on access groups, like profiles. Thus, as described in the previous section, the identification of the Data Functions is done by searching for methods that can modify each entity among the dependencies of system operations.

a) ILF: *Person*, *Student* and *User*.

b) EIF: *Action* and *Group*.

TABLE IV. COUNTING DATA FUNCTIONS

	DET	RET	Complexity	Value
Action	1	1	Low	5
Group	1	1	Low	5
Person	1	1	Low	7
Student	2	1	Low	7
User	2	1	Low	7
Unadjusted Total				31

After identifying each entity, the counting process of RETs and DETs starts. The number of DETs is defined as the number of attributes of the entity plus the number of inherited attributes, recursively, disregarding the identifying attributes. The number of RETs has been assumed to be 1 (one), as stated in the proposal. Applying the values in Table 1, the unadjusted values will be as defined in Table 4.

B. Transaction Functions

In our example, we will use the CRUD of the Entity *Student* to validate the proposal. The left side of Figure 9 represents a screen that allows the *User* to create a new *Student* in the system. It is an example of EI, with two parameters and one button. Its counting is based on Tables 2 and 3, and shown in Table 5.

 ISBN: 978-1-61208-458-9

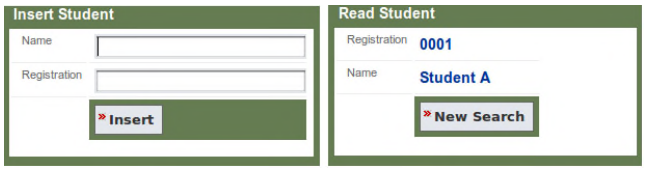


Figure 9. Screens for inserting and reading a student.

As an example of EO, we have the screen displaying the information of a Student, illustrated on the right side of Figure 9, with two result attributes. Applying the values in Tables 2 and 3, we have the counting as shown in Table 5.

The identification and counting of an EQ can be considered the most complicated of the three. We chose to use the same use case described by the Activity Diagram in Figure 5. The activity named *Student Search* is the screen shown on the left side of Figure 10, and the activity *Search Result* represents the screen on the right side. In *Student Search*, the input values are displayed, two parameters and one button. *Search Result* has four attributes, represented by two columns and two buttons, where *View* is counted only once. Based on the same tables as for EO (Table 2 and 3), we have the counting shown in Table 5.

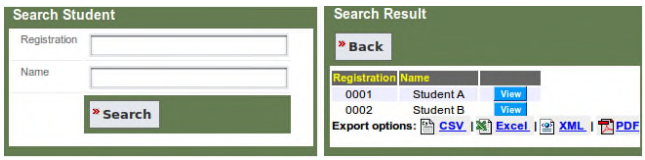


Figure 10. Screens for searching student.

TABLE V. COUNTING TRANSACTION FUNCTIONS

	DET	FTR	Complexity	Value
Insert Student	3	1	Low	3
Read Student	2	1	Low	4
Search Student	7	1	Low	3
Unadjusted Total				10

Finally, from the totals in Tables 4 and 5, *31* and *10*, respectively, we get the unadjusted total of *41* function points. The total obtained represents the complexity of the information system described by the Case Study, according to the rules established by IFPUG [2] and the proposed automation of this work.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a proposal for automatic Function Points Analysis (FPA) following the standards presented by IFPUG. Our proposal counts the number of realized function points, based on functionalities already developed. This work is the first step aiming to answer the following question: is the complexity of what was implemented close to the estimated complexity? The answer to this question provides transparency to the client receiving the product and allows for the creation of a historical base that can be used to improve accuracy of functionalities under planning.

The proposal was evaluated through the construction of a prototype and a case study. The counting performed on the case study was accurate. We are aware that some simplification, such as assuming RET and FTR values always as 1, will generate deviations on more complex or realistic scenarios. That is exactly why our future steps are: (1) evaluate the proposal on real applications developed using the MDArte; (2) perform the counting on such applications with the assistance of a FPA specialist; (3) identify the deviations; and (4) evolve the proposal in order to gather more required information from both models and code. The belief is that the more information we can assume about the patterns and architecture of the information system developed the more accurate the automatic counting procedure will be. This led us to another important issue that will be evaluated in future research: which is the minimum set of assumptions about the system implementation in order to achieve a result with reasonable precision?

REFERENCES

[1] A. J. Albrecht, “Measuring application development productivity,” Proceedings of the Application Development Symposium, New York, USA, 1979, pp. 83-92.

[2] IFPUG, “Function point counting practices manual,” release 4.1, International Function Points Users Group, NJ, 2000.

[3] G. Booch, J. Rumbaugh, and J. Jacobson, “The unified modeling language user guide,” Addison-Wesley, MA, 1999.

[4] J. Siegel, and the OMG Staff Strategy Group, “Developing in OMG’s model driven architecture”, OMG white paper, 2001.

[5] MDArte, “Framework MDArte,” https://softwarepublico.gov.br/social/mdarte/, accessed on 23/12/2015.

[6] AndroMDA, “Framework AndroMDA,” http://www.andromda.org, accessed on 23/12/2015.

[7] R. E. A. Pinel, F. B. do Carmo, R. S. Monteiro, and G. Zimbrão, “Improving tests infrastructure through a model-based approach,” ACM SIGSOFT Software Engineering Notes. 36(1), 2011, pp. 1-5, doi: http://dx.doi.org/10.1145/1921532.1921544.

[8] T. Uemura, S. Kusumoto, and K. Inoue, “Function-point analysis using design specifications based on the unified modelling language,” Journal of Software Maintenance: Research and Practice, vol. 13(4), 2001, pp. 223-243.

[9] T. Iorio, “IFPUG Function Point analysis in a UML framework,” Proceedings of Software Measurement European Forum, 2004.

[10] P. Fraternali, M. Tisi, and A. Bongio, “Automating Function Point Analysis with Model Driven Development,” Proceedings of CASCON, 2006, doi: http://dx.doi.org/10.1145/1188966.1188990.

[11] R. Soley, and the OMG Staff Strategy Group 2000, “Model driven architecture,” OMG white paper, 2000.

[12] V. T. Ho, and A. Abran, “A Framework for Automatic Function Point Counting From Source Code,” International Workshop on Software Measurement (IWSM), 1999.

[13] Dependency Finder, http://depfind.sourceforge.net, accessed on 23/12/2015

[14] R. E. A. Pinel, “Análise de pontos de função em sistemas desenvolvidos usando MDA,” COPPE, Master thesis, 2012

 ISBN: 978-1-61208-458-9

Developing Software for Mobile Devices: How to Do That Best

Invited Panel

Hermann Kaindl[1], Roberto Meli[2], Andreas Kurtz[3], Bernhard Bauer[4], Petre Dini[5]
[1)] TU Wien, Institute of Computer Technology, Vienna, Austria
[2)]DPO Srl, Italy
[3)]BMW AG, Integration Electric/Electronics, Software, Munich, Germany
[4)]University of Augsburg, Institute for Computer Science, Augsburg, Germany
[5)] Concordia University, Canada | China Space Agency Center, China

Emails: {hermann.kaindl@tuwien.ac.at, roberto.meli@dpo.it, Andreas.Kurtz@bmw.de, bauer@informatik.uni-augsburg.de, petre@iaria.org}

***Abstract*—Including computers and applications into mobile devices creates a major break-through in the applicability of computing systems and in the impact this had on users and even the society. While software development has always been costly and challenging, it is even more challenging for mobile devices. This raises the important question of how to best develop software for mobile devices.**

Keywords-mobile device; software development; user interface; Apps; testing.

I. INTRODUCTION

Mobile devices are comparably new and have differences to more traditional computers like mainframes and PCs (Personal Computers), such as the following:

- Different and possibly adaptive mobile user interfaces
- Context-aware/context-sensitive mobile applications
- Ubiquitous interactions, e.g., with wearables

Because of these differences, especially the software development for mobile devices poses challenges beyond that of traditional software development. This raises the important question of how to do that best.

The remainder of this paper is organized in the following manner. First, automated tailoring of user interfaces for smartphones and tablet computers is sketched and discussed in the context of mobile devices. Then, Apps development for mobile devices is contrasted with software measurement. After that, test automation is presented for cars viewed as mobile devices. Finally, challenges on designing and testing both Apps and wearable devices are presented.

II. TAILORED USER INTERFACES FOR SMARTPHONES AND TABLET COMPUTERS (HERMANN KAINDL)

A fairly obvious difference between, e.g., PCs and mobile devices such as smartphones and tablet computers is given through their relative screen sizes. Simply looking up a Web page prepared for a screen of a typical PC from a smartphone reveals problems like a tunnel view, which impair the usability. Sites looked up very often like those of CNN or airlines; therefore, they present their content *tailored* for large or small screens, respectively. This means extra effort for preparing these Web pages twice. In fact, there is a whole spectrum of screens sizes due to the large variability of screens of tablet computers and smartphones. When tailoring for a larger number of screen sizes, even more effort is required.

This issue calls for support through automation. In fact, technology exists for automated generation of Graphical User Interfaces (GUIs) [7][12][13]. In particular, also automated tailoring through optimization techniques is available [15]. Sample GUIs created (semi-)automatically can be viewed online:

- A demo flight booking GUI, see [1]
- An accommodation booking GUI, see [2], reverse-engineered from a real-world site (which is not online any more)

Of course, GUIs cannot be generated through magic. This approach requires high-level Discourse-based Communication Models [7][13] as well as (simple) device specifications to be created manually. While the effort for creating such models may not always pay back for generating GUIs of a single device, it most likely will for generating GUIs for multiple devices from a single model.

Unfortunately, the usability of fully-automatically generated GUIs is insufficient at the current state of the art.

 ISBN: 978-1-61208-458-9

So, we devised the so-called Custom Rules for addressing usability problems in a persistent way, which even showed that such rules can, in principle, be reused for multiple devices [16].

Still, there are obstacles for a wide-spread applicability of such an approach. Recently, we removed the problem of persistently including Custom Widgets through a GUI designer. The flight booking application [1] includes a seat picker widget as usual in real-world applications but unavailable in usual widget libraries.

This approach for automated tailoring even allows choosing different strategies such as tabbing or vertical scrolling, when the content does not fit the given screen size [15]. We found some evidence that the more wide-spread vertical scrolling is more efficient for use [14].

With respect to different screen sizes, we found some evidence that a user is typically more efficient on screens of larger sizes [17]. Of course, there is a trade-off with the mobility of such devices.

III. APP DEVELOPMENT & MEASUREMENT: ALLIES OR ENEMIES? (ROBERTO MELI)

Mobile application engineering is a relatively new branch of software engineering. Mobile application development and maintenance are characterized by:

- Small project sizes and short schedules
- Volatile scope
- Use of diverse technologies,
- User interface and user experience relevance
- Multimedia integration
- Geographical information integration
- Social remote and local interaction

These elements require an organizational approach based on:

- Time responsiveness
- Agile or evolutionary processes
- Small and very integrated teams
- Strong user involvement
- Interdisciplinary skills
- Supportive architectures and tools

Due to the deadline and uncertainty resolution focus and production orientation, teams are usually not too interested in "traditional" engineering practices, especially in measurement activities. They are perceived as "overhead". If any measurement is taken in the App project it is often a technological measurement.

A. Useful or not?

Nevertheless, "Functional" and "Non-Functional" Size Measurement Methods might be very useful in circumstances like the following:

- Corporate context
- Tender / Contract Management
- Project oriented development
- Prioritized and variable resource allocation
- Internal User driven
- Project productivity assessment needs
- Cost control emphasis

On the other side, measurement is not particularly significant in these situations:

- Personal context
- Informal internal contracts
- Service oriented development
- Self-managed team management
- Fixed resource allocation
- Market User driven
- Business Unit productivity assessment needs
- Time to market emphasis

When we consider Apps development effort, duration and staff estimation, apparently, there is no spread adoption of formal methods. Expert judgment seems to be the most adopted strategy. Unfortunately, the quality of these estimates is dependent on the quality of the estimators and many times it is impossible to compare different situations and to share expertise among different teams [1].

B. Typycal Processes & Deliverables

The process to develop an App is not so different from those applied to multimedia product or web-based applications [19]. A typical process should be:

- Agile-oriented
- Iteration-oriented
- Supported by tools

and should include phases like the following:

- Feasibility Study
- Collection of Functional and Non-Functional requirements
- App Wireframe creation
- Target architecture definition (Android, IOS, etc.)
- Back end
  - Defining the back end structure
  - Management of users
  - Server side logic
  - Customization of User Experience
  - Data integration (remote/local)
  - Push notification services
- Front end
  - Caching of data
  - Synchronization of App data
  - Mock ups Wire framing
  - UI design and development
  - UI improvements
  - Testing
  - Deployment

Developing an App is "project-oriented" but maintaining it may be "service oriented" with a continuous improvement process in place.

"The biggest issue, in mobile application development, still seems to be the diversity of platforms and devices.

 ISBN: 978-1-61208-458-9

Offering an App, be it enterprise specific or publicly available, means to provide different versions at least for the most widespread platforms (e.g., Android, Apple iOS, BlackBerry OS), operating system versions (with each version providing new functions or even altered appearance) and device types (with different display sizes and resolutions, controls and navigation styles). Since no standard cross-platform development approach has emerged so far, this plethora of combinations results in considerable development effort." [1]

Deliverables are documents and products in the multimedia domain and developing an App is not only a matter of Programmers and ICT people.

C. Which Measurement and Models?

Any adopted measurement model should be:

- Light
- Quick
- Simple
- Used by developers
- Complete
- Standard
- Product-oriented
- Easy to learn

Simple Function Point [21] has these characteristics for functional sizing. Measurement should be used for the governance of the process and the relationship among the different stakeholders.

In order to estimate effort, duration and staff, a complete model should be used which takes in account not only the functional requirements, but also the non-functional and process requirements like the one presented in [10].

IV. Software Based Test Automation Approach Using Integrated Signal Simulation (Andreas Kurtz and Bernhard Bauer)

New operating concepts are pushing from the Consumer Electronics Sector (CES) in the automotive industry. This change characterizes the development in the automotive industry and makes vehicle manufacturers increasingly become software developers. Software is an enabler for flexible and fast growing innovations. Especially the development cycle in the CES challenges the automotive industry not to lose connection. Vehicles nowadays must be linked with the customers' mobile devices and so become a mobile device. In today's vehicles, classic switches have almost become obsolete. "The automobile is the ultimate mobile device." [22]. Modern vehicles can be considered as mobile devices with Human-Machine Interfaces (HMI) such as displays, touch screens, gesture control and sensor operation. With increasing networking and alternative control options of functions, this change confronts the testing of customer functions of a vehicle with enormous challenges.

A. Challenges

Developing suitable software testing methods is the main challenge in software development for mobile devices to get high quality software. The speed of the hardware development, and software development cycles of the consumer electronic industry infuses the automotive industry. Because of changing trends, the growing networking of systems needs an innovative approach to be able to test the developed software fully automated. Innovative automation methods are a key part to handle the time pressure. In order to meet this challenge needs, a software-based approach with possibility to test the entire chain of reaction. A software-based approach allows reacting flexibly and fast on changes of software, especially changes on the interfaces.

Particularly, in the field of HMI, the technology is changing increasingly towards sensors without mechanical haptics. From the perspective of the user, the sensors and actuators on the HMI are fused to one single interface, touchscreens or sensor areas. This helps the designers to reduce costs because of being able to change the visual surface via software.

Further steps for interacting with the mobile device will be contactless input, gestures or the so-called air touch technology [9]. Following the term 'mobile devices' includes vehicles or subsystems of a vehicle.

The changing types of sensors with the innovation speed lead to new automation methods, to a software-based integrated approach being able to be adapted as fast as the software and hardware changes. Software-based integrated testing methods are missing due to consistent approaches, and lack of standardization. Especially in the automotive industry, software does not have a common architecture. This causes special/customizable solutions for each implementation.

B. Status

As mentioned before, an automobile becomes a mobile device. Depending on the point of view, the vehicle system is a mobile device second order. This means it is a distributed system combining severally mobile devices to a bigger mobile device. This consideration is possible because of the comparable basic architectures of networked Systems-On-a-Chip (SOC) or on the automotive domain networked software components on Electronic Control Units (ECUs) being SOCs. To show current solutions for automated testing these are separated in external- and software-internal solutions. With focus to model based testing methods [18], the testing is separated in four testing steps, in hierarchical order, and refers at each solution.

- Component test
- Integration test
- System test
- Acceptance test

1) External Automation Solution

Test automation with external automation solution makes only sense at component test, integration test or part system test (part system is a system cut in domain systems e.g., power-train system). However, the effort to adapt the interfaces increases enormous at part system test. Automation solutions for testing customer functions are

 ISBN: 978-1-61208-458-9

stimulating the component with physical hardware signals or remaining bus simulation.

2) Software-internal Automation Solution

A different solution, usable at any test layer, are additional software functions for an interaction in the software to trigger customer functions. This allows switching values or triggering customer functions, but needs for each customer function a custom-developed and integrated additional function. If the customer function is changed or moved to another hardware the additional function for software-based interaction has to be changed, too. Duplication of effort, in conjunction with increasing probability of errors may result.

C. The Methodological Approach

Figure 1 shows the methodical approach. Projecting this approach, based on an AUTOSAR [6] architecture, to other software architectures is possible. Various intermediate steps create a system model and test model for the requirements. Integrating an additional Software Component (SWC), called SIMulation Agent (SIM Agent) and deploying it to all ECUs, generate a software-based distributed simulation, with the help of an extended driver module to get access to the new simulation module. This allows simulating signal sequences in the driver layer with the advantage of reduced data types and a standardised interface. All other steps of the methodology are automated. From the test model, abstract test cases with abstract interfaces are created. These abstract interfaces become specific with the help of the deployment files. This allows performing the same test cases on various hardware platforms by adjusting the mapping 'Config', e.g., testing the same software on different mobile phones.

D. Alternative Proposals

An alternative approach could be a different interaction layer for this kind of simulation approach to avoid changes in the AUTOSAR architecture used in automotive domain. This is more compliant to the actual AUTOSAR standard but increases the number of data types.

V. CHALLENGES ON DESIGNING AND ON TESTING FOR WEARABLE DEVICES AND APPS (PETRE DINI)

Three complementary activities are specifically identified in new market communications activities, namely building wearable devices, designing Apps dedicated to them, and testing the solutions. The challenges are driven by several specific features characterizing each of them, but also by the nature of services they are used for and the human behavior. As some of the services are related to life threatening, testing the systems becomes a cornerstone process. The diversity of the devices, the heterogeneity of platforms, the absence of specific APIs and the scattered nature of system parts add to the complexity for verification and validation activities.

There is a continuously growing market boosted by Apple Watch very recently. Analysts predict a 42% growth for the wearable market within the next 5 years, while the Apps market should follow [3].

A. Challenges in Apps Development

The challenges faced by Apps developers are essentially induced by the wearable devices.

1) Devices and Apps

Some of the devices have always the screen on (like Peeble) that should be considered when designing an App to save as much energy as possible. Multiple screen sizes and formats (round, squared, e-paper display) need a fully adapted User Interface (UI) design. Computation options should also be limited to the minimum needed, as developers face limited computed power on a wearable device.

Wearable software is fragmented is more visible than for handheld devices is its intended purpose. Because of lack of established API, all coding of features takes place individually. So far, no accepted development cross-platforms exist; there are several operating systems, but no industry standard. There are ongoing industrial activities: Google is developing their Android wearable software development kit, NTT Docomo's Device Connect WebAP, GitHub is sharing the API as open software to enhance both technical specifications and API for mass commercialization.

There is a tendency to simply re-implement everything in the existing App on the wearable from an existing mobile App. This is not a recommended approach, as the interaction with the wearable watch is different that the interaction with a phone device. As a result, appropriate methodologies and guidelines should be developed and adopted. The current development platforms have limited features for an appropriate animation.

Troubleshooting wearable devices and Apps together leads to time-intensive development process and this is due to the frequency of troubleshooting on the new platforms.

There is a market push for reaching harmonization for Apps development. Juniper Research estimates the health related wearable devices industry will reach $53 billion in four years [4]. As a result, there is a potential that standardization and methodologies see a quick development. The finance sector is also helping, e.g., the introduction of Apple Pay along with the Apple Watch are current solutions; even more, payment-capable bracelets are offered by CaixaBank and Barclays.

The growing segment in the Apps marketplace will need a support for security and privacy. Practically, an embedded approach of wearable devices and Apps is a vital solution.

2) Thermal Considerations

A specific aspect is that wearable devices introduce some unique thermal design challenges that should be considered for devices, Apps and the entire system. This is not only referring to operability, but also to a required comfort level for humans. This design challenge is mainly for processor-intensive applications and units with complex displays.

According to Heussner [8] "electronics placed in direct contact with the skin need to maintain an ideal operating temperature at or below the core body temperature of 37°C (98.6°F). Anything above this is generally considered to be uncomfortable and hot (see Figure 2). Transitioning to much higher heat (above 40°C or 104°F) will trigger discomfort and pain for the wearer."

 ISBN: 978-1-61208-458-9

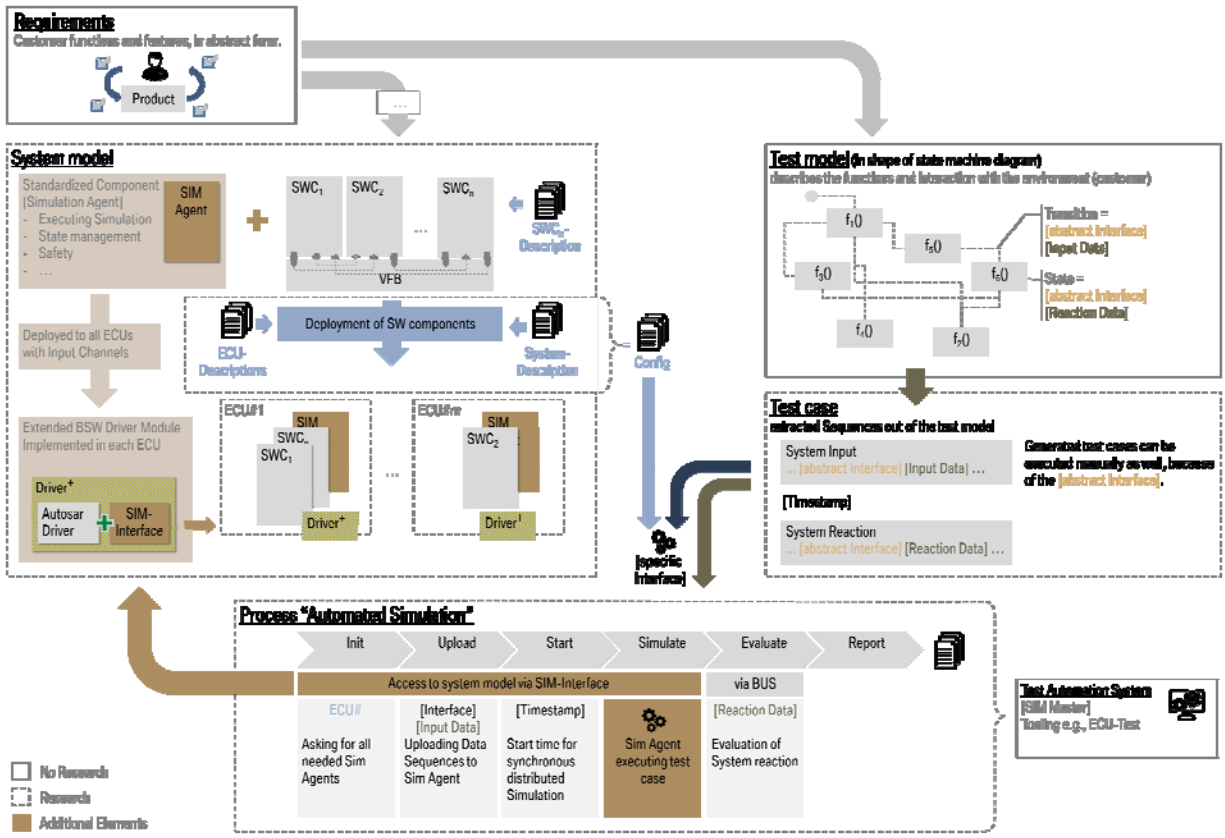


Figure 1. Software-based methodology for test automation in distributed systems.

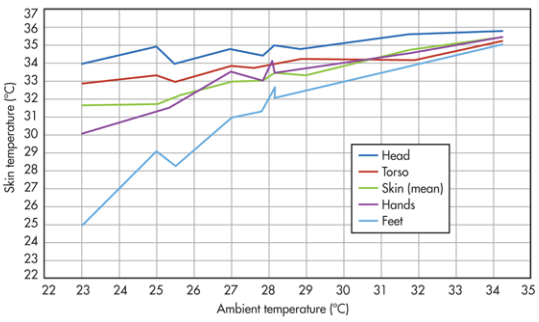


Figure 2. Thermal Considerations [8].

3) Materials and Environment

The design should address issues related to material interaction, reliability of interfaces, and impact on the thermal environment (for devices, Apps, systems). Chemical and a mechanical material interaction have to be calibrated; testing to optimize the package, the coating, and encapsulation is needed.

B. Testing Challenges

1) Testing Wearable

Wearable devices are deployed everywhere, with various functions, such as sensing, computing, transmitting, alerting, etc. A few characteristics make testing challenging, as listed below.

2) Small Screen

The designers must redefine the wearable screens and adapt their designing skills to miniaturization; dimensions should be carefully decided, as every pixel matters. There are certain limits at which a screen can be squeezed, yet being conveniently useful. Little of known UI/UX methodologies can be reused in designing new APIs.

3) Functional Testing

A big testing paradigm change was identified when the mobile devices arrived. Wearable devices comprise also different sensors and specific interactions that cannot be functionally tested by using traditional methods

4) Interaction

Testing should consider a myriad of sensor interactions. The large spectrum of interactions (Bluetooth, WI-Fi, hardware) leads to large coverage needs.

 ISBN: 978-1-61208-458-9

5) Battery Life

Energy and battery-based operation raises special maintenance issues and a real challenge for both wearable App developers and testers. These need also suitable testing criteria tuned to the new features of devices and Apps.

6) Testing for-Real

As wearable devices are quite specific, simply substituting them with emulators is not suitable; as the disciple is evolving in a rapid pace, trusting the results of such emulator is doubtful. Still, there are a few wearable on the market, e.g., Tizen, Android, etc.

7) Materials-oriented Testing

Due to metal migration concerns, biased testing is increasingly important to validate sensitivity in moist environments and to validate the risk of tin whiskers [8].

8) Testing body-wearable systems

There is a large variety of wearable devices and Apps, from fitness bands (which are essentially data collectors) to portable heads-up display; additionally, complex interactions occur between the touch display, cameras, and fast data communication with mobile platforms (see Figure 3).

Figure 3. Body wearable networks.

Complementary and specific components, like smart e-textiles, integrate stretch, pressure, and contact-based sensor elements, integrated within the fabric itself. Testing these components and their interactions requires appropriate experiments and calibration; this includes thermal aspects and materials characteristics on top of standard development guidelines of mobile devices and/or classic software development process.

VI. CONCLUSION

Of course, we cannot ultimately clarify how to best develop software for mobile devices. Still, we present a few related viewpoints that should help to pave the way towards a better understanding.

For instance, it is clear that different mobile devices need different user interfaces. With regard to screen size, automated GUI generation with automated tailoring may become an option.

Even whole cars may be viewed from the perspective of mobile devices today, since the automotive industry is increasingly influenced by the consumer electronics industry. This requires software-based integrated testing methods in order to keep up with the development.

What is specific on designing and testing wearable devices and Apps is that user experience is more relevant than in traditional approaches. It is a challenge to develop and test very specific features; e.g., “smart watches have very small screens and almost no buttons, making the use of space, navigation and user interaction incredibly important” [5].

Overall, it seems as though there will not be any single approach for developing software for mobile devices “best”.

REFERENCES

[1] http://ucp.ict.tuwien.ac.at/UI/FlightBooking

[2] http://ucp.ict.tuwien.ac.at/UI/accomodationBooking

[3] https://www.utest.com/articles/challanges-of-testing-wearable-devices

[4] http://www.foxnews.com/tech/2015/02/23/top-wearables-for-medical-issues.html

[5] http://www.belatrixsf.com/index.php/whitepaper-the-next-frontier-of-technology-wearables

[6] AUTOSAR Partnership, *AUTOSAR Layered Software Architecture*, 2014. [Online]. Available: http://www.autosar.org/.

[7] Falb, J., Kaindl, H., Horacek, H., Bogdan, C., Popp, R., and Arnautovic, E., A discourse model for interaction design based on theories of human communication. In *CHI ’06 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2006. ACM Press, pp. 754–759.

[8] Heussner, D. Texas Instruments, USA, http://electronicdesign.com/digital-ics/wearable-technologies-present-packaging-challenges

[9] Horn N., *BMW Group at the CES 2016 in Las Vegas.*. BMW presents the principle of the contactless touchscreen with AirTouch.

[10] Meli, R. A New Unified Model of Custom Software Costs Determination in Contracts, *Softeng2015*, Barcellona, 2015.

[11] André Nitze, Andreas Schmietendorf, Reiner Dumke, An Analogy-Based Effort Estimation Approach for Mobile Application Development Projects, IWSM-MENSURA, 2014, pp. 99-103, doi:10.1109/IWSM.Mensura.2014.9

[12] Paterno, F., Santoro, C., and Spano, L. D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.* 16 (November 2009), 19:1–19:30.

[13] Popp, R., Raneburger, D., and Kaindl, H., Tool support for automated multi-device GUI generation from discourse-based communication models, in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive computing systems* (*EICS’13)*. New York, NY, USA: ACM, 2013. Tool demo paper.

[14] Raneburger, D., Alonso-Rios, D., Popp, R., Kaindl, H., and Falb, J., A User Study with GUIs Tailored for Smartphones, in *Proceedings of the 14th IFIP TC 13 International Conference on Human-Computer Interaction - INTERACT 2013*, Part II, Springer LNCS 8118, Springer LNCS 8118, 2013, pp. 505–512.

[15] Raneburger, D., Kaindl, H., and Popp, R. Strategies for automated GUI tailoring for multiple device. In *Proceedings of the 48th Annual Hawaii International Conference on*

 ISBN: 978-1-61208-458-9

System Sciences (HICSS-48), IEEE Computer Society Press (Piscataway, NJ, USA, 2015), 507–516.

[16] Raneburger, D., Kaindl, H., and Popp, R. Model transformation rules for customization of multi-device graphical user interfaces. In *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, ACM (New York, NY, USA, 2015), 100–109.

[17] Raneburger, D., Popp, R., Alonso-Rios, D., Kaindl, H., and Falb, J., A User Study with GUIs Tailored for Smartphones and Tablet PCs, in *Proceedings of the 2013 IEEE International Conference on Systems, Man and Cybernetics (SMC'13)*, 2013, pp. 3727 - 3732.

[18] Roßner, T, C. Brandes, H. Götz and M. Winter. *Basiswissen modellbasierter Test*. dpunkt.verl., Heidelberg, 1 edition, 2010.

[19] Ruhe, M., Jeffery, R., Wieczorek, I., Cost estimation for Web applications, in *Proceedings of International Conference on Software Engineering (ICSE'03)*, 2003, 285–294.

[20] Seidl R., Baumgartner M., and Bucsics T. *Praxiswissen Testautomatisierung*. dpunkt, Heidelberg and Neckar, 1 edition, 2011.

[21] SiFPA, *Simple Function Point Functional Size Measurement Method*, Reference Manual SiFP-01.00-RM-EN-01.01, http://www.sifpa.org/en/index.htm, [retrieved: January, 2016].

[22] Diess, H., CES-Keynote, 2016

 ISBN: 978-1-61208-458-9

Developing a Quality Report for Software Maintainability Assessment: An Exploratory Survey

Pascal Giessler
and Michael Gebhart

iteratec GmbH
Stuttgart, Germany
Email: pascal.giessler@iteratec.de,
Email: michael.gebhart@iteratec.de

Manuel Gerster, Roland Steinegger
and Sebastian Abeck

Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: manuel.gerster@student.kit.edu,
Email: roland.steinegger@kit.edu,
Email: sebastian.abeck@kit.edu

***Abstract*—Maintainability can be a key factor concerning the success of a software product, since the majority of software life cycle costs is spent on maintenance. Therefore, there is a deep interest in analyzing and assessing the maintainability of a software product with the objective of identifying the need for action and subsequently minimizing maintenance expenses. Often software quality metrics are used to analyze the influencing factors of maintainability while an expert uses their results for the assessment. However, these metrics are distributed across several tools, dashboards, and literature. Moreover, there are further quality indicators for analyzing the maintainability that cannot be evaluated automatically by means of metrics. Hence, we aim to develop a quality report containing well-known quality metrics and further quality indicators that allows an expert to assess the system under review regarding its maintainability. For this reason, we conducted an exploratory survey in the area of research and industry to get the essential ingredients of such a quality report. In this paper, we present the survey and its outcomes. The survey shows potential ingredients of a quality report, i.e., metrics and quality indicators, which can be measured not only automatically but also manually.**

Keywords–maintainability assessment; software quality; quality report; quality analysis; quality indicators.

I. INTRODUCTION

Nowadays, software products can be seen as ubiquitous components in our daily life. Each software product is designed to satisfy one or more business or user needs. But, these needs may change over time due to several influencing factors, such as changing market conditions or customer behavior. As a result, software modifications are required to support these new needs [1]. From an economic point of view, these modifications should be performed fast with low costs, due to the fact that the majority of software life cycle costs (LCC) is spent on maintenance and not on development [1] [2] [3].

That is why it is important to develop and design software products with maintainability in mind. But, it is unclear how to analyze and assess the maintainability of a software product in order to derive actions for improvement. There is no uniform and agreed set of quality metrics or common quality indicators for maintainability, since they are distributed across several tools, dashboards, and literature. A quality indicator gives us a hint regarding the manifestation of a given quality aspect, such as the maintainability as part of the ISO/IEC 25010:2011 [4]. Therefore, we have to identify quality indicators and collect them in a so-called quality report. On the basis of this report, an expert should be able to assess the maintainability of a software product. Here, an expert is characterized by technical and domain knowledge of the software product.

For developing a quality report regarding software maintainability assessment, we conducted an exploratory survey across several institutions in research and industry. This survey should reveal essential ingredients of such a quality report. The results enabled us to design a quality report for a specific software product family in the area of the SmartCampus ecosystem [5]. The SmartCampus is a service-oriented system that provides functionality for students, guests, and members of a university to support their daily life. For instance, the CompetenceService as part of the SmartCampus system captures competences of students and offers a semantic search to get suitable candidates for projects in the area of information technology [6]. Besides the mentioned results, we have also gained insight into the importance of quality assessment and software maintainability for different software development project members. We think that our results can be used by experts in order to build their own quality reports for maintainability assessment, since we cannot give a universal quality report due to a missing uniform set of quality indicators.

The paper is structured as follows: In Section II, we lay the foundation for the upcoming sections by defining important terms in the area of software quality assessment regarding software maintainability. Afterwards, we present the related work in Section III to give an overview of the state of the art in this research field. By providing the methodology of this paper in Section IV, we illustrate the design of the study, as well as an overview of its goals and underlying research questions. The results of the conducted study are shown in Section V, followed by the threats to its validity in Section VI. By evaluating the results in Section VII, we answer our research questions and draw conclusions. Finally, in Section VIII we conclude with a summary and give an outlook on further work in this research field.

II. FOUNDATION

According to the standard ISO/IEC 25000:2005, software quality is the "capability of software product to satisfy stated

 ISBN: 978-1-61208-458-9

and implied needs when used under specified conditions" [7]. Software quality assessment is the systematic examination of the extent to which a software product is capable of satisfying these needs [7]. As depicted in Figure 1, software quality can be decomposed into several quality characteristics, sub-characteristics, and attributes [4]. These attributes can again be broken down into quality indicators and measurable quality metrics [8], which are formalized quality indicators [9].

Based on the definition of the term process quality indicator in ISO/IEC 33001:2015 [10], a software quality indicator is an assessment indicator that supports the judgment of software quality characteristics. According to the standard IEEE 1061-1998, a quality metric is a "function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality" [11].

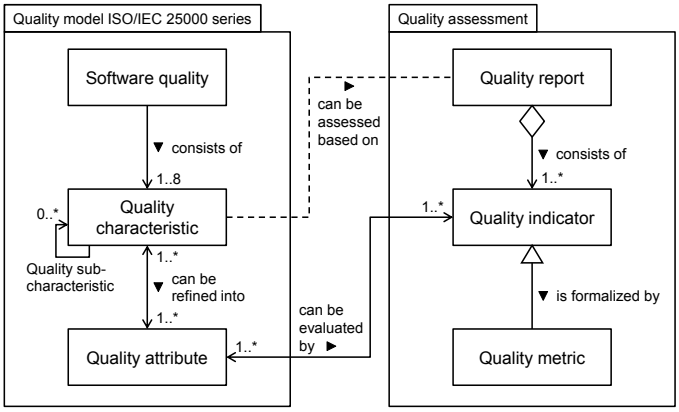


Figure 1. Meta model for software product quality [7] and quality assessment.

In our work, we focus on the quality characteristic maintainability. ISO/IEC 25010:2011 defines maintainability as the "degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers" [4].

A so-called quality report comprises software quality metrics and quality indicators in a comprehensive artifact. On the basis of these quality metrics and indicators, an expert should be able to assess the maintainability of a software product under investigation.

III. Related Work

This section summarizes several papers in the context of software quality assessment and the usage of quality reports.

In Ogasawara et al. [12], an approach to software quality management is introduced and a quality report for software products is presented. This report is based on results which are measured regularly and automatically by a software quality evaluation tool, such as the number of loops or comments. However, the authors do not take a closer look at the quality report. Especially, they do not go into detail about the structure, properties, and ingredients of the quality report. Instead, they show a simple sample quality report for reviewing software product quality containing quality metrics, which can be automatically measured and easily understood.

Steidl et al. [13] present a quality control process that combines quality metrics and manual action: Metrics constitute the basis but software experts have to interpret the metric results within their context. This manual action is based on a quality report. Although we pursue a similar idea, the definition of the term quality report given by the authors differs from our definition. According to the authors, a quality report contains the interpretation of the current analysis results, as well as manual code reviews. As opposed to this, we define a quality report as an artifact comprising software quality metrics and quality indicators, which provides the basis for the interpretation. Moreover, the authors do not describe concrete properties and ingredients of the quality report.

Annex E of ISO/IEC 25040:2011 [14] gives guidance on the structure and contents of an evaluation report for software product quality. However, the structure and contents are only described on an abstract level. Especially, no concrete quality metrics are listed.

As opposed to this, Riaz et al. [15] summarize 15 studies regarding software maintainability prediction and metrics. Table 6 of their paper comprises 45 successful software maintainability metrics gathered at source code level. For example, it contains most of the metrics presented in the well-known Chidamber & Kemerer (CK) metrics suite [16]. Thus, the authors provide a set of potential ingredients of the quality report. However, they present only an unstructured set of metrics, which they extracted from several papers. This loose collection focuses on size, complexity, and coupling, ignoring metrics related to other aspects like testing, static bug detection, and compliance with conventions. Moreover, they do not consider metrics that have to be analyzed manually. Although this set is a good starting point for our work, it is unclear whether these metrics are adequate ingredients of a quality report for software maintainability assessment in research and industry.

Altogether, we identified three papers that show rough information about quality reports and do not go into detail, especially regarding the properties and ingredients of the reports. In addition, we identified one paper that presents an unstructured set of metrics and thus potential ingredients of the quality report, but does not point out whether these metrics are adequate ingredients of the report.

IV. Methodology

This section presents the methodology applied in the study. It outlines its goals and research questions, the design of the study and the study population.

A. Goals and Research Questions

We conducted this study in order to determine the importance of both quality assessment and maintainability in research and industry. Moreover, we wanted to identify properties and ingredients of a quality report for the purpose of software maintainability assessment. These research goals led to the following four research questions:

RQ1: *Is quality assessment considered to be important in research and industry?* There are miscellaneous techniques supporting the development of software products with high quality. In order to identify the status quo regarding software quality, a quality assessment can be conducted. The survey should show whether a quality assessment is actually considered to be important for software development project members in research and industry.

 ISBN: 978-1-61208-458-9

RQ2: *Is maintainability considered to be important in research and industry?* Several publications underline the importance of maintainability since a huge amount of the total software LCC can be spent on maintenance. The survey should reveal whether maintainability is actually considered to be important for software development project members in research and industry.

RQ3: *Which information should be part of a quality report for the purpose of software maintainability assessment?* The assessment of software maintainability is often based on software quality metrics. However, these metrics are distributed across several tools, dashboards, and literature. Moreover, there are further quality indicators for conducting an assessment. In order to develop a comprehensive quality report, which can be used to assess the maintainability of software, relevant quality metrics and indicators for maintainability assessment have to be identified.

RQ4: *How important are the given quality report properties?* Considering requirements engineering, software requirements should possess several properties, such as the traceability [17]. As with software requirements, a quality report should also possess miscellaneous properties. Since there are some properties, which we consider to be relevant to the quality report, the survey should reveal the importance of each property and identify further properties as appropriate.

B. Study Design

The design of the conducted study comprised three phases. In the initial phase, we planned and prepared the study. As part of this, we identified a free web survey tool, *Umfrage Online* [18], for conducting the survey. We settled on this tool, since it is free, easy-to-use and not subject to restrictions concerning the number of questions and answers.

In the second phase, we developed a checklist regarding the structure and content of a survey based on specialist literature (e.g., [19] [20] [21]). In accordance with this checklist, we created a rough sketch for the survey that contains 16 questions (12 open questions, 3 closed questions and 1 partially closed question). Thanks to feedback from researchers, we revised the survey especially by reducing the number of open questions, since open questions are prone to reduce the response rate [22]. Furthermore, we added some demographic questions. The pretest version of the survey consisted of 23 different questions (8 open questions, 12 closed questions and 3 partially closed questions).

In the third phase, we conducted a field pretest with chosen target subjects. Based on the outcomes of the pretest, there were only minor changes to the survey mainly affecting the wording of the questions and answers.

The final version of the survey consists of 23 optional questions (8 open questions, 12 closed questions and 3 partially closed questions) and is available at [23] in German.

C. Study Population

The population of the study comprises software development project members, such as software developers, software architects or research assistants, from various companies and research organizations.

The link to the online survey was sent to chosen members of several companies and research organizations, who distributed the link within their institution. These members and consequently the recipients of the link were selected using convenience sampling. Since we do not know how many people received the survey link, the initial sample and thus the response rate cannot be determined. In total, 113 people answered the survey including 32 respondents ($\approx 28\%$) who did not complete it. In average, the 81 respondents needed 14 minutes (adjusted, standard deviation: 13 minutes) respectively to complete the survey. For survey evaluation, both the respondents who did and did not complete the survey are taken into account. The survey was written in German and was sent to companies and research organizations in Germany; therefore the results may show a German attitude towards software maintainability assessment. Most of the 66 respondents who stated their occupation work as software developers ($\approx 23\%$), followed by software architects ($\approx 20\%$), research assistants ($\approx 18\%$), and project or group leaders ($\approx 12\%$). Approximately 6% of the survey respondents are students and about 21% are occupied otherwise.

As depicted in Figure 2, the respondents work at companies and research organizations with highly diverse size.

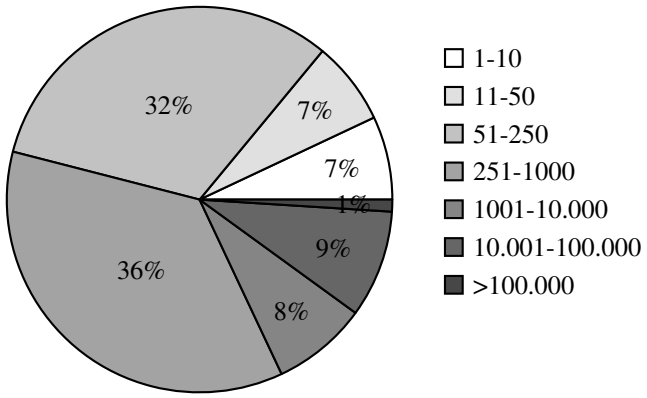


Figure 2. Size of companies and research organizations of survey respondents by the number of employees (n = 77).

Figure 3 shows that the survey covers software development project members working in the domain of software engineering for less than one year up to more than ten years, whereby the latter was stated by more than 30% of the respondents who answered the corresponding question.

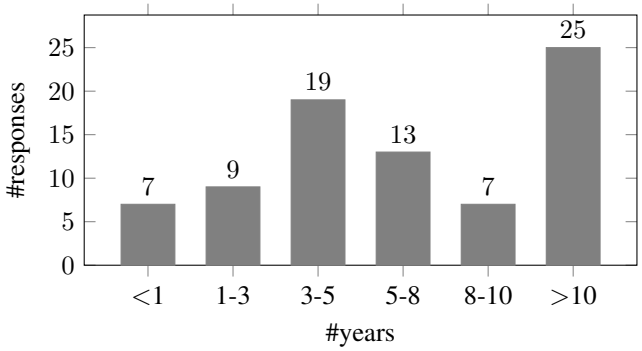


Figure 3. Number of years that respondents are working in the domain of software engineering (n = 80).

 ISBN: 978-1-61208-458-9

Furthermore, nearly 30% of the 81 respondents who answered the corresponding question stated that they have already participated in a software audit. According to the standard IEEE 1028-2008, an audit is an "independent examination of a software product [...] to assess compliance with specifications, standards, contractual agreements, or other criteria" [24]. About 30% of the 96 respondents who stated whether they have already read or created a quality report answered in the affirmative. If we separate the answers from research and industry, about 71% of the respondents from industry have already participated in a software audit and 63% have already read or created a quality report, compared with about 12% and 17%, respectively, in research. Due to the great amount of respondents from industry who are familiar with software audits and quality reports, and thus software quality assessment, we distinguish between answers from research and industry in the following sections.

V. RESULTS

This section outlines the outcomes of the online survey. By reason of space limitations, we only present the survey questions which are most relevant for our research questions.

Question 1: How important is the quality assessment of already existing software for you? The first question should show the importance of quality assessment for our respondents. This question explicitly refers to the quality assessment of already existing software, e.g., in the context of a software audit. Since we aim to develop a quality report for software quality assessment, we should first determine whether there is a demand for assessing software quality. The participants of the survey could answer this question on a five-point ordinal scale: "very important", "important", "partly important", "less important", and "unimportant". Figure 4 shows the frequency distribution of all answers given by 108 respondents.

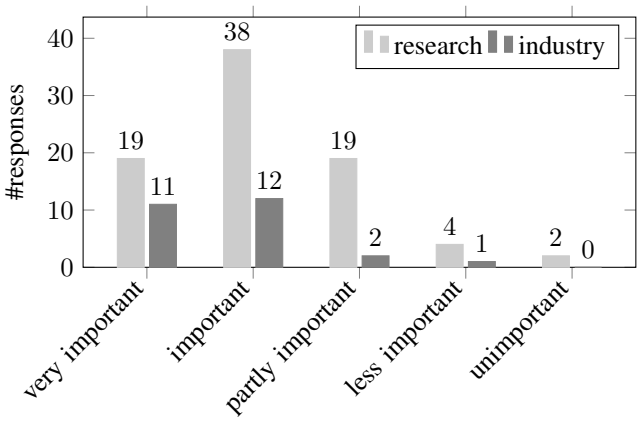


Figure 4. Importance of software quality assessment, absolute (n = 108).

The answers are coded with a scale ranging from 1 to 5, where 1 codes "very important" and 5 codes "unimportant". As depicted in Figure 4, the large majority of the respondents answered with "very important" or "important", namely almost 75%. Calculating some statistical numbers based on the coding lead to the following results: arithmetic mean = 2.06, median = 2, standard deviation = 0.91. As the median indicates, most of the respondents selected the answers "very important" or "important".

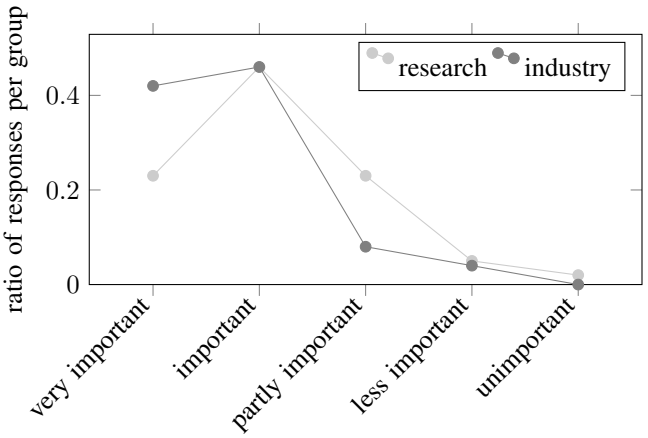


Figure 5. Importance of software quality assessment, normalized (n = 108).

Figure 5 shows that there is a difference between respondents from research and industry. More than 42% of the respondents from industry answered with "very important" compared to about 23% of the respondents from research. This result suggests a more distinct quality awareness in the industry. Nevertheless, the responses to this question show that software quality assessment is important for respondents from both research and industry. Hence, this question reinforces our decision to develop a quality report that aims to support and simplify software quality assessment.

Question 2: How important is software maintainability for you? The second question should reveal the importance of maintainability for our respondents. Due to the fact that the quality report should focus on maintainability initially, the importance of maintainability is crucial to us. The scale of this question and the coding correspond to those in the first question. Figure 6 shows the frequency distribution of all answers given by 91 respondents.

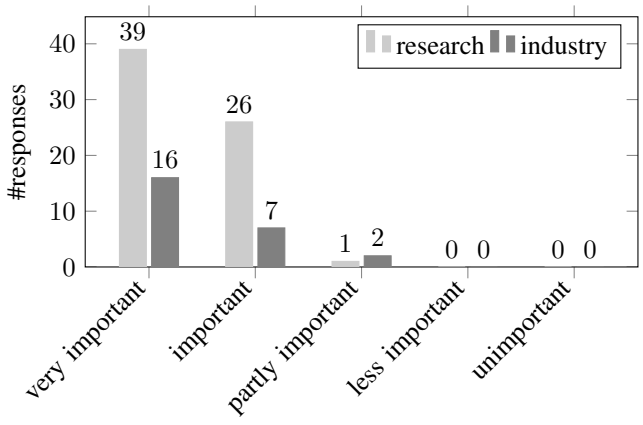


Figure 6. Importance of software maintainability, absolute (n = 91).

As depicted in Figure 6, not a single respondent answered with "less important" or "unimportant". Instead, approximately 97% of the respondents selected the answers "very important" or "important". Quantifying the answers based on the coding lead to the following results: arithmetic mean = 1.43, median = 1, standard deviation = 0.56.

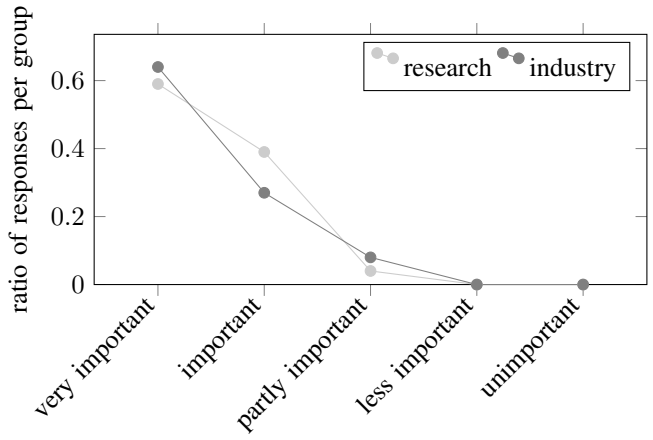


Figure 7. Importance of software maintainability, normalized (n = 91).

In contrast to the first question, Figure 7 shows that there is no significant difference between the answers given by the respondents from research and industry. The responses to this question reveal that maintainability is very important for both groups. Therefore, this question reinforces our decision that the quality report should focus upon maintainability initially.

Question 3: Which information should a quality report provide for you in order to assess the maintainability of a software product? By means of the third question, essential ingredients of the quality report for software maintainability assessment should be determined. Therefore, we conducted a brainstorming session in order to identify relevant answers to this closed question. The identified answers are not only quality metrics, but also more general quality indicators, such as the compliance with conventions. These answers are shown in Figure 8. Beyond this closed question, we created a subsequent open question asking for further information, which should be part of the quality report. Since the determination of ingredients is an essential task for developing a quality report, these two questions can be understood as key issues. Figure 8 depicts the frequency distribution of all answers given by 83 respondents.

Figure 8 shows the number of votes for each software quality metric or quality indicator we identified in our brainstorming session. The most popular answers are “quality of comments” and “understandability of documentation”, the most unpopular one is “cyclomatic complexity”. Figure 8 depicts the answers in order of declining popularity from the most popular answer on the left to the most unpopular answer to the right. However, there is no obvious point which separates the answers in two parts: ingredients and non-ingredients of the quality report. Instead, the most popular answers were selected by about 80% of the respondents and the most unpopular one by about 20%. Therefore, we cannot identify any quality metric or quality indicator which is indispensable or dispensable for the quality report. Moreover, there are no respondents who selected exactly the same answers. These facts lead us to the assumption that there is no uniform set of quality metrics and indicators for the quality report for software maintainability assessment.

Even the 18 responses to the related open question strengthen our assumption. Out of 12 proposed quality metrics and indicators in total, there are nine metrics and indicators which are only mentioned by one respondent. A possible explanation for our assumption is that the ingredients of a quality report depend on different aspects. For example, if we want to assess the maintainability of a web service based on SOAP, the quality report does not have to contain information about the compliance with RESTful best practices. Moreover, the content of the quality report seems to depend on the objective of the assessment. For instance, if we want to assess the readability of a software product, we usually do not need information about the test coverage. Due to the fact that there are no respondents who selected exactly the same answers, the ingredients of the quality report also appear to depend on the software expert who conducts the assessment. Hence, the assessment of maintainability seems to be a subjective task, e.g., based on the experience of the software expert.

Thanks to the related open question, we identified several additional quality metrics and indicators, which can be part of the quality report. These potential ingredients are listed below in descending order of popularity: results of static code analysis, compliance with software development principles, description of fundamental design decisions, application of best practices, maturity level of used technologies, design of interfaces, results of architectural code reviews, description of the basic architecture, age of the examined software product, usage of dependency injection, comprehensibility of the source code and development practices. Due to the fact that the most popular response to this question is stated only by five respondents, we do not elaborate on it.

Separating the respondents from research (n = 60) and industry (n = 23), Figure 8 reveals some differences between both groups. The most important quality indicator for the respondents from industry provides information about the technologies which were used during development. Moreover, the quality indicator concerning the functional naming of classes and methods is more relevant for respondents from industry than for respondents from research. As opposed to this, the completeness of the documentation is more important for respondents from research. Determining the number of responses for each answer relative to the number of respondents from research and industry respectively, the mean deviation between both groups amounts to about eleven percentage points. This indicates that the responses from both groups are quite similar. Merely, the three mentioned quality indicators above and the cyclomatic complexity are considerable outliers (deviation >20 percentage points).

Furthermore, the answers to this question point out that the quality report may contain not only software quality metrics and indicators which can be measured automatically. Instead, it may also contain quality metrics and indicators which have to be determined manually, such as the quality of comments or the understandability of documentation.

Since well-known quality metrics regarding software maintainability are mainly located on the right-hand side of Figure 8, such as the number of packages, comment ratio or cyclomatic complexity, these metrics seem to be less important for software maintainability assessment.

Question 4: How important are the given quality report properties for you? The fourth question should reveal the importance of several quality report properties for our

 ISBN: 978-1-61208-458-9

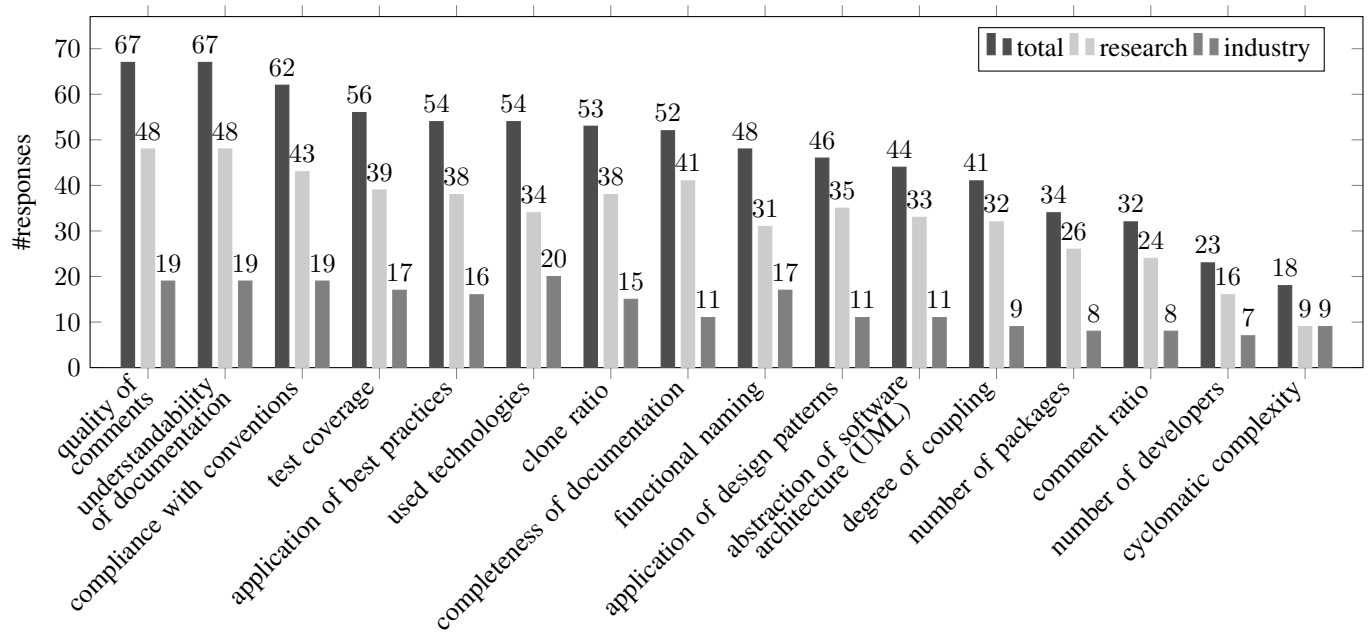


Figure 8. Ingredients of the quality report for software maintainability assessment (n = 83).

respondents. For this purpose, we derived eight properties from different standards and papers published in the software quality field (e.g., [25] [26] [27]). These properties are shown in Figure 9. Beyond this closed question, we created a subsequent open question asking for further quality report properties. Since the quality report has to possess certain properties in order to be used effectively and efficiently, these two questions are crucial to us. The participants of the survey could rate each property on a five-point ordinal scale: "very important", "important", "partly important", "less important", and "unimportant". Alternatively, the respondents could answer with "not judgeable". The number of answers ranges from 80 to 81 per property including 2 to 7 responses with "not judgeable" respectively. As these responses were counted as missing answers, Figure 9 shows the importance of the given properties based on 74 to 79 responses per property.

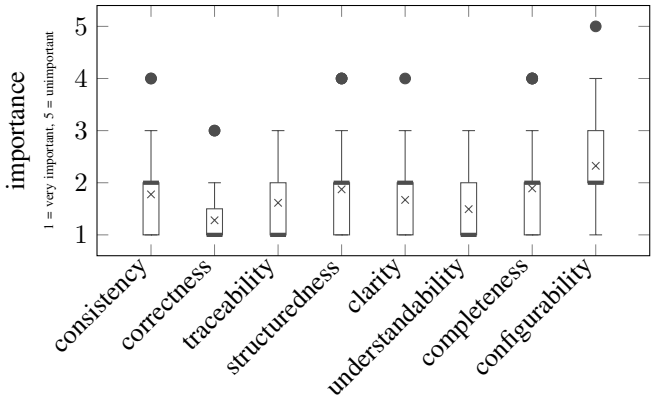


Figure 9. Importance of several quality report properties (n = 74 to 79, median = thick line, arithmetic mean = cross, outlier = dot).

The answers are coded with a scale ranging from 1 to 5, where 1 codes "very important" and 5 codes "unimportant". Figure 9 contains a boxplot, which comprises a box for each quality report property. The bottom and the top of each box are the first and third quartiles, whereas the thick line inside the box depicts the median (second quartile). The lower and the upper whiskers (horizontal lines outside a box) denote the minimum and maximum, respectively, of the coded answers with maximum 1.5 interquartile range. Furthermore, the crosses visualize the arithmetic means and the dots illustrate outliers.

The analysis of the responses shows that all of the given quality report properties are considered as "very important" or "important", due to the fact that the median ranges from 1 to 2 and the arithmetic mean ranges from 1.3 to 2.3.

Upon closer examination, the three most important properties are correctness, traceability, and understandability with a median of 1 and the lowest values for the arithmetic mean. The least important property, but still important with a median of 2 and an arithmetic mean of 2.3, is the configurability of the quality report. This result is a bit surprising considering the outcome of question 3. Question 3 suggests that the ingredients of the quality report depend on different aspects, e.g., the used technologies or the objective of the assessment. In order to equip the quality report with the required ingredients, it has to be configurable. While answering the survey, the respondents possibly think of several concrete quality reports instead of a general report. Therefore, the configurability is circumstantial. However, in our work we aim to develop a general quality report from which we deduce these concrete reports by adapting the general one. Hence, the configurability is a property which we deem very important.

Separating the respondents from research (n = 54 to 57) and industry (n = 19 to 22) shows that the importance of the given quality report properties is quite similar for both groups. There is only one difference in the median concerning the

 ISBN: 978-1-61208-458-9

traceability (research: 1.5, industry: 1). Moreover, the average difference between the arithmetic means of each property for respondents from research and industry is 0.14, which supports our statement.

The responses to the related open question yield merely two usable properties, simplicity and robustness. Both properties were proposed by one respondent respectively. Therefore, we do not elaborate on it.

Question 5: Do you use tools for static code analysis? If so, which tools? By means of the fifth question, the usage of tools for static code analysis should be determined and relevant tools should be identified. Primarily, the respondents were asked whether they use tools for static code analysis. If so, they were asked which tools they use. Therefore, we conducted a brainstorming session in order to identify several tools for static code analysis. In addition to these given answers, the respondents could state further tools. Since software quality metrics and indicators provided by static code analysis tools are potential ingredients of the quality report, this question matters to us. Figure 10 shows the number of votes for the different tools. Due to space limitations, we only present the six most popular tools out of 27 in total.

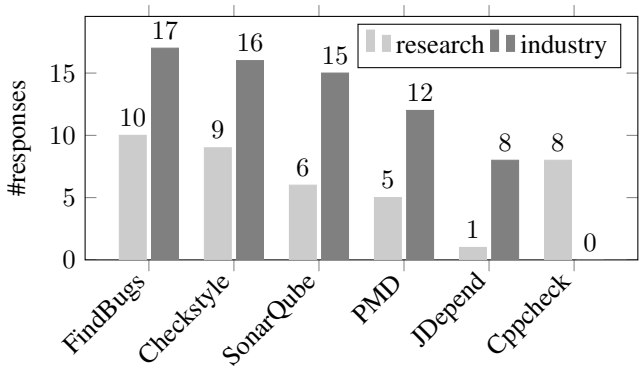


Figure 10. Most popular tools for static code analysis (n = 47).

Altogether, 49 out of 83 respondents who answered the corresponding question stated that they use tools for static code analysis. Separating the respondents from research (n = 59) and industry (n = 24), 88% of the respondents from industry use these tools in contrast to 47% from research. Although the usage of static code analysis tools seems to be more common in industry than in research, these answers underline the importance of quality assessment for both groups.

Figure 10 contains the six most popular tools, which are stated by 47 respondents. All these tools are well-known and provide a huge set of software quality metrics and quality indicators, which can be used for software maintainability assessment.

Considering the three tools for Java code, FindBugs, Checkstyle, and JDepend, we make an interesting discovery. Altogether, 22 out of the 32 respondents who selected at least one of these tools use two or all of them. Moreover, these tools support the same programming language. These two facts support the statement that currently quality metrics and indicators are distributed across several tools and dashboard. Hence, a comprehensive quality report comprising relevant software quality metrics and indicators provided by these tools and dashboards seems to be useful.

VI. Threats to validity

This section comprises an analysis of potential aspects threatening the validity of the survey and its outcomes.

The population of the study comprises software development project members from companies and research organizations of different size (see Figure 2). Moreover, we got responses from participants with a great diversity in experience regarding the number of years they are working in the domain of software engineering (see Figure 3). Therefore, the study population does not pose a threat to the validity.

In total, 113 people answered the survey, 86 from research and 27 from industry. Here, the relatively small number of respondents from industry may threat the validity of the survey and its outcomes.

The survey was written in German and was sent to companies and research organizations in Germany. This definitely is a threat to the validity of the survey and its outcomes, since different cultures and customs may imply different attitudes towards software maintainability assessment.

VII. Answers to research questions

In this section, we summarize the answers to the four research questions mentioned in Subsection IV-A.

RQ1: *Importance of quality assessment in research and industry:* Due to the answers to question 1 and question 5, we come to the conclusion that quality assessment is considered to be an important task, e.g., in order to identify areas of improvement. This is true for respondents from both research and industry, although the answers suggest more distinct quality awareness in industry.

RQ2: *Importance of software maintainability in research and industry:* The answers to question 2 lead us to the conclusion that software maintainability is a very important quality characteristic for respondents from both research and industry.

RQ3: *Ingredients of the quality report for software maintainability assessment:* Due to the answers to question 3, we conclude that there is no uniform set of quality metrics and indicators for the quality report for software maintainability assessment. Instead, the ingredients of the quality report seem to depend on different aspects, e.g., the used paradigms. Therefore, we cannot identify any quality metric or indicator which is indispensable or dispensable for the quality report. In addition, the answers to this question point out that the quality report may contain not only quality metrics and indicators which can be measured automatically. It may also contain quality metrics and indicators which have to be determined manually, e.g., due to the fact that domain knowledge is required.

RQ4: *Importance of quality report properties:* The answers to question 4 outline that all given quality report properties are considered as very important or important for respondents from both research and industry. Correctness, traceability, and understandability are the properties considered to be most important.

 ISBN: 978-1-61208-458-9

VIII. CONCLUSION

Today, the majority of software life cycle costs is spent on maintenance. The systematic or even automatic evaluation of software regarding its maintainability by means of a quality report might help to reduce these costs. However, there is no uniform understanding about the metrics and quality indicators required to assess the maintainability of software. For that reason, we showed the structure and the results of an exploratory survey for developing an appropriate quality report.

With our survey, we reached 113 respondents, partially software developers, software architects, research assistants, and project or group leaders. The survey presented us the following three key finding: 1) Our initial assumptions about the importance of quality assessment and maintainability of software were confirmed by the first questions of the survey. There was not one respondent who considers that software maintainability is less or unimportant. 2) The answers to question 3 showed that there is no uniform set of quality metrics and indicators for the quality report for software maintainability assessment. Instead, the software expert who conducts the assessment has to determine the ingredients of the quality report depending on different aspects. 3) The answers to question 3 point out that the quality report may contain not only quality metrics and indicators which can be measured automatically. Instead, it may also contain metrics and indicators which have to be determined manually.

In the future, we will further work on the development of a quality report for assessing software maintainability. On the one hand, as we could not identify concrete metrics or indicators being indispensable or dispensable for the quality report, we will categorize the identified metrics and indicators initially. Subsequently, we will try to identify additional metrics and indicators for each category based on literature research and an examination of several tools for static code analysis. These metrics and indicators should be understood as potential ingredients, which can be added to or removed from the quality report. On the other hand, we will develop a hybrid approach, which combines automatic and manual analyses in order to generate a quality report tool-based and with minimal effort. The basic idea for such an approach already exists [9] and will be seized by us.

ACKNOWLEDGMENT

We would like to thank all survey participants for their participation. Furthermore, we would like to thank the members of the research group Cooperation & Management (C&M), Karlsruhe Institute of Technology (KIT), and the engineers of iteratec GmbH for their valuable comments and suggestions regarding the development of the survey.

REFERENCES

[1] NASA, "Software Design for Maintainability," URL: https://oce.jpl.nasa.gov/practices/dfe6.pdf [accessed: 2015-08-08].

[2] T. Pearse and P. Oman, "Maintainability Measurements on Industrial Source Code Maintenance Activities," Proceedings of International Conference on Software Maintenance, 1995, pp. 295–303, ISSN: 1063-6773.

[3] J. Choudhari and U. Suman, "An Empirical Evaluation of Iterative Maintenance Life Cycle Using XP," ACM SIGSOFT Software Engineering Notes, vol. 40, no. 2, 2015, pp. 1–14, ISSN: 0163-5948.

[4] ISO/IEC, "Std 25010:2011: Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models," 2011.

[5] M. Gebhart, P. Giessler, P. Burkhardt, and S. Abeck, "Quality-Oriented Requirements Engineering of RESTful Web Service for Systemic Consenting," International Journal on Advances in Software, vol. 8, no. 1&2, 2015, pp. 156–166, ISSN: 1942-2628.

[6] P. Giessler, M. Gebhart, D. Sarancin, and S. Abeck, "Best Practices for the Design of RESTful Web Services," Tenth International Conference on Software Engineering Advances (ICSEA 2015), 2015, ISSN: 2308-4235, to be appear.

[7] ISO/IEC, "Std 25000:2005: Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE," 2005.

[8] M. Gebhart and S. Sejdovic, "Quality-Oriented Design of Software Services in Geographical Information Systems," International Journal on Advances in Software, vol. 5, no. 3&4, 2012, pp. 293–307, ISSN: 1942-2628.

[9] M. Gebhart, "Query-Based Static Analysis of Web Services in Service-Oriented Architectures," International Journal on Advances in Internet Technology, vol. 7, no. 1&2, 2014, pp. 136–147, ISSN: 1942-2652.

[10] ISO/IEC, "Std 33001:2015: Information technology – Process assessment – Concepts and terminology," 2015.

[11] IEEE, "Std 1061-1998: IEEE Standard for a Software Quality Metrics Methodology," 1998.

[12] H. Ogasawara, A. Yamada, and M. Kojo, "Experiences of Software Quality Management Using Metrics through the Life-Cycle," Proceedings of the 18th International Conference on Software Engineering, 1996, pp. 179–188, ISSN: 0270-5257.

[13] D. Steidl, F. Deissenboeck, M. Poehlmann, R. Heinke, and B. Uhink-Mergenthaler, "Continuous Software Quality Control in Practice," 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2014, pp. 561–564, ISSN: 1063-6773.

[14] ISO/IEC, "Std 25040:2011: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Evaluation process," 2011.

[15] M. Riaz, E. Mendes, and E. Tempero, "A Systematic Review of Software Maintainability Prediction and Metrics," 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM), 2009, pp. 367–377, ISSN: 1938-6451.

[16] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994, pp. 476–493, ISSN: 0098-5589.

[17] ISO/IEC/IEEE, "Std 29148:2011: Systems and software engineering – Life cycle processes – Requirements engineering," 2011.

[18] enuvo GmbH, "Umfrage Online," URL: https://www.umfrageonline.com/ [accessed: 2015-07-27].

[19] D. A. Dillman, Ed., Mail and Internet Surveys – The Tailored Design Method. John Wiley & Sons Inc., Hoboken, New Jersey, Jun. 2007, ISBN: 978-04-70-03-85-6.

[20] A. Bryman, Ed., Social Research Methods. Oxford University Press Inc., New York, Mar. 2008, ISBN: 978-01-99-20-29-5.

[21] L. Bickman and D. J. Rog, Eds., The SAGE Handbook of Applied Social Research Methods. SAGE Publications Inc., Thousand Oaks, California, 2009, ISBN: 978-14-12-95-03-1.

[22] D. Robbins, Ed., Understanding Research Methods – A Guide for the Public and Nonprofit Manager. Taylor & Francis Group LLC, Abingdon, 2009.

[23] M. Gerster and P. Giessler, "Entwicklung eines Qualitätsberichts," 2015, URL: https://www.umfrageonline.com/s/7e3cb3d [accessed: 2015-08-07].

[24] IEEE, "Std 1028-2008: IEEE Standard for Software Reviews and Audits," 2008.

[25] ISO/IEC, "Std 25012:2008: Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Data quality model," 2008.

[26] R. Plösch, A. Dautovic, and M. Saft, "The Value of Software Documentation Quality," 14th International Conference on Quality Software (QSIC), 2014, pp. 333–342, ISSN: 1550-6002.

[27] R. E. Al-Qutaish, "Quality Models in Software Engineering Literature: An Analytical and Comparative Study," Journal of American Science, vol. 6, no. 3, 2010, pp. 166–175, ISSN: 1545-1003.

 ISBN: 978-1-61208-458-9

A Tree-Based Approach to Support Refactoring in Multi-Language Software Applications

Hagen Schink, David Broneske*, Reimar Schröter*, and Wolfram Fenske*
University of Magdeburg, Germany
Email: hagen.schink@gmail.com, *{ david.broneske, reimar.schroeter, wolfram.fenske}@ovgu.de

***Abstract*—Developers build software applications using different programming languages, so they can benefit from the programming languages' specific advantages. To allow an interaction of different programming languages, each programming language offers Application Programming Interfaces (API) to be called. However, such interactions pose challenges for source-code refactoring across programming languages. To this end, we present a generalized approach to refactoring in multi-language software applications based on graphs of trees. To illustrate the broad application of our approach, we implement a library that builds the foundation for two tools that support the refactoring of database applications implemented in Java and Java applications that invoke code of the functional programming language Clojure.**

Keywords–refactoring; multi-language software application; Java, Clojure, Relational Database

I. INTRODUCTION

Programming languages provide different language constructs for the description of algorithms. Depending on the language constructs, a developer's effort to implement an algorithm may differ between programming languages. Hence, if a developer is able to choose another programming language for each problem in a single system, she can describe the solution with a minimum of effort. Consequently, developers use different programming languages in concert to implement software applications [1]–[8]. We call such a software application implemented by means of different programming languages a multi-language software application (MLSA) [5].

Irrespective of the programming language at hand, refactoring is a common technique to modify a source-code's structure while preserving the source-code's semantics [9]. Refactorings are used to improve the maintainability and extensibility of a code base. A number of refactoring transformations exist for different programming languages and programming paradigms, such as object-oriented programming-languages [9][10], functional programming-languages [11], and relational schemata [12].

However, refactoring transformations are defined for single programming languages and do not consider the interaction of languages in an MLSA. Thus, applying a refactoring on source code of one language can break the interaction of languages within an MLSA. For instance, in a database application, renaming a table breaks the application code that depends on the original table name [13]. Since compilers do not check language interaction at compile time, developers need a sufficient test coverage to detect the broken interaction or, otherwise, the broken system goes into production.

In this paper, we present a generally applicable concept based on graphs of trees that supports developers in checking and preserving language interaction within an MLSA. To show the practical applicability of our concept, we present two prototypes that support refactoring in applications that use (1) Java and a relational database and (2) Java and the functional programming language Clojure [14]. Additionally, we provide a brief discussion of the concept's performance and discuss the concept's generality in respect to different MLSA setups.

The paper is structured as follows: In Section II, we introduce different realizations of MLSAs and give examples of how refactoring affects language interaction in MLSAs. In Section III, we describe and justify our concept for supporting refactoring in MLSAs. In Section IV, we present two tools, sql-schema-comparer and clojure-java-interface-checker, which implement our concept for two different language combinations. We discuss different aspects of the concept in Section V. Finally, we present related work in Section VI before we conclude the paper in Section VII.

II. BACKGROUND

In this section, we first describe different approaches to implement language interaction in MSLAs. Then, we describe how refactoring can break language interaction in MLSAs by means of a database application.

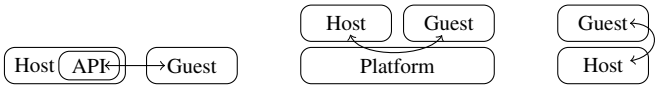
A. Implementations of Language Interaction in MLSAs

In general, a software application contains source code written in one programming language that initializes that application. In an MLSA, we call the programming language in which the application's initialization code is written the application's *host language*. From the code of the host language, developers invoke source code implemented in other languages. We call the invoked languages *guest languages*. Based on this definition, we distinguish three realizations of language interaction:

1) Foreign Function Interface (FFI) [15]
2) Host and guest language share the same platform
3) Guest language is implemented in the host language

The first realization, FFI, describes APIs, which allow developers to use host language syntax elements for accessing syntax elements in a guest language (cf. Figure 1a). For instance, in Java the Java Database Connectivity (JDBC) and the Java Persistence API (JPA) allow developers to query relational databases via SQL or an object-relational mapping, and the Java Native Interface (JNI) allows developers to invoke C/C++ functions. Platforms such as Java and .NET represent the second realization, in which both, host and guest language, share the same platform (see Figure 1b). For instance, the programming languages C#, Visual Basic .NET, and F# can all be compiled to the Common Intermediate Language (CIL) and

 ISBN: 978-1-61208-458-9



(a) FFI (b) Platform (c) Language

Figure 1. Relations between host and guest language.

all languages can invoke CIL code [16]. Thus, by compiling the source code of the guest language to the platform of the host language, a developer can call the code implemented in the guest language from the host language. The third realization (see Figure 1c) describes programming languages such as Lisp or Ruby, which provide meta-programming features that allow developers to implement new language elements or DSLs that represent the guest language [17]–[19].

In case a guest language is implemented in the host language (cf. Figure 1c), the guest language's source code is actually valid source code of the host language. Thus, developers can reuse existing tools for static code analysis of the host language to check the interaction between source code of the guest and the host language. In case the guest language's source code can be compiled to the host language's platform (cf. Figure 1b), developers can reference the compiled code of the guest language in the host language and the host language's compiler can check the interaction between source code of the guest and the host language [16]. For instance, F# source code can be compiled to a managed DLL which a developer can reference in C# source code. In contrast, if languages interact by means of an FFI (cf.Figure 1a), existing tools for the host language cannot check language interactions.

B. Refactoring of MLSAs that use FFIs

Since there is no tool support for refactoring languages that communicate using FFIs, developers have to assure language interaction manually. Currently, developers must provide automated tests with a reasonable test coverage to reliably detect broken language interactions after source-code modifications such as refactoring. In former work, we discussed problems in the case of Java to SQL interactions [13].

For the Java programming language, there are two APIs to interact with a relational database via SQL: JDBC and JPA. JDBC allows developers to directly execute SQL statements. For instance, Figure 2 shows an SQL query that selects the values of column `label` of table `departments` that start with *M*.

JPA, in turn, provides an interface for object-relational mapping (ORM). The ORM maps Java classes to relational tables and Java methods to table columns. An object relational mapper can automatically create SQL statements based on the ORM. Developers can use JPA to access a relational database without having to write plain SQL. For instance, Figure 3 shows how a developer can map the Java class `Department` to the relational table `departments` with its columns `id` and `label`.

In the following, we explain how refactoring either the Java source code or the relational schema affects the interaction between the Java application and the relational database.

```
1  String stmt = "SELECT label FROM departments "
2              + "WHERE label LIKE ?";
3  PreparedStatement query = con.prepareStatement(stmt);
4
5  query.setString(1, "M%");
6  ResultSet result = query.executeQuery();
```

Figure 2. JDBC: Parameterized query for department names.

```
1  @Entity
2  @Table(name="departments")
3  public class Department implements Serializable {
4
5    private int id;
6    private String label;
7
8    public void setId(int id) { this.id = id; }
9    @Id
10   public int getId() { return id; }
11
12   public void setLabel(String label) { this.label = label; }
13   public String getLabel() { return label; }
14 }
```

Figure 3. JPA: Annotated class `Department`.

a) Host-Language Refactoring: Given that we use JPA to access the relational database, we rename the property `label` of class `Department` (cf. Figure 3) to `name`, because `name` is more specific than the more general `label`. Consequently, to be consistent, we also rename the methods `setLabel` and `getLabel` to `setName` and `getName`, respectively. This refactoring breaks the application because the ORM cannot find a matching column `name` for table `departments` in the database schema.

b) Guest-Language Refactoring: Let us assume we apply a Rename Column refactoring to rename the column `label` of table `departments` to `name`. This refactoring breaks the application regardless of whether we use JDBC or JPA because neither the SQL statement (cf. Figure 2) nor the ORM (cf. Figure 3) reference the renamed column.

III. Checking and Preserving Language Interaction

The main idea of our concept to check and preserve language interactions is as follows. First, we extract those syntax elements from the host and guest language that are involved in language interaction and represent these syntax elements in graphs of trees for the host and the guest language. Second, by comparing the graph of the host and the graph of guest language with each other, we are able to detect broken language interactions. Now, we present this approach in detail.

A. Modeling Language Interaction

For the source code of a guest language C_G and the source code of a host language C_H, we call the references to the guest language's source code extracted from the host language's source code $R_{C_H \to C_G}$. Additionally, we call the syntax elements in the guest language's source code involved in language interaction R_{C_G}. Based on the representation as abstract syntax trees (AST) [20], we describe $R_{C_H \to C_G}$ and R_{C_G} as sets of

labeled trees. Consequently, $r_{C_H \to C_G} \in R_{C_H \to C_G}$ is a tree representing a single invocation of a guest language structure element in the host language and $r_{C_G} \in R_{C_G}$ is a tree representing a single structure element defined in the guest language that is involved in language interaction.

Since guest languages do not define a uniform set of syntax elements for language interaction, syntax elements for language interaction can be different between guest languages. Consequently, we cannot use a single type of tree to represent all syntax elements involved in language interaction, because we neither can oversee all current syntax elements, nor foresee all future elements involved in language interaction. Thus, we need to define specialized types of trees, which only represent the syntax elements in the guest language that are actually involved in language interaction. For instance, with JDBC or JPA, the table and column identifiers defined in the relational database are elements involved in language interaction. In contrast, with JNI, a function's name and parameters in C source code are elements involved in language interaction.

B. Checking the Referential Integrity between Languages

We check the referential integrity between languages by comparing all trees in $R_{C_H \to C_G}$ with the trees in R_{C_G}. To ensure the referential integrity, for all $r_{C_H \to C_G} \in R_{C_H \to C_G}$ there must be one $r_{C_G} \in R_{C_G}$, so that the following condition is satisfied:

$$r_{C_H \to C_G} \text{ is a top-down subtree of } r_{C_G} \quad (1)$$

However, this precondition is not sufficient because nodes may be missing in $r_{C_H \to C_G}$ that are mandatory for language interaction. For instance, in JDBC, if a developer defines an *INSERT* statement, this statement must provide values for all columns of the referenced tables with a Not Null constraint, or else the statement fails. Hence, for the set of mandatory nodes r_M, we additionally have to check if

$$r_M \subseteq r_{C_H \to C_G} \quad (2)$$

The set r_M is defined as follows for mandatory nodes m and the function $parent(x)$, which returns x's parent node:

$$r_M = \{m \mid m \in r_{C_G} \wedge parent(m) \in r_{C_H \to C_G}\} \quad (3)$$

In Figure 4, we illustrate the process of checking language interaction: First, we need to extract $R_{C_H \to C_G}$ from C_H and R_{C_G} from C_G. In accordance with (3), we compute the set of mandatory nodes R_M for R_{C_G} and $R_{C_H \to C_G}$. Then, we can compare the extracted references. The comparison returns a result which contains the possibly empty sets $R_{C_H \to C_G} \setminus R_{C_G}$ and $R_M \setminus R_{C_H \to C_G}$, i.e., the elements, which were extracted from the host language source code but are missing in the guest language source code, as well as the mandatory elements defined in the guest language source code that are not referenced in the host language source code. Language interaction is preserved if both sets are empty. Otherwise, the sets contain the syntax elements, which are involved in the broken language interaction.

IV. Implementing a General Approach to MLR

In this section, we first introduce the structure-graph library [21]. The structure-graph library implements an algorithm which we use to implement the sql-schema-comparer and the clojure-java-interface-checker. The sql-schema-comparer checks the language interaction between Java source code and a relational database schema and the clojure-java-interface-checker checks the language interaction between Java and Clojure source code.

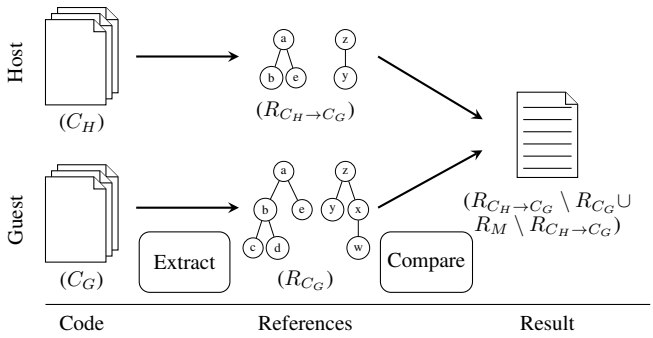


Figure 4. The process of checking language interaction.

A. The Structure-Graph Library

The structure-graph library provides a prototypical implementation of a comparison algorithm for trees of syntax elements as defined in Section III-A. The library takes two arguments: a source graph G_S and a target graph G_T. Both graphs represent sets of trees. Each node v of a tree has a *name* and a *path*. A node is uniquely identified by its name and path; thus, name and path represent the node's *id*. The comparison of G_S and G_T returns a list of modified nodes. The library distinguishes two node modifications: *Added* and *Deleted*. A node v is *added* if $id(v) \in G_T \wedge id(v) \notin G_S$. Conversely, a node v is *removed* if $id(v) \notin G_T \wedge id(v) \in G_S$. For checking language interaction, we pass $R_{C_H \to C_G}$ as G_S and R_{C_G} as G_T to the library. Consequently, $R_{C_H \to C_G} \setminus R_{C_G}$ is the set of added nodes and $R_M \setminus R_{C_H \to C_G}$ is the set of removed nodes that are marked as mandatory, respectively.

B. Java to Relational Database Interaction

In [22], we presented the sql-schema-comparer (SSC) library that applies the structure-graph library to detect mismatches between a relational database schema and a schema expected by the interacting Java source code. To this end, SSC extracts the expected schema $R_{C_H \to C_G}$ from SQL statements and JPA entities defined in the Java source code and the actual schema R_{C_G} from a relational database. The representation of R_{C_G} contains a tree for each table. Each tree contains a root that holds the table's name. The root has child nodes for each table column, and each column has child nodes for each column constraint (cf. Figure 5). Since SSC marks a column as mandatory if a Not Null constraint is specified on that column, SSC is able to check if all columns required for an insertion are referenced by an SQL statement or JPA entity. The representation of $R_{C_H \to C_G}$ contains a tree for each SQL statement and JPA entity defined in the interacting Java source code. The trees of the expected schema only contain nodes for the referenced table names and columns, because constraints are not referenced in the Java source code. For brevity, we do not discuss column type information here.

C. Java to Clojure Language Interaction

Now, we introduce the Clojure programming language and describe how the clojure-java-interface-checker preserves the interaction between Java and Clojure source code.

 ISBN: 978-1-61208-458-9

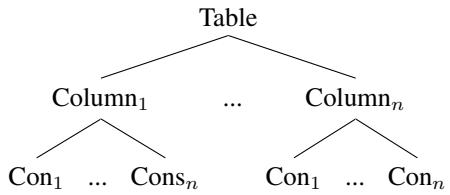


Figure 5. Database Schema Graph.

1) The Clojure Programming Language: Clojure is a functional programming language that runs on the Java Virtual Machine (JVM). Syntactically, Clojure is a Lisp dialect. Since Clojure runs on the JVM, developers can directly use libraries available for the programming language Java.

Since Clojure and Java share the same platform (cf. Figure 1b), developers do not need additional means to invoke Clojure source code from Java and vice versa. However, invoking Clojure functions in Java without additional means works only for compiled Clojure source code. Additionally, the Clojure library provides an FFI (cf. Figure 1a), which allows developers to dynamically load and invoke Clojure source code directly from Java source code.

For dynamically invoking Clojure functions from Java source code, developers must provide the namespace and the name of the function to be called. For instance, to call the function `add2` in namespace `i.o.c.Test` (see Figure 6), developers use the class `RT` shown in Figure 7. Since version 1.6, the preferred way of calling a Clojure function is to use class `Clojure` that returns an instance of class `IFn`. Nevertheless, the basic principle has not changed.

```
1 (ns o.i.c.Test)
2
3 (defn add2 [x]
4   (+ x 2))
```

Figure 6. Definition of a namespace and a function in Clojure.

```
1 Var f = RT.var("o.i.c.Test", "add2");
2
3 f.invoke(2);
```

Figure 7. Invocation of Clojure function in Java.

2) The Clojure-Java-Interface-Checker Library: We used the structure-graph library to implement the clojure-java-interface-checker [23]. The clojure-java-interface-checker is a library that checks the dynamic invocation of Clojure functions in Java source code. To this end, the library creates a graph for the function invocations in the Java source code and for the actual function definitions in the Clojure source code. The graph contains a tree for each namespace defined in the Clojure source code. Each namespace has a child node for each function defined in that namespace. Additionally, each node representing a function has a child node for each function parameter. Having a node for each parameter allows to check that the function invocation in the Java source code contains the correct number of parameters. Figure 8 shows the generalized structure of a tree for a Clojure namespace.

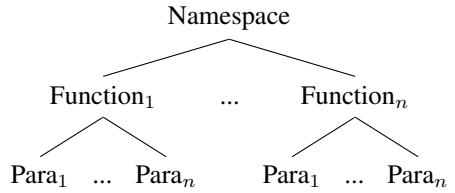


Figure 8. Clojure Function Graph.

For the function definition in Figure 6 and the function invocation in Figure 7, the library creates two different trees (see Figure 9): The only difference between the two trees is that in Java, we have no information about the called parameter but its position. Therefore, in Figure 9b the parameter `x` is represented by the parameter's position `0`. Accordingly, before we can compare these graphs, we need to replace the parameter name in Figure 9a by its position in the function definition.

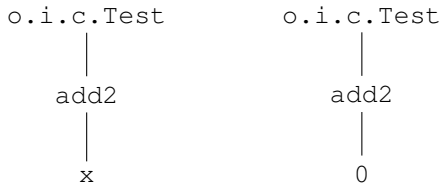
V. Discussion

In Section III, we discussed an algorithm for checking the interaction between a host and guest language. We presented two tools that show the practicability of our approach. In the following, we discuss the theoretical performance with respect to the complexity of checking the Conditions (1) and (2) from Section III-B and the generality of our approach. Furthermore, we describe the necessary implementation effort and other areas of application for our approach.

A. Performance

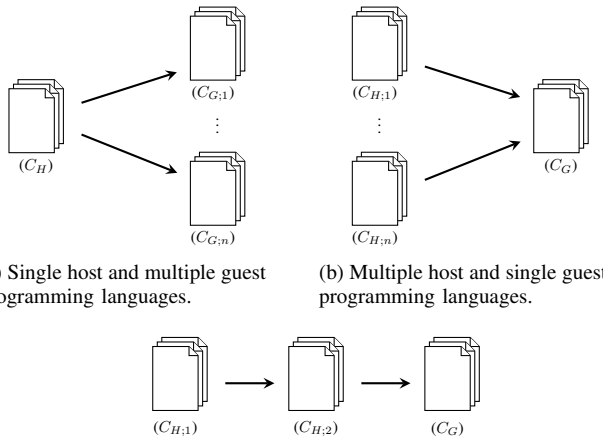
For Condition (1), we check that for each node in $R_{C_H \to C_G}$ a node exists in R_{C_G}. In a tree structure, each node has a unique path, thus, we need to check at most h_G nodes in R_{C_G} for each node in $R_{C_H \to C_G}$ where h_G is the height of R_{C_G}. Hence, we get each missing node in $O(n_H h_G)$ where n_H is the number of nodes in $R_{C_H \to C_G}$.

For Condition (2), we check that for each mandatory node in R_M a node exists in $R_{C_H \to C_G}$. Again, we need to check at most h_H nodes in $R_{C_H \to C_G}$ for each node in R_M where h_H is the height of $R_{C_H \to C_G}$. Hence, we get each missing mandatory node in $O(n_M h_H)$ where n_M is the number of nodes in R_M. Since the maximum height of the trees is constant, we get a complexity of $O(n_H + n_M)$ for checking all conditions.



(a) R_{C_G} of Figure 6. (b) $R_{C_H \to C_G}$ of Figure 7.

Figure 9. Function graphs.



(a) Single host and multiple guest programming languages.

(b) Multiple host and single guest programming languages.

(c) A host is also a guest programming language.

Figure 10. MLSA setups.

B. Generality of the Approach

Until now, we discussed an MLSA consisting of one host and one guest programming language. However, MLSAs can consist of more than two programming languages. Thus, we need to discuss our approach in the context of generalized MLSA setups to justify the generality of our approach.

1) Multiple Guest and Single Host Programming Languages: The single host programming language uses different interfaces to interact with multiple guest languages (cf. Figure 10a). To check the referential integrity of the language interaction, we need to compute and compare $R_{C_{G;n}}$ and $R_{C_H \to C_{G;n}}$ with n being the nth guest language of the MLSA.

2) Single Guest and Multiple Host Programming Languages: The host programming languages use the same elements of the guest programming language for language interaction (cf. Figure 10b). Thus, to check the referential integrity of the language interaction, we need to compute and compare $R_{C_{H;n} \to C_G}$ and R_{C_G} with n being the nth host language of the MLSA.

3) A Host is also a Guest Programming Language: Given code of three programming languages C_G, $C_{H;1}$, $C_{H;2}$ involved in an MLSA where C_G is accessed by $C_{H;2}$ and $C_{H;2}$ is accessed by $C_{H;1}$ (cf. Figure 10c). Hence, this MLSA contains code of two guest programming languages (C_G and $C_{H;2}$) and code of two host programming languages ($C_{H;1}$ and $C_{H;2}$). In other words, we have multiple guest and multiple host programming languages. However, having multiple guest and multiple host programming languages corresponds to a combination of the preceding cases.

Apart from the three MLSA setups shown in Figure 10a to 10c, to the best of our knowledge, no other generalized MLSA setup exists. Since our approach supports all three MLSA setups, we conclude that our approach supports the refactoring of arbitrary MLSAs.

C. Implementation Effort

Our approach requires developers to provide language-specific components: parsers and the evaluation logic. Parsers extract the source code of the interacting host and guest languages and create the graphs for $R_{C_{G;n}}$ and $R_{C_H \to C_{G;n}}$. For the creation of graphs, SSC includes the Java graph library JGraphT [24]. The evaluation logic interprets the results of SSC and gives language-specific feedback to the user.

D. Fields of Application

Apart from refactoring, another use case for our approach is content assistance. That is, we can use R_{C_G} to provide developers with a list of elements with which the developers can interact with. However, especially in respect to legacy or undocumented source code, we have to note that our approach does not consider the guest language's semantics. Thus, developers still need to know the behavior of C_G.

VI. Related Work

TexMo [25] and XLL [7][26] are tools for linking and refactoring MLSAs. TexMo uses GenDeMoG [27] which implements a dependency graph and XLL implements a linking model to link artifacts of interacting languages. Links are resolved by dependency patterns in GenDeMoG and binding resolvers in XLL. Thus, in TexMo and XLL, the artifacts involved in language interaction between two languages are hidden in dependency patterns or binding resolvers. However, based on our experience with other language combinations [13], we developed our graph-based approach that makes the structures involved in language interaction transparent. We argue that this transparency is crucial because language interaction is not only affected by renames of interacting elements. Yet, TexMo and XLL solely support rename refactorings.

Language composition comprises approaches to extend a programming language's syntax and semantics [28]. For instance, *SugarJ* [29] and *TSL Wyvern* [30] allow developers to embed different languages as first-class citizens in Java and Wyvern [31], respectively. As a first-class citizen, an embedded language's syntax is validated at compile-time. Compile-time validation allows developers to fix syntax errors in the code of the embedded language before run-time. Our approach complements language composition because, additionally, it allows to check the interaction introduced by embedded languages. For instance, embedding SQL by language composition can only ensure syntactical correctness, but not that SQL statements reference elements that are available in the database schema. Thus, language composition cannot ensure language interaction in general.

UMLDiff detects differences between two UML class models and reports added, removed, renamed, and moved UML elements [32]. UMLDiff is part of the Eclipse plug-in *JDEvAn* [33]. JDEvAn allows to retrieve UML representations from Java source code but is extensible to retrieve UML representations from other languages. JDEvAn and UMLDiff may be used to retrieve information about the changes that led to a broken language interaction and, thus, complement our check of the referential integrity for language interaction.

Orthographic Software Modeling (OSM) introduces views as first-class entities in software development [34]. In OSM, all information about a software application is represented in a single underlying model (SUM). All other models, such as UML or source code, are generated from the SUM. Since all changes to a view must be propagated to the SUM, all views are kept consistent automatically. For instance, in a database application using JPA, changing the relational model results in the adaption of the relational schema as well as the

 ISBN: 978-1-61208-458-9

JPA entities. However, the OSM approach requires a SUM and a developer, called *methodologist*, who implements and maintains the SUM. In general, we cannot presume these requirements to be fulfilled. Furthermore, the methodologist needs to model language interaction in the SUM, which requires intimate knowledge of implementation details.

VII. Conclusion and Future Work

Language interaction in a multi-language software application (MLSA) can be diverse and, thus, complicates the refactoring of involved languages. In our approach, we use graphs of trees to represent syntax elements of different programming languages that are involved in language interaction. Based on these graphs, we can check if the source code of one language correctly interacts with the source code of another language.

Based on our approach, we presented two tools, sql-schema-comparer and clojure-java-interface-checker, which check the interaction within a database application and between source code of the programming languages Java and Clojure, respectively. We presented performance considerations which suggest that graphs of trees are a viable basis for checking the interaction of source code of different languages. We also discussed transferability of our approach to arbitrary MLSA setups.

In our future work, we want to integrate the sql-schema-comparer and clojure-java-interface-verifier in the Eclipse IDE [35] to simplify their usage for daily software development. Furthermore, we want to re-use graphs of trees extracted from the guest language for detecting source-code modifications, such as refactorings, that led to a broken language interaction. We assume that the information about source-code modifications can support developers in fixing language interaction.

Acknowledgments

The work of Reimar Schröter is funded by BMBF, grant number 01IS14017B.

References

[1] B. Kullbach, A. Winter, P. Dahm, and J. Ebert, "Program Comprehension in Multi-Language Systems," Working Conference on Reverse Engineering, 1998, pp. 135–143.

[2] T. C. Jones, Estimating Software Costs. Hightstown, NJ, USA: McGraw-Hill, Inc., 1998.

[3] M. Grechanik, D. Batory, and D. E. Perry, "Design of Large-Scale Polylingual Systems," International Conference on Software Engineering, 2004, pp. 357–366.

[4] D. Strein, H. Kratz, and W. Lowe, "Cross-Language Program Analysis and Refactoring," IEEE International Workshop on Source Code Analysis and Manipulation, 2006, pp. 207–216.

[5] P. K. Linos, W. Lucas, S. Myers, and E. Maier, "A Metrics Tool for Multi-Language Software," International Conference on Software Engineering and Applications, 2006, pp. 324–329.

[6] N. Chen and R. Johnson, "Toward Refactoring in a Polyglot World: Extending Automated Refactoring Support across Java and XML," Workshop on Refactoring Tools, 2008, pp. 1–4.

[7] P. Mayer and A. Schroeder, "Cross-Language Code Analysis and Refactoring," International Working Conference on Source Code Analysis and Manipulation, 2012, pp. 94–103.

[8] N. Ford, The Productive Programmer. O'Reilly, 2008.

[9] W. F. Opdyke, "Refactoring Object-Oriented Frameworks," Ph.D. dissertation, University of Illinois at Urbana-Champaign, USA, 1992.

[10] M. Fowler, Refactoring: Improving the Design of existing Code. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[11] H. Li and S. Thompson, "Tool Support for Refactoring Functional Programs," ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation, 2008, pp. 199–203.

[12] S. Ambler, Agile Database Techniques: Effective Strategies for the Agile Software Developer. John Wiley & Sons, Inc., 2003.

[13] H. Schink, M. Kuhlemann, G. Saake, and R. Lämmel, "Hurdles in Multi-Language Refactoring of Hibernate Applications," International Conference on Software and Database Technologies, 2011, pp. 129–134.

[14] Clojure homepage. [Online]. Available: http://www.clojure.org/ (visited on Dec. 3, 2015).

[15] M. Furr and J. S. Foster, "Checking Type Safety of Foreign Function Calls," ACM Transactions on Programming Languages and Systems, vol. 30, no. 4, 2008, pp. 1–63.

[16] J. Hamilton, "Language Integration in the Common Language Runtime," ACM SIGPLAN Notices, vol. 38, no. 2, 2003, p. 19.

[17] M. Mernik, J. Heering, and A. M. Sloane, "When and how to Develop Domain-Specific Languages," ACM Computing Surveys, vol. 37, no. 4, 2005, pp. 316–344.

[18] L. Tratt, "Compile-Time Meta-Programming in a Dynamically Typed OO Language," Symposium on Dynamic Languages, 2005, pp. 49–63.

[19] S. Günther, "Multi-DSL Applications with Ruby," IEEE Software, vol. 27, no. 5, 2010, pp. 25–30.

[20] J. Jones, "Abstract Syntax Tree Implementation Idioms," Conference on Pattern Languages of Programs, 2003, pp. 1–10.

[21] Structure graph. [Online]. Available: https://github.com/hschink/structure-graph (visited on Dec. 3, 2015).

[22] H. Schink, "sql-schema-comparer: Support of Multi-Language Refactoring with Relational Databases," International Working Conference on Source Code Analysis and Manipulation, 2013, pp. 164–169.

[23] Clojure java interface checker. [Online]. Available: https://github.com/hschink/clojure-java-interface-checker (visited on Dec. 3, 2015).

[24] Jgrapht. [Online]. Available: http://jgrapht.org/ (visited on Dec. 3, 2015).

[25] R. H. Pfeiffer and A. Wąsowski, "TexMo: A Multi-Language Development Environment," European Conference Modelling Foundations and Applications, 2012, pp. 178–193.

[26] P. Mayer and A. Schroeder, "Automated Multi-Language Artifact Binding and Rename Refactoring between Java and DSLs used by Java Frameworks," European Conference Object-Oriented Programming, 2014, pp. 437–462.

[27] R. H. Pfeiffer and A. Wąsowski, "Taming the Confusion of Languages," European Conference on Modelling Foundations and Applications, 2011, pp. 312–328.

[28] S. Erdweg, P. G. Giarrusso, and T. Rendel, "Language Composition Untangled," Workshop on Language Descriptions, Tools, and Applications, 2012, pp. 1–8.

[29] S. Erdweg, T. Rendel, C. Kästner, and K. Ostermann, "SugarJ: Library-Based Syntactic Language Extensibility," ACM SIGPLAN Notices, 2011, pp. 391–406.

[30] C. Omar, D. Kurilova, L. Nistor, and B. Chung, "Safely Composable Type-Specific Languages," European Conference on Object-Oriented Programming, 2014, pp. 105–130.

[31] L. Nistor, D. Kurilova, and S. Balzer, "Wyvern: A Simple, Typed, and Pure Object-Oriented Language," Workshop on MechAnisms for SPEcialization, Generalization and inHerItance, 2013, pp. 9–16.

[32] Z. Xing and E. Stroulia, "UMLDiff: An Algorithm for Object-Oriented Design Differencing," IEEE/ACM International Conference on Automated Software Engineering, 2005, pp. 54–65.

[33] Z. Xing and E. Stroulia, "The JDEvAn Tool Suite in Support of Object-Oriented Evolutionary Development," Companion of the International Conference on Software Engineering, 2008, p. 951.

[34] C. Atkinson, D. Stoll, and P. Bostan, "Orthographic Software Modeling: A Practical Approach to View-Based Development," in Evaluation of Novel Approaches to Software Engineering, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2010, vol. 69, pp. 206–219.

[35] Eclipse. [Online]. Available: http://www.eclipse.org/ (visited on Dec. 3, 2015).

 ISBN: 978-1-61208-458-9

Collecting Product Usage Data Using a Transparent Logging Component

Thorvaldur Gautsson, Jacob Larsson

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
e-mail: {gautsson, jacobla}@student.chalmers.se

Miroslaw Staron

Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
e-mail: miroslaw.staron@gu.se

***Abstract*—Continuous software engineering and experiments on released products have become very popular in modern software development. In this paper, we present a software component used to transparently log usage data from products in order to facilitate the use of customer-usage data by software developers. Such a component can aid with software maintenance and life-cycle management, but also provide help in software production and validation. We present a technical solution, the evaluation of its influence on the performance of a sample product, and an initial study on the acceptance of such a technology by regular users. Our results show that the mechanisms available in modern programming languages make it possible to integrate such a component without manual interventions in product code and that users are generally positive towards using this technology for logging the usage of work-related applications. We conclude that this type of technology can provide new possibilities for developers to adjust product planning based on the customer usage data.**

Keywords–Logging; usage patterns; features; data analysis.

I. INTRODUCTION

Modern software development emphasizes the need for rapid delivery of customer value and many software development companies use the principles of continuous software engineering to deliver on these needs [1][2][3]. The concept of continuous software engineering drives the development of technology towards continuous integration, continuous deployment and continuous feedback from customers. Continuous integration allows companies to decrease the internal feedback cycles on the quality of software by advocating quick delivery of small software increments and testing them directly after integration with the rest of the software product code. The continuous deployment philosophy prescribes methods and tools for delivering software without the need for manual installation (e.g., changing a web application) and finally the continuous feedback from customers is often realized as customer-experiments (also known as A/B testing [4]).

In this paper, we contribute to the area of rapid feedback from customers by developing a logging component which can be integrated with software products (e.g., desktop applications) without the need to modify the product code. The logging component collects data about the usage of features and functions — both per user and per feature. It also provides the possibility to store strategically taken screenshots of the GUI (*graphical user interface*) of an application, and enables analysing the status of applications before exceptions or crashes occur. Collecting this kind of data has the potential to speed up the development feedback substantially compared to the currently used data (c.f. [5]). The research question which we analyzed in our research was:

How can an external logging component be used to aid in the process of software development by providing developers with information about usage patterns?

We set off to design a component which could be integrated with existing products without manual intrusion in the product source code, though recompilation was allowed. We also defined the term *usage pattern* as *how users use an application at a high level*, i.e., how they interact with the application through mouse clicks and keyboard input, which pre-defined features of the application they use, when and how often they use those features, and when and how they cause exceptions to occur. The logging component scrutinized in the research therefore provides developers with the flexibility to define the precise usage pattern that constitutes a feature.

Although there exists a body of research on both the logging of software applications as well as managing large quantities of data (c.f. [6][7]), there is still much to be researched. Our working hypothesis was that analyzing feature usage with the help of screenshots and unconstrained logging of method-calls can be highly valuable for developers. Such an approach has been found to help companies to remain innovative in the long run [8].

Our results show that automated logging of method-calls combined with pre-defined packaging of the sequences of method-calls into features can help developers to understand the usage patterns of an application. Based on a survey which was conducted, we also found that the logging of feature usage is generally met with a positive attitude for work-related applications, but skepticism for privately used software. The study led us to conclude that the largest developmental benefits are the combination of logging and pre-definition of features — which is a rather unique approach.

This paper is structured as follows: Section II presents some of the most related work in the field of customer data collection and continuous software engineering. Section III presents the design of our research. Section IV presents the logging component and its design. Section V presents the results from the evaluation of the logging component. Finally, Section VI presents the conclusions from our study.

II. RELATED WORK

Backlund et al [9] studied post-deployment data collection by conducting a case study on a web-based portal system.

 ISBN: 978-1-61208-458-9

They began by identifying which quantitative data needed to be collected, collected it and finally compared the collected data with survey answers from test subjects. The quantitative data used was collected through aspect-oriented programming and included various user actions, such as button clicks and task completion times. The authors found a correlation between the survey data and the measurements. For instance, both the survey and the measurements suggested that a task called *change password* was the most difficult task to perform.

Olsson et al. [5][10] studied patterns in post-deployment data collection in products from three companies in the embedded software development field. Their results show that the collected post-deployment data that the companies used came from the operating system, or concerned performance. They found that while feature-usage data is valuable, it is generally not collected. In our work, we address this challenge in the context of applications written in C#.

Lindgren et al. [4] studied the implications of using experiment systems in the development of software. Through a survey among companies, they found that there is still no consensus on how to collect customer feedback and how to use it during development. They also found that customer feedback is rarely used during development. Our work contributes to the ability to collect data automatically, thus presenting a way to use customer data during development.

Börjesson and Feldt evaluated in 2012 two tools for automated visual GUI testing on a software system developed by a Swedish aerospace company. They found that visual GUI testing can perform better than manual testing practices and that it furthermore has benefits over manual GUI testing techniques. They stated however that visual GUI testing still had challenges which had not been addressed [11][12].

III. RESEARCH DESIGN

The research presented in this paper utilized two research methods: design science research for the development of the logging component and a case study for its evaluation.

A. Design science research

The design science research approach [13][14] was used to construct a logging component which serves as a proof of concept for detecting usage patterns in external applications. After the logging component had been constructed it was integrated with a prototype application and then assessed. Further evaluation was conducted in a case study in which both qualitative and quantitative aspects were considered.

In order to ensure industrial applicability, the logging component was developed in cooperation with Diadrom Systems AB (hereafter: Diadrom) in Gothenburg. Representatives from Diadrom provided continuous feedback, both in formal as well as informal settings. Employees from the company were also part of the evaluation processes.

Before the development phase began, a prototype software application called *PersonDatabase* was built in order to have an application which the logging component could be built around. This application had the purpose of storing information about employees of a fictitious company.

B. Case study

After the design science research phase was over, the logging component was further evaluated through a case study [15]. Two applications which had previously been developed by Diadrom were used in this phase. The assessment involved both qualitative and quantitative aspects, using both metrics which were measured as well as structured group interviews. The case study process model which was used to evaluate the logging component consisted of the following steps:

1) Two suitable applications for evaluation were identified: an application from Diadrom (hereafter: Application X), consisting of ca. 40 000 LOC; and an open source application named *ScreenToGif* which allows users to record an area of their screen, manipulate, edit, and then save as a gif image file [16].
2) Qualitative data was collected through workshops and quantitative data by using the aforementioned applications.
3) The collected data was analyzed.

1) Interviews: In order to obtain qualitative data, two workshops were held where semi-structured interviews were conducted. Application X was evaluated through a workshop held at Diadrom which was attended by developers at the company. ScreenToGif was evaluated through a workshop held at Chalmers University of Technology. Both workshops followed the same structure. First, the project was introduced and the research question presented. Next, the integration of the logging component and the target application was shown. The attendees were then asked a series of open-ended questions relating to how difficult they perceived the integration process to be. Thereafter, a live demonstration was given to show how the logging component worked on the application which it had been integrated with. Following that, the logging component was discussed and questions about its benefits were posed. The remaining part of the workshop was then used to present open-ended questions.

2) Measurements: Several criteria were defined which the logging component had to fulfill, along with ways to measure them. The following aspects were measured:

- Time to execute a sequence of operations with or without the logging component
- CPU usage with the logging component
- The size of the database after a sequence of operations
- The size of an average screenshot

In order to obtain data which could be generalized, measurements were taken for three different applications. The applications tested were the applications used in the workshops, i.e., ScreenToGif and Application X, as well as the PersonDatabase application previously mentioned. The measurements were conducted using the Visual Studio Profiler, the `db.stats()` function in MongoDB and SikuliX. The Visual Studio Profiler is a tool for analyzing performance issues in an application and gathering performance data. The MongoDB function returns statistics about a particular database. SikuliX is a visual GUI testing tool.

The first aspect was measured on the target application, both with and without the logging component. To measure the time to execute a sequence of operations in the application,

 ISBN: 978-1-61208-458-9

SikuliX was used. The tool was used to define a sequence of operations which was then executed 100 times using a loop. The time of each execution was then measured from when the pre-defined sequence started until it stopped. By doing this both with and without the logging component it was possible to investigate whether the it slowed down the application significantly.

To measure CPU usage the Visual Studio Profiler was used. Due to restricted access and technical limitations it was not possible to measure CPU usage for Application X, as permission was not granted to install the application on computers which had the measuring tools needed. The size of an average screenshot as well as the size of the database were measured using the `db.stats()` function in MongoDB.

C. Analysis methods

The Student's t-test was used to see whether there was a statistically significant difference between the time it took for the PersonDatabase and ScreenToGif applications to execute a sequence of operations with and without the logging component. Since the variance of the data sets for Application X varied greatly, the Welch t-test was used in that case, as it performs better for data sets of unequal variance.

The null hypothesis was that there should not be a significant time difference in executing the sequence of operations dependent on whether the logging component was integrated with the application or not.

IV. The Logging Component

The purpose of the logging component was to log various user actions and program behavior and store those logs in a remote database. Among the user actions logged were keyboard input and mouse clicks. Handled and unhandled exceptions, as well as method-calls, were also logged. Screenshots were taken when a user interacts with the application — to make it possible to understand how the application behaved from the users' point of view.

The way the logging component operated was by weaving log statements into the source code of an application at compile time. Weaving is a technique for automatically injecting code into previously written code and is further explained in Section IV-A.

Immediately after the logging component started to generate data, the need for a GUI to view the data emerged. It became necessary to develop a GUI both for verifying that the correct data was being logged, and to be able to view how the logging component worked. The development therefore resulted in three different modules:

- a logging module that logged usage patterns
- a GUI module for presenting the data
- a weaving module for the code injections

Together these three modules constituted a system which defines logging and presentation of data for C# WPF (Windows Presentation Foundation) applications.

A. Weaving Module

The Weaving module only had one purpose: to inject code into a target application in order to connect it with the logging module. To achieve this, an open source weaving tool called Fody was used. By using Fody it was possible to define how and where code should be injected into the application without specific knowledge about the Microsoft Build Engine and the Visual Studio APIs. Since Fody had been released as a NuGet package it was possible to create a NuGet package out of the Weaving module that would automatically install Fody.

B. Logging Module

The logging module was developed using C# WPF. To store the large amount of data that the logging module produced, a NoSQL database called MongoDB was used due to its flexibility. The logging module provided an interface for logging information and timestamps for the following:

- Method-calls
- Handled and unhandled exceptions
- Mouse clicks and mouse scroll (start and stop)
- Keyboard button clicks
- Specified Keyboard Shortcuts

The logging module logged method-calls in order to track the data flow of an application. The method logs contain data concerning the namespace, class name, method name, parameter types, parameter names and the time of execution. This can allow developers to find any given method in their source code, and the associated timestamp makes it possible to view the sequential order of execution.

Logging exceptions is also important and an interface was provided for logging both handled and unhandled exceptions. To log handled exceptions, it was necessary to insert a log statement into every "catch"-block in an application, which is automatically done by the weaving module. To log unhandled exceptions, an event handler in C# WPF was used.

To capture user interaction, screenshots were taken for mouse clicks, mouse scroll, button clicks and specified keyboard shortcuts. All screenshots contained a timestamp to make it possible to follow the interaction between user and computer in a sequential order. All saved logs, except for the exception logs, were accompanied by a screenshot. The purpose of the screenshots was to allow developers to view what actions a user had taken; it made it for instance possible to view what a user did before an exception occurred or how a user used a certain feature.

C. GUI Module

The GUI module was developed in C# WPF. The GUI module displays data for a selected individual user or aggregated data for all users. Special sub-menus for *statistics* and *features* sub-menus were available for both one selected user as well as in the form of aggregated data from all users. A figure of the statistics view for one users is displayed in Figure 1. The statistics sub-menu contains information about the following:

- Most common exceptions
- Most used features
- Most called methods
- At what time of day the application is used

 ISBN: 978-1-61208-458-9

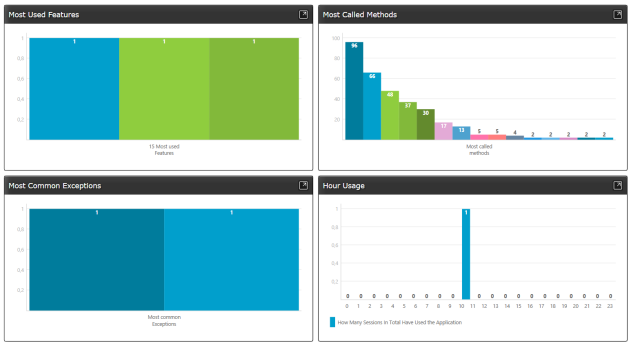


Figure 1. Statistics sub-menu showing: most called methods, when an application is used, most common exceptions and most used features

General information presented about an application included the following: (i) total number of users, (ii) average number of sessions per user, (iii) average sessions which crashed the application per user, (iv) total number of sessions which crashed the application, (v) total number of sessions, (vi) average time for a session, (vii) average number of different flows used per session, and (viii) average number of features used per session.

Defining features as sequences of method-calls

In order to know how the users of an application use its features, those features need to be defined in some way. This was achieved through the GUI module, which provided functionality for defining and mapping what method calls a feature consists of.

To give a concrete example of feature definition, we can consider the previously mentioned PersonDatabase application. This application enabled its users to add a person to a database by pressing an *add* button, then entering the first and last names in respective input boxes, and finally clicking on a *save* button. In this case, it seems reasonable that the *add* feature finishes when the *save* button has been pressed. The above scenario will trigger methods to be called in the application, and all method-calls are logged by default. The *add* feature can therefore be defined as a sequence of method calls that begins when the user clicks on the *add* button and ends when the user clicks on the *save* button. The feature definition is therefore kept completely detached from the source code of the application that is being logged. This also makes it possible to re-define features at will, without affecting the underlying data.

In addition to a sequence of method-calls being mapped to a feature, there can be several *flows* leading to the same feature being used. For example, a person could be added to *PersonDatabase* by using an *add* button or perhaps by using a keyboard shortcut. Each flow is defined by one or several events, where an event is a method-call that has been defined by developers as *important* using the GUI module. An example of an event would be the first method-call triggered when clicking the *add* button. When executing the feature calculation, all method-calls are mapped to the defined events. In case there is a defined event for a method-call, the method-call will be marked as an event. After all the events have been found, they are iteratively mapped towards flows. If the events appear in a sequence, as defined by a flow, the flow is marked as having been used once — together with the feature that is represented by the flow. In this way, it is possible to calculate statistics about feature usage. How the mapping process functions is illustrated in Figure 2.

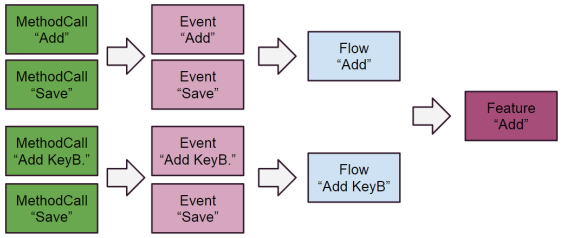


Figure 2. The different steps in the mapping of a feature.

After the features have been mapped using the method in Figure 2, various statistics can then be viewed in the GUI module. The feature usage view for all users is displayed in Figure 3.

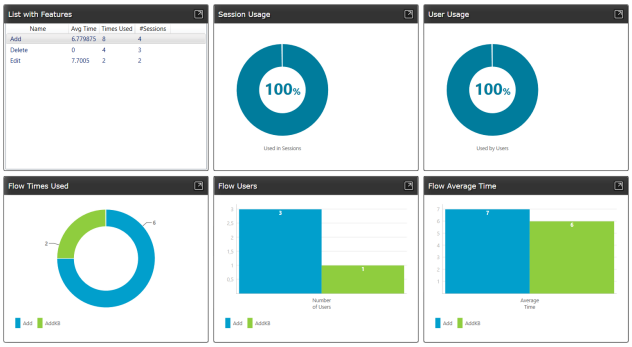


Figure 3. Feature statistics. The bottom row shows data on feature flows

For every feature, it is possible to view statistics for the average execution time, number of times it was used, how many session have used it, and the percentage of users that have used it. Information about the flows for the feature is available and provides statistics for how many times each flow has been used, how many users have used the flow and what the average time for each flow is. A similar view is shown if a single user is selected.

V. Evaluation Results

After development had ceased and all data had been gathered, the results were evaluated and conclusions drawn.

A. Measurements

The average time it took to run a pre-defined sequence of steps in ScreenToGif, PersonDatabase and Application X with and without the logging component is summarized in Table I. The sequence of steps was automatically executed 100 times and the results then averaged. The data gathered was then tested by using a t-test with $\alpha = 0.05$ to see if there was any significant difference in execution time with and without the logging component. For PersonDatabase and ScreenToGif there was a significant difference, but not for Application X. For PersonDatabase and ScreenToGif the difference observed is therefore likely caused by the integration of the logging component. The data collected for Application X had a much

 ISBN: 978-1-61208-458-9

higher variance than the data collected for the other two applications, which is the reason to why no significant difference could be confirmed.

The performance measurements therefore indicate that the logging component has some negative effect on the performance of an application. How large the effect is depends on the nature of the logged application. For instance, an application which has a method that is called thousands of times during a short interval will likely create latency issues and might even cause the logging component to run out of memory. In those cases it would be necessary to disable logging for that particular method. What the performance measurements appear to show is that in most cases the time difference is within an acceptable range, as the observed time difference in all three cases would be quite hard for a regular user to notice.

TABLE I. AVERAGE TIME TO EXECUTE A SEQUENCE OF STEPS

	PersonDatabase	ScreenToGif	Application X
With the logging component	43.88 s	18.23 s	56.36 s
Without the logging component	42.64 s	17.44 s	55.87 s
Difference	1.24 s	0.79 s	0.49 s

Measurements of CPU usage were gathered using a CPU measurement tool in Visual Studio by running PersonDatabase and ScreenToGif with the logging component. The results are presented in Table II. The CPU usage of the logging component was divided into three areas: method-call, screen events and buffer & DB (database). The logging component constituted 25.93% of the CPU usage of PersonDatabase and 11.91% of the CPU usage of ScreenToGif. The reason for the large difference is that ScreenToGif requires more CPU computation in general just to run the application. PersonDatabase is a small application with a relative low CPU usage.

TABLE II. HOW MUCH OF THE TOTAL CPU POWER OF AN APPLICATION THE LOGGING COMPONENT USES

	Method-call	Screen Events	Buffer & DB	Total
PersonDatabase	0.55 %	14.01 %	11.37 %	25.93 %
ScreenToGif	0.2 %	4.45 %	7.26 %	11.91 %

To measure the size of the data generated by the logging component, a built in measurement tool in MongoDB was used. The size of an individual image and of a log statement for a method-call was calculated from the data. The results are presented in Tables III and IV. *Count* defines how many logs the database contained and *average object size* is calculated by dividing the *total size* with *count*.

TABLE III. DATABASE SIZE FOR STORING IMAGES

Approx. Image Size	Count	Total Size	Avg. Object Size
Small (300x350)	560	7200 kB	12.86 kB
Medium (880x600)	200	12907 kB	64.5 kB
Large (1550x840)	206	26416 kB	128.23 kB

TABLE IV. DATABASE SIZE FOR STORING METHOD-CALLS

Application	Count	Total Size	Avg. Object Size
PersonDatabase	17452	8656 kB	0.496 kB
ScreenToGif	6475	3217 kB	0,496 kB

B. Interviews

In order to evaluate whether an external logging component could be used to aid software development by providing developers with information about usage patterns, two workshops were held to obtain qualitative data. The participants in the first workshop were four students in the Software Engineering M.Sc. programme at Chalmers University of Technology. This workshop was used as a pilot study before conducting the workshop at our industrial partner. All of the subjects considered software development to be their area of work and they all had previous industrial experience which ranged from 2 to 7 years. The main findings from the workshop were:

1) The integration process using weaving was perceived as straightforward as it required only a few mouse clicks in the Visual Studio GUI.
2) A set of situations when the logging component should be modified — e.g., a case of a function which crashed when being logged (because of buffers) — were noted.
3) The logging component was not found to hinder application performance. Furthermore, a suggestion was raised that logging component could be used to re-architecture an application to its improve performance.
4) It was considered essential to conduct a survey to find how users would perceive being logged.

The second workshop was held at Diadrom and the participants were four developers with years of experience in developing software applications. All participants had previous experience with Visual Studio and all had used logging tools of some kind at some point. The target application which the logging component was integrated with in the workshop was built for a Swedish aerospace company. The results were:

1) The practitioners perceived the integration process to be straightforward.
2) The practitioners suggested further uses — for instance customization of which design-time elements should be logged (e.g., namespaces).
3) The practitioners identified further development areas — e.g., a management view, adding temporal aspects or presentation of user clicks as a heat-map.

Finally, the participants concluded that they would not mind using an application which was logged by the logging component — as long as the data was not used to try to measure the productivity or performance of an employee. As long as the logging component was used for debugging or developmental purposes, they found logging to be acceptable.

C. Survey

It was considered essential to conduct an initial survey to find how users of an application would perceive being logged. This was evaluated by presenting the logging component to employees at four different companies in Sweden, and afterwards handing out a survey. The total number of participants in the survey was 27, of which 16 were software developers, 8 worked in management, and 3 worked in other fields. No participant had less than 2 years of work experience, and nearly 50% had 10 or more years of work experience.

The participants were also asked about the developmental perspective. Over 90% of the participants said that they knew

 ISBN: 978-1-61208-458-9

of a project where the logging component would have been useful, and 100% of the participants thought that the logging component had potential to provide information that could aid in the further development of an application. Just under 90% of the participants thought that the logging component had potential to provide information that would facilitate the debugging of an application.

Three questions were asked to query how users would perceive being logged. The participants were first asked whether they would be comfortable using an application for their own private matters if they knew that it was being logged. As Figure 4 shows, 60% of the respondents said that they would not be comfortable with using a logged application for private matters. The participants were then asked whether they would be comfortable using an application at work if they knew that it was being logged. In this case, the results were very different. As Figure 5 shows, only 3 out of 27 participants – around 10% – answered that they would not be comfortable with this.

Q: I would be comfortable if an application that I use for private matters is being logged by the logging component

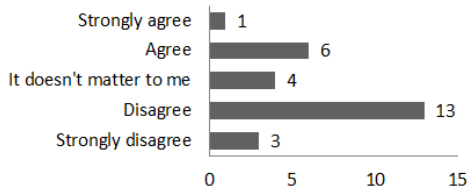


Figure 4. Comfort with using an logged application for private matters.

Q: I would be comfortable if an application that I use at work is being logged by the logging component

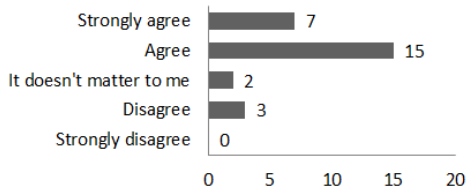


Figure 5. Comfort with using a logged application at work.

Finally, the participants were asked whether they would be more comfortable using an application that is being logged at work rather than one they use for private matters. 20 out of 27 answered that they would be more comfortable using a logged application at work than at home, while 7 participants answered that they felt that was no difference between the two. These results therefore suggest that the context in which the logging component is used can greatly affect the perception of users about whether logging is acceptable or not. Using a logged application at work, for instance, seems to be much more tolerable for most people rather than using a logged application at home.

VI. Conclusions

Continuous deployment, experiment systems and user-centered software engineering approaches have become very popular in modern software development. However, there are still challenges, such as how to add logging functionality to an application, what to log, and how to translate low-level logging data into knowledge about how product features are used by their users. In this paper, we contributed by developing and evaluating a logging component which could be integrated with a product without affecting its source code, and which added negligible performance penalties. It was found to provide developers with informative data on how a product is used by enabling them to define features and then visualize how they are used. Since this style of logging can lead to ethical issues, we conducted a survey with 27 participants at four different companies to initially assess the attitude of users to being logged. The results from the survey showed that users consider the logging process to be acceptable as long as they are informed of it, and if the logged application is not one that they use for private matters.

References

[1] M. Staron, W. Meding, and K. Palm, “Release Readiness Indicator for Mature Agile and Lean Software Development Projects,” in Agile Processes in Software Engineering and Extreme Programming. Springer, 2012, pp. 93–107.

[2] B. Fitzgerald and K.-J. Stol, “Continuous software engineering and beyond: trends and challenges,” in Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. ACM, 2014, pp. 1–9.

[3] J. Bosch, Continuous Software Engineering. Springer, 2014.

[4] E. Lindgren and J. Münch, “Software development as an experiment system: A qualitative survey on the state of the practice,” in Agile Processes, in Software Engineering, and Extreme Programming. Springer, 2015, pp. 117–128.

[5] H. H. Olsson and J. Bosch, “Post-deployment data collection in software-intensive embedded products,” in Continuous Software Engineering. Springer, 2014, pp. 143–154.

[6] R. Veeraraghavan, G. Singh, K. Toyama, and D. Menon, “Kiosk usage measurement using a software logging tool,” in Information and Communication Technologies and Development, 2006. ICTD’06. International Conference on. IEEE, 2006, pp. 317–324.

[7] T. Menzies and T. Zimmermann, “Goldfish bowl panel: software development analytics,” in Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 2012, pp. 1032–1033.

[8] A. Steiber and S. Alänge, “A corporate system for continuous innovation: the case of google inc.” European Journal of Innovation Management, vol. 16, no. 2, 2013, pp. 243–264.

[9] E. Backlund, M. Bolle, M. Tichy, H. H. Olsson, and J. Bosch, “Automated User Interaction Analysis for Workflow-based Web Portals,” in Software Business. Towards Continuous Value Delivery. Springer, 2014, pp. 148–162.

[10] H. H. Olsson and J. Bosch, “The hypex model: From opinions to data-driven software development,” in Continuous Software Engineering. Springer, 2014, pp. 155–164.

[11] E. Borjesson and R. Feldt, “Automated System Testing Using Visual GUI Testing Tools: A Comparative Study in Industry,” in Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on. IEEE, 2012, pp. 350–359.

[12] R. Feldt, M. Staron, E. Hult, and T. Liljegren, “Supporting software decision meetings: Heatmaps for visualising test and code measurements,” in Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on. IEEE, 2013, pp. 62–69.

[13] V. Vaishnavi and W. Kuechler, “Design Research in Information Systems,” 2004.

[14] R. H. von Alan, S. T. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” MIS quarterly, vol. 28, no. 1, 2004, pp. 75–105.

[15] P. Runeson and M. Höst, “Guidelines for Conducting and Reporting Case Study Research in Software Engineering,” Empirical software engineering, vol. 14, no. 2, 2009, pp. 131–164.

[16] Nicke Manarin. ScreenToGif. [Online]. Available: https://screentogif.codeplex.com (Retrieved: January 2016)

 ISBN: 978-1-61208-458-9

The ICT Measurement System

Definition, components and a maturity evaluation approach.

Roberto Meli
CEO
Data Processing Organization Srl
Rome, Italy
email: roberto.meli@dpo.it

Abstract— **The measurement of ICT (Information & Communication Technologies) processes and related products / services is often experienced, by the organizations which make them for themselves or for the market, as a too expensive activity that slows down the primary production activities. Nevertheless, the "measurement of ICT" can give a strong support to its governance. Rarely measurement is perceived as an opportunity, most often as a threat. Many measurement initiatives, in the past, were not successful in becoming stable frameworks for supporting software processes. A formalized ICT-Measurement System (ICT-MS) can facilitate to position the "seemingly expensive" measurement activities in the context of corporate governance which, at least, makes them more justified, productive and interesting for the management and for the other involved stakeholders. An ICT-MS is a governance tool for both the contractual relations between customers and suppliers and the internal production processes. This paper outlines the rationale for the construction of an ICT-MS; it focuses on the differences between a Measurement System, a Measurement Program and a Measurement Plan; it analyzes the ICT-MS into its constituent parts; it provides useful indications for a custom path of construction of the ICT-MS. At the end, a simple maturity model is sketched to allow assessing the maturity and adequacy of an ICT-MS as a function of specific context variables.**

Keywords- ICT Measurement System; Metrics Program; maturity model.

I. INTRODUCTION

The measurement of ICT processes and of their related products / services is often perceived, by people in charge of production, as an unnecessary activity disturbing the primary workflow of system deployment. If it is practiced, it is often done to fulfill contractual obligations, sometimes to formally answer to necessary requisites for a certification of quality or of a maturity level (i.e., CMM-I model). Measurement is rarely perceived as a management opportunity, more often as a threat. It is not easy to win this prejudice if measurement remains a facultative and naive task left to the personal initiative of individual analysts or project managers.

A formalized ICT Measurement System allows to position this "apparently useless" measurement activity in a context of business governance that, if nothing else, makes it explicit, integrated into the production processes, cost effective and valuable for the management and for the other involved stakeholders. Measurement may be done at different levels in the ICT organization: at a project level it gives information about the progress and the state of a specific project; at the portfolio level it gives information about the global usage of organizational resources and the progress and the state of the totality of interacting projects; at the process level, it gives information about the general behavior of projects and the statistically derived trends over time, at the Organizational Unit it gives information about the efficiency and effectiveness of the structure.

It is essential, in our opinion, to recognize that "measurement of ICT" can give a strong support to its governance and that, in most organizations, any activity which is not continuously fed with resources and managerial attention is destined to decline or to rapidly reach the cemetery of good ideas and intentions. This is why it is preferable to speak about a "Measurement System" more than a "Measurement Program". As defined in [1], a Program is "a group of related projects, subprograms and programs activities managed in a coordinated way to obtain benefits not available from managing them individually." A Program, by definition, has a start and an end point for activities; it is not a permanent endeavor. On the contrary, measurement should be conceived as a permanent service for the organization so it should be associated with a stable structure. The present paper is composed of seven sections plus references: Section 1 is the introduction; Section 2 is the definition and qualification of an ICT Measurement System; Section 3 is devoted to the description of the main available resources and standards; Section 4 describes the several components of an ICT-MS; Section 5 is centered on the ICT-MS life cycle from start-up to on-going operations; Section 6 presents a draft of a capability model for ICT-MS; Section 7 contains a short Case Study description; Section 8 presents the conclusions.

II. ICT MEASUREMENT SYSTEM DEFINITION

Very often, terms like Metrics Program and Measurement Plan are used by the software engineering community [1]-[8], but what are the differences among a Metrics Program, a Measurement Plan and an ICT Measurement System?

A Metrics Program is an initiative to promote the usage of a measurement process in the organization: it is a project oriented effort; in other words, it is temporary. A Measurement Program has a specific and obtainable set of goals to be achieved within a limited and predefined budget and time. Resources are not assigned to a specific

 ISBN: 978-1-61208-458-9

permanent organizational unit on a repetitive and regular base but dynamically allocated to the Program according to a specific plan. A Metrics Program usually deploys deliverables and not continuous services.

A Measurement Plan is a project related document describing all that is needed to implement the measurement process within a specific ICT project. It lives with the associated project and does not survive to it.

Fredreiksen H.D. and Mathiassen L. in [7] stated that "To be successful, a metrics program should be planned in context with an organization's processes, structures, climate, and power". This leads toward a more extensive approach to the subject, an approach which is not limited by the projects constraints and capable to survive to its start up phase.

An ICT Measurement System is something more than a process: it may be defined as a *living operational sociotechnical system deputed to the management of the measurement aspects of the ICT processes and products / services* [9]. It represents a governance tool both of the contractual relationships between clients and suppliers and of the internal production processes. An ICT-MS contributes to the enhancement of the processes involved in the management, design, implementation and delivery of projects and ICT services. Its purposes are mostly devoted to support process improvement, the governance of the incoming / outgoing supplies, benchmarking, calculating and monitoring of productivity, cost, time and quality.

The adoption of a metrics system involves the definition, collection and use of a set of metrics and rules of use related to the various organizational levels which will be composed in such a way as to favor the understanding of the business performance. The ICT-MS is the best tool to develop the culture and practice of measurement of ICT business in organizations and include them in their 'genetic code'.

An ICT Measurement System (Figure 1) is composed of various elements: Processes, Methods, Techniques, Tools, Infrastructures, Information Systems, Organizational Structures, Competencies, Motivations. An ICT Measurement System is the only permanent entity capable to supply adequate measurement services to the staff that needs them.

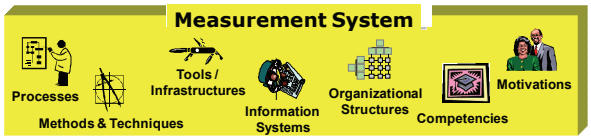


Figure 1. ICT Measurement System components

Measuring ICT systems and related services thus requires dedicated resources, expertise, time, equipment and organization. This activity is often seen as a too expensive task that slows down the primary development activities. Just because of the actual difficult market conditions, unfortunately, public and private organizations are less motivated to stake resources to ensure the performance of the measurements, without having a confidence about the extent of the economic return associated with such resources. A paradoxical situation, then, occurs: not investing resources in the culture and practices of measuring generate wastes in the management of projects that affect willingness and capability to invest in areas that are not clear in terms of returns, thus creating a vicious cycle that is difficult to break. A formalized ICT-MS can facilitate to position the "seemingly expensive" measurement activities in the context of corporate governance which, at least, makes them more justified, productive and interesting for the management and for the other involved stakeholders.

III. Relevant Resources and Standards for an ICT Measurement System

Several detailed resources are available to guide the implementation of a measurement process. Unfortunately, they do not cover adequately the constitution of the ICT-MS as a company's permanent system but are, anyway, precious sources of information for some of the major components of that system. Specifically, there are some de facto and de jure standards helping the identification of: stakeholders, information requirements, activities, tools, deliverables of a measurement process. Figure 2 clarifies the relationships among the main standards and models.

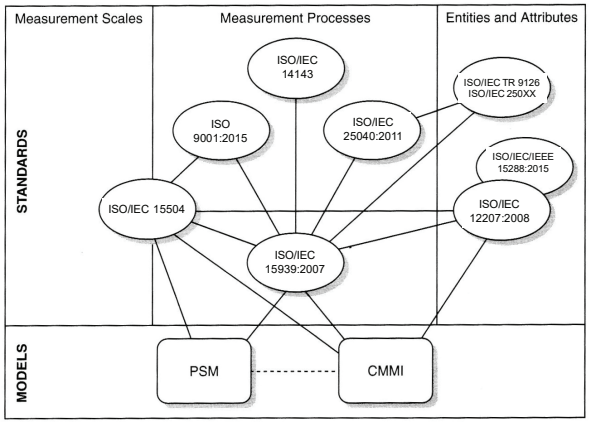


Figure 2. Available Standards & Models for ICT measurement

The ISO/IEC 15939 standard [10] and the Practical Software Measurement framework (PSM) [7], describe a Measurement Process and its related Measurement Information Model (MIM). These models are the reference guides for anyone involved in the field. They outline a project oriented measurement process in terms of information models, activities and results. Other important available resources are: the Capability Maturity Model Integration (CMM-I) [11], the "Guidelines to Software

 ISBN: 978-1-61208-458-9

Measurement" by IFPUG [4] and the Software Engineering Body of Knowledge (SWEBOK) [12].

A. ISO/IEC 15939

Figure 3, extracted from the standard documentation, shows the flow of activities and their relations.

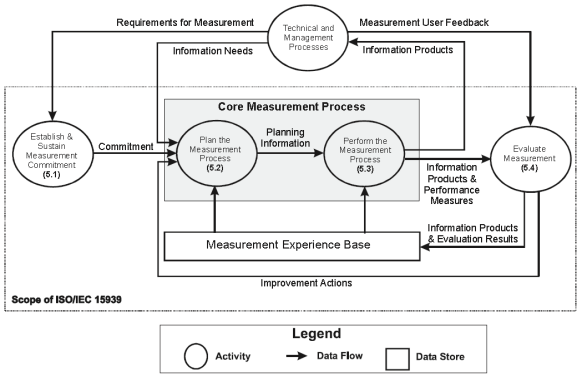


Figure 3. Measurement activities according to ISO/IEC 15939

The ISO approach muddles up activities needed to build up a permanent ICT-MS with activities needed at the temporary project level. Actual practices suggest to keep them separate to be more effective and efficient. The cycle shown in Figure 3 could be covered at various levels: the project level and the portfolio level (both of them being temporary), the functional level and the ICT-MS level (both of them being permanent). Information needs are different at all levels and feedback loops have different timings.

B. Practical Software Measurement (PSM)

In this model, the differentiation between actions at the system level and actions at the project level is clearer although a permanent system is not explicitly identified and suggested. Section 5 of the First Part of the Practical Software and System Measurement guide is devoted to the 'Enterprise and Organizational Context' from which the following Figure 4 has been extracted. PSM approach is still very "project" oriented and the global organizational context is not completely covered at the same detailed level as the project is.

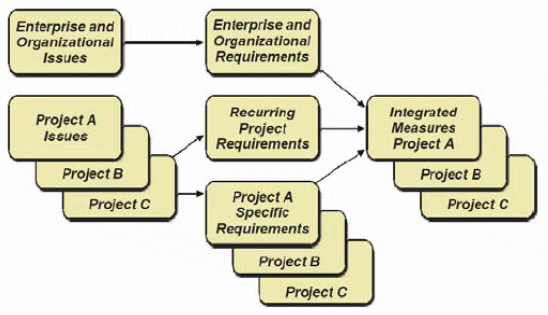


Figure 4. Enterprise & Organizational Context

C. CMM-I.

CMM-I is a set of best practices used to help an organization succeed. Best practices are grouped by activities (Process Areas) like Requirement, Risk, Configuration, Planning, etc. One of the Support Process Areas is "Measurement & Analysis", whose purpose is to develop and sustain a measurement capability that is used to support management information needs. This area is positioned at Maturity Level 2. This means that it is considered as a basic subject that should be metabolized by most organizations. Also this framework is process / project oriented (Figure 5) and it is not centred on permanent structures to perform measurement activities.

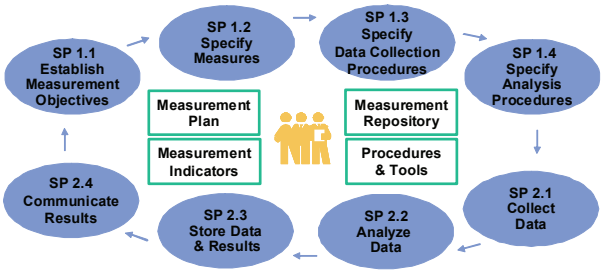


Figure 5. CMM-I – PA "Measurement & Analysis"

D. Guidelines to Software Measurement (IFPUG)

This document has a section named " Implementing and Sustaining a Software Measurement Program", which "provides guidelines for both implementing and sustaining a software measurement program using a project based approach. " The main sections are:

- Introduction
- Stakeholders and Participants
- Features, Benefits and Applicability
- Implementation Methods
- Considerations and Issues
- Sustaining a Software Measurement Program

In this document, too, a project oriented approach is outlined. No information about how to build and run a permanent measurement system.

E. SWEBOK

The Software Engineering Body of Knowledge (SWEBOK) covers the measurement theme in a light way. The measurement issues are embedded into the Engineering Foundations and in other related areas like Software Requirements, Software Engineering Management, Software Quality but a general framework is missing and a "project/process" view is adopted.

 ISBN: 978-1-61208-458-9

F. *Resources Summary*

To summarize, it is possible to state that there is a large amount of useful information and models about how to run a Metrics Program or initiative and about how to measure and analyse specific products and processes, but there is less documented knowledge about the building of permanent ICT-MS. Specifically, the following points are not clearly addressed:

- support a project manager in using measures to manage a project;
- support a service manager in using measures to manage continuous services
- support a line manager to use measures to manage the business units
- create and maintain a permanent structure capable to supply measurement services in an explicit and recognizable way and to evolve itself to follow the business needs.

This paper describes a model/framework that could help organizations in defining their own paths to achieve permanent results in the goal of adopting measurement as a common practice to make ICT governance.

IV. ICT-MS COMPONENTS

In this section, the main components of an ICT-MS (Figure 6) will be highlighted: Services, Processes, Methods & Techniques, Tools, Information System, Organization, Competencies and Motivations.

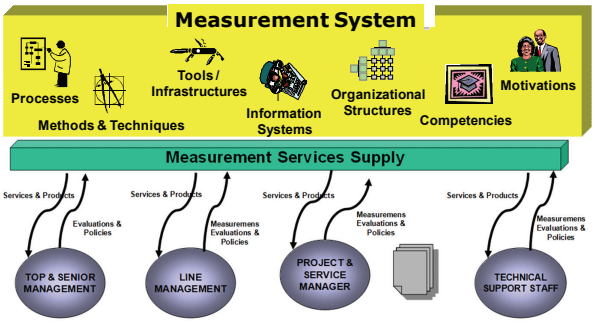


Figure 6. ICT-MS on-going components

A. *Services*

An ICT-MS should supply, as a priority, the following general services:

- identification of indicators and metrics required at various levels of the organization
- choice of measurement standards and definition of internal guidelines
- maintenance of measurement processes and tools
- creation and maintenance of the database of ICT measures
- management of a metrics documentation centre (paper / intranet)
- active promotion and dissemination of standards, methods and tools
- internal promotion and search for consensus on measurement initiatives
- on line helpdesk
- internal / external (suppliers) auditing
- collection, processing and interpretation of the collected data (reports, statistics, studies, analysis, benchmarking)
- modelling productivity and internal / external benchmarking
- reports to the Management and Organizational Units
- overall improvement of the metric system itself (model of productivity calibration, development of procedures, guidelines update, etc.)

and hopefully:

- research / testing / publication in measurement subjects
- external representation
- participation in technical conference events and external initiatives

The main services to the project *and ICT service management* should be the following:

- sizing Products and ICT Services
- supporting effort, time, cost and staff forecasts
- consulting projects or service units on measurements, estimates, surveys of interest on projects and applications
- support in the drafting of documents for (active and passive) procurement and contracts
- support in the negotiation and management of disputes

The main services to the top management should be the following:

- internal benchmarking
- external benchmarking
- asset sizing
- multilevel reporting
- feeding of Balanced Scorecard or dashboards

B. *Processes*

The processes of a ICT-MS should be identified, shared and described in documents of the Quality Management System. Documenting an ICT-MS requires both writing specific procedures and changing those ruling the production and delivery of ICT products and services. If measurement is integrated in the usual production processes then more reliable data will actually be collected. If measurement is perceived as a task that does not contribute to the project then involved stakeholders may resist providing reliable data.

 ISBN: 978-1-61208-458-9

C. Methods, Techniques and Tools, Infrastructures

The choice of methods, techniques and tools must be done in strict accordance with the goals and objects of measurement. The techniques are specific business approaches, described by objectives, deliverables, work steps, means, standards used etc. that can achieve limited results within a broader methodological framework. For example 'the structured interview' is a technique for data collection that can be adopted in different methodological framework. Tools are usually software applications that support the use of techniques and methods or they are collections of templates or documents.

Organizations often reverse the correct sequence that starts from information needs, derives necessary metrics, sets out the responsibilities of measurement and identify the techniques and tools needed.

Pressed by market forces and victims of too simplified approaches, so captivating as inconclusive, the manager who is not particularly well-informed believes that with the purchase of last generation tools, all the problems of measuring ICT are automatically and magically solved.

Despite the bitter disappointments in the ICT history when technologies have substituted "system thinking", it is always much easier to acquire a tool than designing a major organizational change.

If properly attributable to the role for which they were born - helping and not replacing human expertise - the tools are invaluable for several reasons: they

- ensure greater uniformity and standardization of measurements made in different contexts;
- can automate repetitive and boring work parts of measurement, accelerating related processes and making them, consequently, less expensive;
- can automatically detect some measures from the products or services;
- allow cross-checks that are difficult to achieve manually;
- reduce the possibility of errors and render the measurement more reliable;
- can 'capture' knowledge and make it available to people with little experience.

According to the PSM framework, tools can be classified as follows:

TABLE 1 CLASSIFICATION OF TOOLS TO SUPPORT THE MEASUREMENT FUNCTIONS

Type of Tool	*Support Function*
Database, Graphing and Reporting	Store and manage measurement data to produce graphical and text-base reports
Estimation Models	Provide predictive capabilities, such as cost estimation models and reliability models
Statistical Analysis	Provide enhanced analytical capabilities such as regression
Schedule and Project Management	Assist in project scheduling and tracking resource allocations and expenditures
Financial Management	Support collection and storage of funding Data
Product Analysis	Generate automated analyses of specific product characteristics (e.g., complexity)
Data Collection	Automatically extract measurement data from elements of the engineering process

D. Information System

The information system of the organization should be prepared or modified to permit, as smoothly as possible, the collection, processing and distribution of basic data and indicators. Sometimes it is necessary to juxtapose "official" data and "unofficial but real" data. This is why the connection between business information systems and support systems for measurement can not be mechanistic but should have to be flexible and configurable. For example, it is possible that in some organizations it is not officially allowed to register and charge the staff overtime to the projects, however, this does not mean that the overtime is not done. Very often professionals are involved beyond the pure material compensation and they prefer to release a good product even if it will cost some or many hours of unpaid work. In this way the accounting figure does not keep track of what actually happens and the apparent productivity will be better than the real one. In this way the estimates that will be made on future projects from the current actual data will contain a systematic underestimation.

Since there are many stakeholders in the process, the tools should be modular:

 ISBN: 978-1-61208-458-9

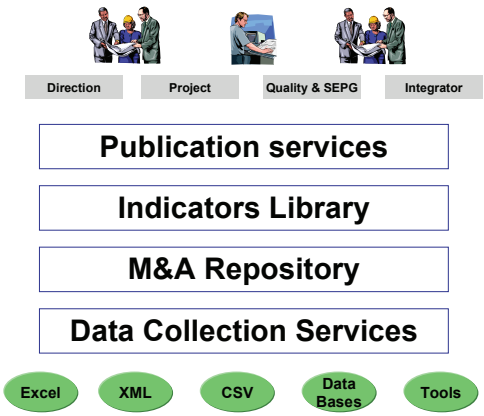


Figure 7. Information System architecture

If we have a look at Figure 7, [13] and we start from the bottom, we can identify:

Data Collections Services: in any office environment there are a lot of sources which can provide data. There will be many Excel files, sometimes some XML or Text files (CSV), usually a lot of data bases (ERP, etc.), and a lot of tools (configuration management, change request management, planning, requirement management, test, ….).

It is very important that the M&A solution provides services that help one easily build a new data collector and reuse it at any time. It should be possible to adapt it to a new situation. The data collector should automate the collection of data, as frequently as one needs, and wherever the sources are.

M&A Repository: this is the knowledge database. Histories of projects will help to predict the future with accuracy. In general, this is an SQL database.

Indicators Library: once an indicator is specified, it must be possible to reuse it on another project. The other project could also reuse an extractor. The entire M&A system must be designed with a "reuse" approach. In large organizations having a lot of projects and a good level of process maturity, it is important to build a project dashboard in a few seconds. In the Library, it should be also possible to find templates of indicators, like curves' profiles.

Publication services: the purpose of this component is to provide the information products to the allowed people, with security management. The dashboard must be accessible through the web, wherever the user is, always updated.

E. Organization

The organization of an ICT-MS can take different forms and structures depending on a number of variables to be considered. The main structural options concern the degree of centralization / decentralization of the measurement function with respect to the production structures, its size in terms of human resources and the mixture of insourcing / outsourcing.

A centralized function concentratesamong under one manager all the necessary professional resources devoted to the provision of metrological services, has a budget for the procurement of products and consulting and is particularly suitable to attract and retain talent in the field of software measurement. A structure of this type provides services to the rest of the organization as any other supporting techno-structure. It can define service level agreements and internal protocols of services engagement.

Among the advantages of a centralized organization, with respect to a distributed one, we include the following points:

- guarantee of greater consistency in measurement
- development of specific professional roles
- better management of interproject priorities
- reduction of redundant costs
- easy sharing of the know-how

Among the disadvantages:

- greater organizational effort
- dependence of the projects by an "external" body
- greater complexity of planning
- less awareness on the issues of measurement by productive staff
- possible rejection as a "foreign body" by the production facilities

With respect to the dimensioning of the measurement organizational units, observe that it must be calibrated in function of the requests for service which, in turn, depend on the number of projects / services to be performed, the degree of maturity of the processes, from the extension of the field of application of measures / estimates, the number of measures to be taken, the degree of turbulence of the processes, the degree of formality of the contractual aspects associated with products / services ICT and other factors.

Finally, regarding the mixture of outsourcing / insourcing options, it can be said that, being the measurement service associated with a process that does not belong to the repertoire of "core processes" and being very easily measurable in itself, it is an ideal candidate for a full outsourcing. Any choice, going from a total delegation to an exclusively internal service, is possible. Full outsourcing may be a starting point in order to have a very quick activation of the system using external expertise. As soon as the maturity of organization and internal skill grow up at a sufficient level it is possible to internalize many activities. A balanced outsourcing may be an arrival point using external resources to complement internal capabilities for peak

 ISBN: 978-1-61208-458-9

workloads or the provision of very specialized and unusual skills at the state of the art. In any case the government of the whole process must remain an internal capability.

Factors to consider when choosing organizational options are:

- Company size
- Frequency of use of the measures
- Average level of measurement experience
- Cultural maturity in measurement subjects
- Level of awareness and managerial support
- Specificity of application domains
- Resources overload
- Confidentiality domain.

In an ICT Measurement System, there are at least 3 categories of persons.

- Project : People who are developing and maintaining the software or system
    - Provide objective information
    - Provide subjective information
    - Attend training
    - Produce lessons-learned experience
    - Use provided processes and models
- Measurement & Analysis Team: people who understand, assess and refine the ICT-MS
    - Analyse experiences
    - Develop models and relationships
    - Produce standards and training
    - Provide measurement services
    - Provide feedback
- Technical Support: people who maintain the information repository
    - Write data collection procedure
    - Establish database structure
    - Quality Assurance
    - Archive data and documents

F. *Competencies*

Staff skills are critical to the success of the measurement process. Depending on the adopted organizational solutions it may be possible to engage new already trained personnel otherwise it will be necessary to identify internal candidates and to develop their professional skills. The level of rapid obsolescence of the industry is such that training must be continued and implemented both through traditional classroom interventions and through e-learning and self-training options. It is often necessary or appropriate to pursue certification of competencies regarding specific methods. Of course, the learning path will be different for every position in the ICT-MS.

To be able to manage its tasks, the M&A team should have a lot of technical skills for building measurement plans, Balanced Score cards, Tableaux de Bord, GQ(I)M (Goal Question (Indicator) Metric), SPC (Statistical Process Control), ETL (Extract Transform Load Techniques), Estimate, benchmark, causal analysis, Function Point Analysis, SLOC (Source Line Of Code), Earned Value Analysis, etc.

In addition to technical competencies it should be considered that the measurement professional must play a consultant role interacting with people not often very happy to be involved and sometimes impatient to go back to programming and "producing". Soft skills are then as important as technical knowledge.

G. *Motivations*

As written, a metric system that does not provide motivation (rewards and penalties), for the application of measurement procedures will not be used reliably. We must develop the whole project of creation of the Metric System around the understanding of the true goals of the people involved, their interests, attitudes, approaches, cultural and operational needs. The measures, once developed the correct preparation of the field, may also become an instrument of evaluation of individual performance, but this takes time and organizational maturity: it is not advisable to start with this goal in mind, it is an arrival point and not a starting point. The system of measures should be designed so that all persons involved in the process of collecting, processing and distribution of information and metrics understand why and what they are doing and, possibly, what benefits brings them.

V. ICT-MS LIFE CYCLE

As we have seen, an ICT-MS is a permanent entity that provides services throughout the organization on an ongoing basis. No organizational system, however, is created and evolves on its own, there must be an organized and temporary effort whose purpose is the design, construction and start-up of the system itself. The supporting activities that have not responsible in the organization and resources allocated to them tend to fall on deaf ears after an initial period of general curiosity. A key to success, for the establishment of an ICT-MS, is the ability of the sponsor to set a credible cost / benefit analysis. We need to imagine the extent of intervention on ICT business support as a systemic initiative in which the various cultural, organizational and technical elements are combined in a harmonious and structured way. So the evolution of the ICT-MS, once started, will be guaranteed by specific improvement initiatives, i.e., change projects (Figure 8).

 ISBN: 978-1-61208-458-9

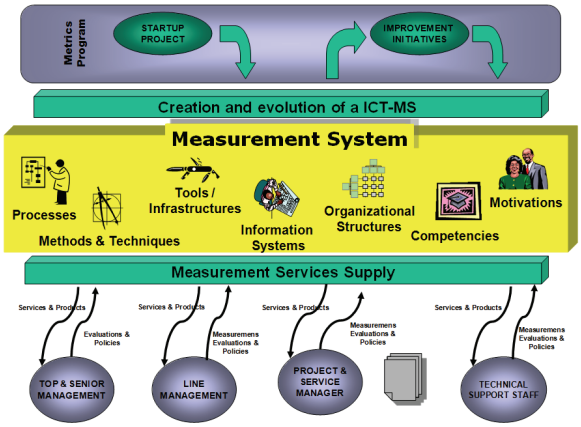


Figure 8. start up and improvement of an ICT-MS

The basic prerequisites for the success of an ICT-MS are:

- the metric system and its objectives are supported by the management;
- measurement must provide added value to all parties involved in the collection, measurement and analysis of historical data;
- the ICT-MS must cooperate and do not come into conflict with other operating systems in an organization, reusing as much as possible of existing assets.

A master plan for a metrics program (building an ICT-MS) may include the following points:

- Setting goals and benefits of the metrics system and its creation project
- Identification of the relevant stakeholders
- Obtaining sponsorship
- Communication and promotion of the initiative
- Identify roles and responsibilities of project
- Preliminary definition of the metrics to be adopted
- Defining procedures and measuring tools
- Definition and management of training plans
- Releasing into operations the ICT-MS

VI. ICT-MS CAPABILITY ASSESSMENT

As in many other ICT areas, it is possible and useful to define a model to assess the capability of an ICT-MS. Currently, there is no specific model for Measurement Systems but measurement is present in many frameworks to assess the global capabilities of ICT organizations like CMM-I, OPM3 [14], Prado [15]. A capability assessment is useful to benchmark an organization in this area and to design an evolutionary path for improvement.

A. *Capability*

Measurement capabilities depend significantly on how the ICT-MS was designed and created, but also by the three other factors shown in Figure 9: commercial practices, competitive context and managerial commitment.

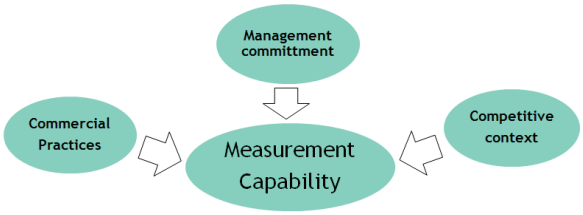


Figure 9. Measurement Capability

The first factor is the mode of relationship between customers / suppliers. If they are focused on management of social relational and adaptive behaviour based on mutual compromises and agreements of a 'political' type, hardly the measure will emerge as a real need, because it takes away flexibility in the implicit/explicit negotiations. Conversely, if the relationships are based on the clear identification of the reciprocal rights / duties and explicit assessment of the interdependence of the factors of negotiation then the measure will become essential to give practical support to the negotiation process.

The second factor mentioned is the competitive context in which a company or public administration is immersed. Monopolistic demand or supply can lead to the superfluity of the measure while the presence of a high number of competitors pushes up the mechanisms of transparency and quantitative comparison.

Finally, the orientation of management is crucial to ensure sponsorship and directions for practical use of the resources made available by the ICT-MS.

B. *Maturity and Adequacy*

Maturity is a general level of evolution of the system, which is demonstrated by the possession of some practices and properties considered as advanced by general consensus. Adequacy is a different concept. It is related to the relationship with the environment and its requests on the organization but it is also a measure of internal consistency of the different ICT-MS components. To be adequate means to be consistent with the environment requests and that the different components show an homogeneous level of evolution. If the context of customers and rulers is informal then a too formal measurement process is a threat instead of an opportunity. The reverse is true too. If we have very skilled measurement professionals but no defined processes and no measurement tools then the ICT-MS is not adequate. In order to be adequate, the organization must be flexible in terms of the system components. Maturity and Adequacy are not related one to each other: it is possible to be mature

 ISBN: 978-1-61208-458-9

and not adequate or the opposite or any combination of the two variables [16].

VII. A SHORT CASE STUDY

The framework outlined in this paper has been adopted by the largest Italian regional ICT public company. In 2010 it started a project to build up an ICT-MS outsourcing the services to start up the system and involving heavily internal resources to go live with the system. The management was satisfied with the results and after 5 years of life the ICT-MS is a stable part of the organization. All the components of the system were addressed properly starting from knowledge evolution (traditional training and on the job training) for more than 140 staffing units and progressing to process and methods definition, tools adoptions, a partially distributed organization (a central competence reference staff and a distributed responsibility in organizational units), administrative data definition, data collection procedures, links to requirements management and analysis, formal outsourcing documents and processes. Measurements goals were added to Management By Objective (MBO) framework to assess manager's performance over the years. The initial start up project ended after 2 years but the measurement activities are part of the ordinary production processes and supported by ordinary budget. The evolution of the ICT-MS is granted by the central functional staff.

VIII. CONCLUSIONS

This paper has presented a relatively new concept named ICT-MS, which is different from Metrics Program that is largely described in literature and standards. The highlight is on the systemic approach and the permanent nature of the system with respect to the "project" oriented approach. An ICT-MS is crucial for the exploitation of the ICT governance because it gives evidence, objectivity and independence from individuals. Any organization interested in the incorporation of measurement in its core processes should devote adequate resources to the set up and maintenance of an ICT-MS taking care of all the components in an integrated way. To help grow in the most efficient and effective way an ICT-MS Capability Model may help in defining the "as is" situation as well as an evolutionary path for improving.

REFERENCES

[1] PMI, "A Guide to the Project Management Body of Knowledge (PMBOK Guide, 5th edition), PMI, pp. 9-10, 2013.
[2] IFPUG, "IT Measurement: Practical Advice from the Experts", Addison-Wesley, ISBN: 020174158X, Part II Measurement Program Approaches, 2002.
[3] IFPUG, "The IFPUG Guide to IT and Software Measurement", Auerbach Publications, ISBN-10: 1439869308, Section V Measurement Programs, 2012.
[4] IFPUG, "Guidelines to Software Measurement", release 2, 2004.
[5] Pressman R.S., "Software Engineering: A Practitioner's Approach," Sixth ed: McGraw-Hill, Chap. 2, 6, 7, pp. 22-26 , 2004.
[6] Fenton N.E., Pfleeger S.L., "Software Metrics: A Rigorous & Practical Approach", 2/e, International Thomson Computer Press, Chap. 1-14, 1998.
[7] Fredreiksen H.D. and Mathiassen L., Information-centric assessment of software metrics practices, IEEE Transactions on Engineering Management, 52(3), 350–362, August 2005.
[8] Practical Software and Systems Measurement, PSM Guide, http://www.psmsc.com/, Nov 2015.
[9] GUFPI-ISMA, Metriche del software. Esperienze e ricerche, Roberto Meli, "The Metric System as a tool to regulate the processes of ICT production and services ", ISBN: 9788846471390, Franco Angeli, 2006.
[10] ISO/IEC 15939:2007, http://www.iso.org/, Nov 2015.
[11] CMMI Institute, CMMI® for Development (CMMI-DEV), Version 1.3, http://cmmiinstitute.com/cmmi-models, Nov 2015.
[12] SWEBOK project, http://www.swebok.org, Nov 2015.
[13] Hamon Patrick, Meli Roberto, A successful roadmap for building complex ICT indicators, SMEF 2007, Rome (IT), http://www.dpo.it/smef2007/, Nov 2015.
[14] PMI, Organizational Project Management Maturity Model (OPM3®) – Third Edition, 2013, http://marketplace.pmi.org/Pages/ProductDetail.aspx?GMProduct=00101463501, Nov 2015.
[15] Prado, PMMM model, http://www.maturityresearch.com/novosite/en/index.html, Nov 2015.
[16] R.Meli, "Adequate maturity in Project Management", Il Project Manager, Franco Angeli, fascicolo 2, 2010.

 ISBN: 978-1-61208-458-9

Distributed Asynchronous Focus Group Interviews

Gathering Requirements from Distributed Stakeholders Using Asynchronous Focus Group Methodology

Ulrike Hammerschall
Department of Computer Science and Mathematics
University of Applied Sciences Munich
Munich, Germany
ulrike.hammerschall@hm.edu

Abstract— **Globally distributed project teams are a more and more common trend in software development. Stakeholders and development teams of the same project are situated in different countries and time-zones. As a consequence, coordination between team members relies heavily on suitable online communication environments. This is especially the case during requirements elicitation, when requirements for a new system need to be identified. Most elicitation techniques require physical presence of stakeholders in order to be effective. This is not always possible in distributed project teams. The question is if and how traditional elicitation techniques can be adapted to distributed project settings. This paper proposes a concept to adapt a special elicitation technique - traditional face-to-face focus group interviews – to online focus group interviews. The concept proposes a discussion model based on a questionnaire that allows conducting asynchronous online focus groups in online environments as similar as possible to traditional face-to-face focus group discussions. Furthermore, a process model is introduced to plan and conduct asynchronous online focus groups. Finally, the paper discusses open issues of the concept that need further investigation.**

Keywords- focus group; distributed asynchronous focus group; requirements elicitation.

I. INTRODUCTION

Requirements Elicitation is the process of finding requirements for a software system. A common technique for requirements elicitation is interviews with stakeholders [1]. Interviews in requirements elicitation are usually performed face-to-face with each stakeholder. The interviewer asks a list of questions based on a questionnaire and documents answers from the interviewee.

Besides traditional interviews, the use of focus group interviews has emerged as an effective elicitation technique as well [2]. Focus groups are a powerful social interviewing technique that allows researchers to elicit several viewpoints from users at the same time [3]. Individuals are asked to participate in what is usually a structured interview on a predesignated topic [4]. During a focus group session, data is collected through group interaction on a topic determined by the researcher [5]. Focus groups emerged as a qualitative research method used in market research or social sciences [6]. Its strength is to reveal hidden information through group interaction in addition to information that could be gathered by face-to-face interviews.

In traditional focus groups, the interviewer and the interviewees meet in a room and discuss face-to-face. In global project teams presence of all group members at the same time in the same room might not always be possible. In this case, focus group discussion needs to be conducted online. Research in focus groups addresses this problem. Due to the widespread use of internet technology, online focus groups have emerged during the nineties [7][8]. Online focus groups can be performed synchronously or asynchronously. Synchronous groups are similar to face-to-face focus groups, as they are conducted in real time via chat or video-conferencing. Asynchronous groups on the other hand, do not require real time attendance of participants during a session. Communication is done via forums or email. Participants can contribute, read and comment on contributions from other group members [8].

Traditional face-to-face focus groups are a well-established elicitation technique. However, this type of group discussion is not useful in distributed project environments when stakeholders live globally distributed in different countries and time-zones. There are platforms available for asynchronous online group discussions, however, support for focus group methodology is still rare [9].

This paper proposes a concept for tool-supported asynchronous online focus groups (AOFG). This includes a model for online discussions and a process model to plan and conduct focus group events with distributed participants. Both models - discussion and process model – can be used in distributed, tool-supported environments.

Section 2 starts with a survey on focus group methodology and identifies relevant aspects that need to be considered in online focus group methodology. Based on this, a discussion model for focus group sessions is proposed in Section 3. Section 4 defines a process model to prepare, conduct, and analyze AOFG. This paper marks the first step of a larger research project. Therefore, Section 5 discusses open issues and research questions for further investigation. Related work and summary in Sections 5 and 6 round up this paper.

 ISBN: 978-1-61208-458-9

II. CONDUCTING FOCUS GROUPS

Focus groups are a carefully planned discussion, designed to obtain the perceptions of the group members on a defined area of interest [10]. The term "focus" (or "focused") refers to the fact that a moderator intervenes to shape the discussion using a researcher-determined strategy [4]. The group setting enables the participants to build on the responses and ideas of the others, which increases the richness of the information gained [11]. Traditional (face-to-face) focus groups are usually conducted in similar ways with small variations:

Group setting: A group of people is gathered in one room and is discussing a topic. Group size is usually small. Many authors propose a group size between four and twelve participants ([10][13][14][15][18]). Size is a crucial aspect for group success. Large groups are more difficult to manage. They require a higher level of moderation and control which might not be desirable for the research topic of the group [15]. On the other hand, it might be difficult to maintain an active discussion in a smaller group. Small groups also run the risk of being less productive. They work best when the participants are likely to be both interested in the topic and respectful of each other [15].

Duration: Duration of focus group sessions depends mainly on group size. Kitzinger proposes session length up to two hours [12]. Powell and Single determine session time from 90 up to 120 minutes [6].

Roles: Focus groups are usually based on a two role model: moderator and participant. To [13] the moderator is quite critical to the success of the group. He or she supervises and guides group session in order to achieve the best results for the research question. Davis proposes a third role, the client [7]. This might be reasonable, e.g., in case of market research. Client representatives of the product under discussion observe focus group discussion without interfering.

Discussion methodology: Focus groups are group interviews. Therefore, there are several ways to conduct a group session. An obvious proceeding would be to ask each participant in the group the same question and document his or her answers. However, Kitzinger states that group interaction should explicitly be used as part of the interview method. The interviewer has a series of open ended questions to discuss within the group. Participants are encouraged to explore the questions and talk to one another, asking questions, exchanging anecdotes and commenting on each other's experiences and points of view [12]. Powell and Single propose up to six open ended questions for a focus group session [6].

Most aspects of face-to-face focus groups, e.g., group size or role model can be easily adapted to asynchronous online focus groups. Discussion methodology on the other hand will need to be adapted to distributed environments.

III. DISCUSSION MODEL

In this section, a discussion model for asynchronous online focus groups is proposed. This model adapts as much as possible face-to-face group discussions to an online environment, when discussion participants are not available in real time. Figure 1 summarizes the model structure. Each online session needs - similar to face-to-face focus groups - two roles: a moderator who supervises the discussion and several participants who conduct the discussion. In the center of any focus group session is a questionnaire with a list of questions that guides the focus group discussion. The questionnaire and the questions are prepared by the moderator based on group objective and context information.

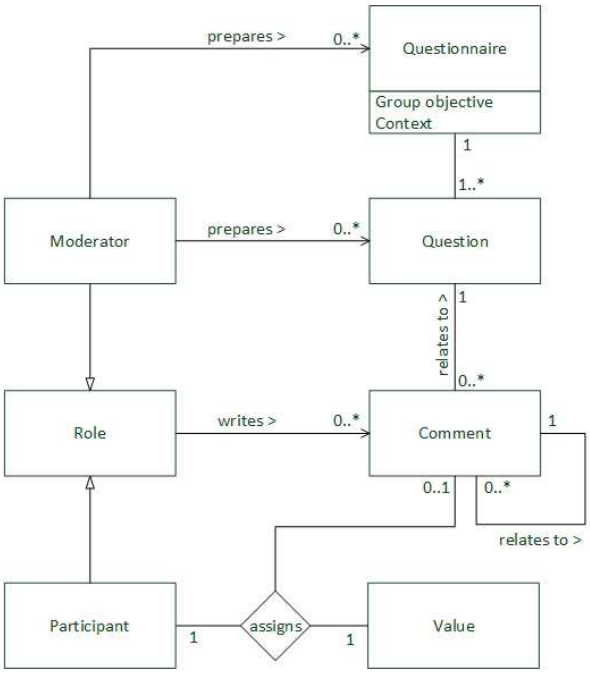


Figure 1. Discussion model for focus group sessions with questionnaire.

The list of questions constitutes a kind of backbone for focus group discussion. All participating roles may comment on questions during the focus group session. A comment might be an answer to the question, another question, a thought, further information about objective and context, a reply to another comment or just a note. Even the moderator may participate and enter the discussion. However, his or her task is to keep the discussion focused on the initial research question.

Asynchronous online focus groups lack the power of nonverbal interaction in face-to-face focus groups, e.g., body language, facial impressions, approving or disapproving nods. A simple mechanism of evaluation helps to overcome this disadvantage in some ways. Any participant may assign a value of approval to any comment. Values are visible to all group members. A participant may never assign a value to his or her own comments. Finally, the moderator should not assign any values to comments in order to avoid any influence on group interaction.

The model discussed in this chapter tries to keep formalization of focus group discussion as simple as possible

 ISBN: 978-1-61208-458-9

in order to stay flexible and not to hinder discussion flow. Possible model refinements are discussed in Section 5.

IV. PROCESS MODEL

In this section, a process model for planning and conducting asynchronous online focus groups is defined. The model supports the discussion model defined in Section 3. It is based on findings in [3][10][13] and the principles for traditional face-to-face focus groups identified in Section 2.

Step 1: Define online focus group objective.

The focus group method is well suited to generate ideas or discuss concepts [10], but it is not realistic to expect explicit and well defined requirements. The objective of a focus group for requirements elicitation can be, for example: share experience about process, legacy system and system context or generate new and exciting ideas for possible requirements. Input to this process step is information about business objective, system idea and constraints. The result of this step should be a clear view on focus group context, focus group objective and a coarse questionnaire to guide the discussion. Furthermore, focus group composition with expected participant profiles and viewpoints should be determined.

Step 2: Plan online focus group event.

Preparing an online focus group event requires several organizational tasks. First of all, an online platform for focus group discussion has to be set up. This includes setting up a new questionnaire with questions and ensuring platform availability and access rights to participants. Finally a time-frame for discussion has to be defined, in particular start, end and, duration. Distributed groups will need considerably more time to ensure that any participant has sufficient time to follow and contribute to the discussion.

Step 3: Define and recruit participants.

Group size plays a crucial role in focus groups. Small groups may be more efficient than large groups. However, in case of online focus groups, a higher number of participants might be reasonable in order to achieve a lively and more fruitful discussion. Finding and recruiting participants for a focus group event requires a high amount of effort. This step needs to follow a process to determine how to identify possible participants and motivate them to engage in the focus group event. Recruited participants need access to the discussion platform and need to be informed about discussion schedule.

Step 4: Conduct and analyze focus group session.

In online focus groups, not all participants might be available at the start of the discussion. The moderator has to ensure that any participant receives the information necessary to enter the discussion. This includes discussion procedure, rules and duration. The discussion itself is based on questions and comments. Questions are published by the moderator (one at a time or all at the same time), participants keep discussing by writing comments on questions and other comments. To express agreement with a comment or point out its importance any participant may assign a value. The number of values assigned to a comment may give a hint about its importance with respect to the initial group objective and constitutes an important means for focus group analysis.

V. RESEARCH QUESTIONS

The model discussed in this paper adapts face-to-face focus group methodology as much as possible to online focus groups. The main idea behind this approach is that in general face-to-face focus groups are the best solution, but in some circumstances personal attendance of participants is not possible. In this case, online focus groups could be the second best solution. However, the question remains if online focus groups could be a methodology of their own. Research questions are for example:

1. Asynchronous online focus groups allow a highly variable groups size due to their virtual character. There are no physical limitations as for example room size. An interesting question to investigate would be what is a good group size for online focus groups to achieve the best results in requirements elicitation?

2. Group discussions that last over a couple of days may have a problem to motivate participants and to keep them engaged in focus group discussion. A corresponding research question would be how to improve motivation and achieve high engagement even over a longer period of time? This may include questions about using gamification techniques in online group discussions.

3. Traditional discussion methodology for face-to-face focus groups is based on questionnaires. The discussion model proposed in Section 3 adapts this approach to online focus groups. However, this might not be the best solution. Investigation can reveal discussion models more suitable for online focus groups. Maybe a more specific approach that distinguishes between questions, comments, jokes, answers, and so on might be more appropriate.

4. Focus groups usually use a two-role-model: a moderator supervises focus group session and several participants take part and discuss. Online focus groups may use a different role model. The moderator could for example be supported by co-moderators. Another model could use the client role as proposed in [7]. Discussion about online focus groups could find flexible role models that can be adapted to specific focus group events.

5. The approach presented in this paper addresses requirements elicitation in distributed stakeholder teams. However, it might as well be suitable for another research area that emerged recently: crowd requirements engineering. Requirements are elicited via crowd sourcing. The objective is to provide the engineering team access to a wide diversity of actual and potential users of new products [17]. An interesting research question could be how to successfully perform online focus group sessions with a potentially unlimited number of participants (the crowd).

 ISBN: 978-1-61208-458-9

VI. RELATED WORK

Group techniques for requirements elicitation are mostly group discussions with a specific discussion methodology. Examples are group work, brainstorming, requirements workshops or Joint Application Development (JAD) [1]. Using these techniques with asynchronous distributed stakeholder groups requires suitable tool support. A study conducted by Zarinah and Salwah reveals that there is a trend towards group based requirements elicitation tools. Discussion technique is mainly group meeting, group discussion and participatory design [9]. Group based requirements elicitation tools using focus group methodology are rare. The authors of [9] introduce a multi-viewpoint approach for tool-supported focus groups in requirements elicitation based on an iterative elicitation algorithm. A similar approach based on chat-messages is proposed by Davis [7]. The objective in this case is to support marketing research with distributed stakeholders.

Lloyd et al [16] investigated in a study the effectiveness of elicitation techniques in distributed environments. They found that requirements elicitation techniques like Question and Answer method, Customer Interview or Brainstorming were effective in distributed environments. However, the authors state that synchronous environments seem to be more effective than asynchronous environments.

The approach proposed in this paper tries to overcome this drawback. Presuming that concentration on well-defined requirements as a result of online group discussions may hinder creativity and group dynamics the approach proposed in this paper concentrates on group discussion methodology, how to support group dynamics in an online environment and how to engage participants that cannot share group session in real time.

VII. CONCLUSION

This paper proposes a concept to support asynchronous online focus group interviews as similar as possible to traditional face-to-face focus groups. In the center is a discussion model for focus groups based on a predefined questionnaire. Group members discuss by commenting on questions asked by a moderator. A comment can be any type of information: a real answer, a joke, an opinion, information or an experience report. Group members can assign marks of approval to contributions they think valuable to the topic under discussion. Furthermore, the concept defines a process model for planning and conducting asynchronous online focus groups using the discussion model.

The concept is still under research. Next steps will be to provide an implementation and evaluate the concept in real life project environments. Further research questions include group engagement and motivation as well as improved data analysis.

VIII. REFERENCES

[1] D. Zowghi and C. Coulin, "Requirements Elicitation: A Survey of Techniques, Approaches, and Tools," In: A. Aurum and C. Wohlin (eds.) "Engineering and Managing Software Requirements," pp. 19–46. Springer Verlag, Berlin/Heidelberg, (2005).

[2] Z. M. Kasirun and S.S. Salim, "Focus Group Discussion Model for Requirements Elicitation Activity," In: 2008 International Conference on Computer and Electrical Engineering (ICCEE), pp. 101–105.

[3] R. Mazza and A. Berre, "Focus Group Methodology for Evaluating Information Visualization Techniques and Tools," In: 2007 11th International Conference Information Visualization (IV '07), pp. 74–80.

[4] E. Perecman and S. R. Curran, "A Handbook for Social Science Field Research. Essays & Bibliographic Sources on Research Design and Methods," SAGE Publications, Inc, 2455 Teller Road, Thousand Oaks California 91320 United States of America (2006).

[5] D. L. Morgan, "Focus Groups," Annu. Rev. Sociol. 22, 129–152 (1996).

[6] R. A. Powell and H. M. Single, "Focus Groups," International Journal for Quality in Health Care 8, 499–504 (1996).

[7] H. O. Davis, "System and method for conducting focus groups using remotely loaded participants over a computer network," Google Patents, http://www.google.com/patents/US6256663 (2001).

[8] R. Rezabek, "Online Focus Groups: Electronic Discussions for Research," Forum Qualitative Social Research 1 (2000)

[9] M. K. Zarinah and S. S. Salwah, "Supporting collaborative requirements elicitation using focus group discussion technique," International Journal of Software Engineering and Its Applications 3, 59–70 (2009).

[10] J. Kontio, L. Lehtola, and J. Bragge, "Using the focus group method in software engineering: obtaining practitioner and user experiences," International Symposium on Empirical Software Engineering, 2004. ISESE '04, pp. 271–280.

[11] J. D. Langford and D. McDonagh, "Focus groups, Supporting effective product development," Taylor & Francis, London, New York (2003).

[12] J. Kitzinger, "Qualitative research. Introducing focus groups," British medical journal 311, 299 (1995).

[13] L. Litoselliti, "Using focus groups in research," continuum research methods (2003).

[14] R. A. Krueger: "Designing and Conducting Focus Group Interviews," University of Minnesota (1998).

[15] "Focus Groups as Qualitative Research," SAGE Publications, Inc, 2455 Teller Road, Thousand Oaks California 91320 United States of America (1997).

[16] W. J. Lloyd, M. B. Rosson, and J. D. Arthur, "Effectiveness of elicitation techniques in distributed requirements engineering," In: IEEE Joint International Conference on Requirements Engineering, (2002) pp. 311–318.

[17] M. Hosseini, K. Phalp, J. Taylor, and R. Ali, "Towards Crowdsourcing for Requirements Engineering, " in Proc. of the Empirical Track of REFSQ, (2014) pp. 82–87.

[18] M. Shitanshu and R. Rekha," A Software Solution to Facilitate Moderation, Observation and Analysis in a Focused Group Interview (FGI)," In: IEEE International Conference on Technology for Education, (2013) pp. 9-12.

 ISBN: 978-1-61208-458-9

Applying Privacy by Design in Software Engineering - An European Perspective

Karin Bernsmed
Department of software engineering, safety and security
SINTEF ICT
Trondheim, Norway
karin.bernsmed@sintef.no

Abstract— **Privacy by Design (PbD) is an approach to protect privacy by embedding it into the design specifications of technologies, business practices, and physical infrastructures. However, despite its many advantages, many organizations struggle with incorporating these practices in their existing software engineering processes. This paper evaluates the current state-of-the-art related to PbD in software engineering and analyzes the impact of the proposed European data protection legislation on this process. We propose four key viewpoints of PbD and discuss how these can be applied in a software engineering process. We then translate these viewpoints into a self-assessment method that can be used to evaluate to what degree an organization has managed to adopt the PbD mindset in their software engineering projects.**

Keywords-privacy; PbD; privacy engineering; personal data; EU data protection law

I. INTRODUCTION

Privacy and personal data protection issues have been frequently in the news during the last few years, in particular in the context of social networking, big data and cloud computing. Consumer profiling by online advertising companies is a huge market and the loss of privacy is the price that consumers have to pay for the free services that they utilize. At the same time, the right to data protection is a highly developed area of law in Europe. Creating and maintaining software that is compliant with European data protection laws are therefore crucial for organizations that want to do business in Europe.

Broadly speaking, personal data means any kind of information that can be used to identify an individual. Some obvious examples include someone's name, address, national identification number, credit card number or a photograph. Less obvious examples are metadata in electronic documents, log files and system configurations and IP addresses. Personal data is not just information that can be used to identify an individual directly; information that can be used to single out a person from a group of people using a combination of information (or other identifiers) will also fall in the category of personal data. Almost all software that provides services targeted towards individual end-users will therefore collect personal data and hence be subject to applicable data protection law.

Privacy by Design (PbD) is an approach to protect privacy by embedding it into the design specifications of technologies, business practices, and physical infrastructures. PbD consists of seven foundational principles [1]:

1. *Proactive not Reactive; Preventative not Remedial*, which means to anticipate and prevent privacy invasive events before they happen.
2. *Privacy as the Default Setting*, to ensure that personal data are automatically protected in any given IT system or business practice. No action is required by the user – privacy is built in by default.
3. *Privacy Embedded into Design*, not bolted on as an add-on. Privacy becomes an essential component of the core functionality being delivered.
4. *Full Functionality — Positive-Sum, not Zero-Sum*, meaning that one seeks to accommodate all legitimate interests and objectives in a positive-sum "win-win" manner. The purpose is to avoid dichotomies, such as privacy vs. security or privacy vs. functionality.
5. *End-to-End Security — Full Lifecycle Protection*, to ensure that all data are securely retained throughout its entire lifecycle, and then securely destroyed at the end of the process, in a timely fashion.
6. *Visibility and Transparency — Keep it Open*, to assure all stakeholders that whatever the business practice or technology involved, it is in fact, operating according to the stated promises and objectives.
7. *Respect for User Privacy — Keep it User-Centric*, which requires architects and operators to keep the interests of the individual uppermost by offering such measures as strong privacy defaults, appropriate notice, and empowering user-friendly options.

PbD hence implies a proactive integration of technical privacy principles in the design of a system or software (such as privacy default settings or end-to end security of personal data) as well as the recognition of privacy in a company's risk management processes [2]. According to Ann Cavoukian, the Ontario Canada information and privacy commissioner who first coined the term, PbD can thus be defined as "an engineering and strategic management approach that commits to selectively and sustainably minimize information systems' privacy risks through technical and governance controls" [1].

PbD is often presented as the solution to the digital world's privacy problems. However, despite the obvious advantage with adopting the PbD approach, many organizations still struggle with how to incorporate these practices in their existing software engineering processes [3]. The seven principles are expressed in abstract terms and there are many open challenges that need to be addressed. *Privacy engineering* has emerged as a concept for transforming the PbD principles into a framework for

 ISBN: 978-1-61208-458-9

implementing privacy in system design and development processes. As concluded from the 2014 NIST Privacy Engineering Workshop [4], there is currently a communication gap around privacy between the legal and policy, design and engineering, and product and project management teams, which makes it difficult to understand and manage privacy risks. Moreover, there is a need for tools that measure the efficiency of existing privacy practices in organizations.

The purpose of this paper is to help organizations apply the Privacy by Design concept in their software engineering lifecycle by providing support for analyzing the current situation and practical guidance for building in PbD data protection practices that are compliant with European Data protection legislation. The paper is organized as follows. Section II summarized existing guidelines, tools and research related to engineering Privacy by Design. In Section III, we discuss the legislative aspects of PbD in Europe. Section IV outlines our approach to integrating PbD in a software engineering process and Section V presents a self-assessment method for PbD. Finally, Section VI concludes our work.

II. STATE OF THE ART

In this section, we summarize existing work related to PbD. We pay particular attention to the papers and reports that provide practical guidance on how to operationalize PbD, i.e., how to integrate the principles into existing software engineering processes. We also provide an overview over relevant ongoing research efforts in Europe.

A. Reports and Guidelines from the Software Industry

The report "Operationalizing Privacy by Design: A Guide to Implementing Strong Privacy Practices" from 2012 [5] gives a thorough introduction to the seven principles of PbD and provides practical advices for how each of the different principles can be implemented in an organization and by whom (i.e., the management, the application and program owners or the software engineers). For each of the PbD principles, the report also outlines a number of different case studies from different domains and explains how this particular principle has been implemented in practice. The report represents an overview over the work that has been performed at the Information and Privacy Commissioner in Ontario, Canada.

The OASIS Privacy by Design for Software Engineers (PbD-SE) Technical Committee has developed a draft specification to help document software engineering make privacy-informed decisions about a system's architecture. Their Privacy Management Reference Model and Methodology (PMRM) [6] intend to help system architects to analyze the system from a privacy point of view and to help them identify necessary technical and process mechanisms that must be implemented to support existing privacy policies in the organization. The methodology is based on defining and analyzing how actors and systems integrate in use-cases and the report contains a number of illustrative examples of how this can be done. PMRM is primarily specified with the Fair Information Practice Principles (FIPPs) [24] in mind, however it also supports parts of the PbD concept since it encourages building privacy in already from day one of a system design.

Microsoft's guidance document "Privacy Guidelines for Developing Software Products and Services" [7] includes an overview of basic concepts and definitions that are related to software security and provides guidelines for how the principles notice, choice, onward transfer (to third parties), access, security, and data integrity should be implemented. The document includes several practical examples (figures) showing how many of the concepts, for example explicit consent and opt-in, have been implemented in Microsoft's own software portfolio.

Finally, the position paper "Privacy Engineering & Assurance" written by NOKIA in 2014 [10] presents a process consisting of a set of proactive engineering activities. The activities include identifying the privacy impact of a given object, designing controls and mitigations to ensure appropriate Privacy by Design, and then verifying that the implementation is complete and operational, while documenting evidence of this state for reference of regulatory compliance and in the event of a privacy breach.

B. Relevant Research on Privacy by Design

The paper "Engineering Privacy" from 2009 by Spiekermann and Cranor [2] provides a structured overview over the different topics included under the term privacy engineering. The paper introduces the term privacy spheres to categorize the collection of personal data w.r.t whether the data are stored at the users' own devices under their own control (the so called "user sphere"), in back-end servers and networks under the service providers' control (the "recipient sphere"), or a combination thereof where users have some control over their personal data (the "joint sphere"). Spiekermann and Cranor recognize the necessity to consider the users' privacy expectations as well as possible regulatory issues when analyzing how system activities will impact privacy and they point out a number of different privacy issues that the needs to consider when designing IT systems. They also give some guidance for how to design a "privacy friendly" system, based on the degree of identifiability that will be required by its users, and provides some practical advices for how to maximize privacy for different types of systems. The paper also provides a nice overview over the existing research disciplines in the field of information system privacy.

The paper "Engineering Privacy by Design" by Gurses et.al, [3] points out data minimization as the necessary first step in order to create systems that are in line with the PbD concept. The authors point out the lack of concrete guidance of how to actually implement the PbD principles and they further argue that the FIPPs' focus on control and transparency, and the European data protection regulation's focus on purpose specification and user consent, are not sufficient to protect the individuals' privacy. The paper

 ISBN: 978-1-61208-458-9

presents two case studies where the authors show how privacy risks can be heavily reduced when data minimization is applied. They generalize their findings into five main steps for system design that should be taken to reduce privacy risks: 1) Functional Requirements Analysis (the necessary system functionality is clearly described), 2) Data Minimization (for each functionality, the data that are absolutely necessary to fulfil the functionality is analyzed), 3) Modelling Attackers, Threats and Risks (models of attackers and threats are developed, and the likelihood and impact of the threats are used to do a thorough risk analysis), 4) Multilateral Security Requirements Analysis (to ensure that the security and correct behavior of the system), and 5) Implementation and Testing of the Design (making sure that the system fulfils the integrity requirements revealing the minimal amount of personal data and that the functional requirements are fulfilled). Finally, the authors point out the need for experts trained in privacy engineering methodologies that also have a basic understanding of legal requirements related to personal data protection.

The paper "Privacy Design Strategies" by Hoepman [11] presents eight privacy design strategies, which are derived from legal requirements from the European data protection legislation. The strategies are derived both from a data oriented perspective (focusing on the principles minimize, hide, separate and aggregate) and from a process oriented perspective (focusing on the principles inform, control, enforce and demonstrate). For each of the eight strategies, the author has identified a number of privacy design patterns that can be applied to implement the strategies. The paper represents work in progress and the author state that further research will be performed to classify existing privacy design patterns into privacy design strategies, and to describe these design patterns in more detail.

Privacy by Design is also a topic of investigation in several ongoing European FP7 research projects; the most prominent being CIPHER [15], which will analyze security and trust in information systems that process personal data, and provide a methodological framework and a global European regulatory and technological roadmap, PRIPARE [16], which will deliver a privacy and security-by-design software and systems engineering methodology, A4Cloud [17], which will (amongst other things) deliver a Privacy Impact Assessment tool for cloud services and USEMP [18], which aim to empower users with control over the sharing of their personal data. In particular, PRIPARE is relevant to our work since they aim to deliver a methodology for Privacy and Security by Design that can be embedded into current methodologies for ICT systems and software [12].

Our analysis of the existing work in this section concludes that either the existing guidelines on PbD do not consider the strict EU personal data legislation in their guidance documents [2][3][5][6][7] or (implicitly) they assume that the organization that will operate the software develops its own software [4][10][11]. Even though there are promising ongoing research efforts, much work remains to be done. In particular, there is currently a gap of knowledge in how PbD can be built in the procurement phase of IT systems for organizations that engage consultancies or external software development companies and that have little or no knowledge of how to derive security and privacy requirements and how to impose such requirements on their software vendors. In the next section, we will present the main implications of the existing personal data protection legislation is Europe, before we proceed with presenting our approach for applying PbD.

III. PbD in European Data Protection Legislation

The processing of personal data in Europe is regulated by the implementation of the Data Protection Directive ("the Directive") [19], which ensures that personal data can only be collected and used legally under strict conditions, for a legitimate purpose, and that the data subject, who is an identified or identifiable natural person, must always be informed about the intention to collect and use his/her data. According to the Directive, the person, or organization, that is defined as the data controller, i.e., the entity that determines the purposes and means of the processing of personal data, will (in most cases) be held responsible and accountable to the data subject for ensuring that personal data are processed according to the rules in the Directive. Even though the Directive aims to protect the privacy of individuals, it only supports a limited part of PbD, and to a very limited extent. For example, as pointed out in the RAND report [8], while privacy policies are considered to be an acceptable way to meet the legislative requirements of obtaining consent and providing transparency, these policies are rarely read and even if they are, they appear to serve little useful purpose for the data subject due to their length, complexity and extensive use of legal terminology.

However, with the evolution of regulation, PbD has received more attention. In 2012, the Commission proposed a major reform of the EU legal framework on the protection of personal data (the "proposed Regulation") [9]. The European Commission has explicitly stated that the Proposed Regulation will embrace the concept of Privacy by Design [20]. Unfortunately, the current version of the Proposed Regulation is still quite general and vague. The most relevant part of the Proposed Regulation, from the PbD perspective, is its Article 23 - Data protection by design and by default. The first paragraph in this article states that "*the controller shall, both at the time of the determination of the means for processing and at the time of the processing itself, implement appropriate technical and organizational measures and procedures...*". Even though this statement indicates that privacy must be considered both when the system is to be designed ("the time of the determination of the means") and when it is operating ("the time of the processing itself"), nothing is said of how these requirements should be implemented in practice. Further,

 ISBN: 978-1-61208-458-9

the second paragraph of Article 23 states that "*The controller shall implement mechanisms for ensuring that, by default, only those personal data are processed which are necessary for each specific purpose of the processing...* " and "*... those mechanisms shall ensure that by default personal data are not made accessible to an indefinite number of individuals*". Here we note that, even though the "by default" part of PbD is supported, the Proposed Regulation does not aim to minimize the purpose of data collection at all; it merely states that the default setting should be to only process data for a specific purpose. This requirement already exists in the current EU Data Protection legislation. Further we note that the Proposed Regulation only points out the controller as being responsible for implementing these mechanisms. In many practical cases settings (for example when public cloud services are adopted) the controllers will not be involved in neither the design nor the implementation of the system. In our opinion, even though the European Commission has emphasized that the Proposed Regulation will support PbD; it is unclear to whether it will have any impact at all on existing software engineering processes. This view is also shared by Koops and Leenes, who argue that Article 23 cannot, and should not, "be read as a procedural requirement to embed data protection rules as much as possible in system design, but instead as a substantive requirement calling upon data controllers to consistently keep privacy at the front of their minds when defining system requirements" [14].

Even though PbD is vaguely described in Article 23, there are other parts of the Proposed Regulation, which will strengthen the rights of the data subjects. One example is Article 17, which emphasizes the data subjects right to "be forgotten", meaning that the controller must be prepared to erase all links to, copies of and replications of the data subject's personal data. Another example is Article 18, which specifies that a data subject has the right to obtain a copy of all his/her personal data that has been collected.

Awaiting the Proposed Regulation, several of the European Data Protection Authorities (DPAs) have started to promote the PbD concept, for example the British ICO [21] and the Norwegian Datatilsynet [22]. However, similar to the SOTA presented in Section III, there is a clear gap between the advices provided by these authorities and the concrete mechanisms that must be implemented in the software in order to be compliant with the Proposed Regulation.

IV. INTEGRATING PBD IN THE SOFTWARE SYSTEM ENGINEERING PROCESS

It is a non-trivial path for an organization with little knowledge of security and limited funds to go ahead and implement the best practices presented in Section II and the requirements that steam from the regulation presented in Section III. Very little research has been done to address the real world challenges of using the proposed methods in organizations, apart from the large software companies. This is especially the case where the organization has no dedicated software security or privacy group, which is often the case in, for example, SMEs and the public sector where few, if any, dedicated developers are employed. Instead, procurement and integration of commercially available (or open source) software into the enterprise architecture is more common, often along with custom built software components for integration and various functionality "plumbing". Consultants are commonly used for development and integration, making it hard to establish privacy and security engineering practices within the organization. In cases like these, the data management lifecycle, which spans from the moment personal data are gathered by the organization until the moment they have been destroyed (i.e., the retention period), is in the hands of the organization itself whereas the software engineering lifecycle, which spans from the early design phase until the software is fully installed and operating, is managed by the consultants.

Moreover, implementing PbD in the software engineering lifecycle is in itself a multidisciplinary exercise, comprising technical, organizational and legal concerns. A properly defined set of security and privacy policies must for instance be in place for application owners and developers to elicit specific sets of security and privacy requirements. On the other hand, true support is a matter of the management in the procuring organization, ensuring that the organization has the capabilities needed to accomplish its mission.

Figure 1. The stakeholders involved in the software engineering process.

An organization that wants to implement the seven PbD principles therefore needs to concretize them into a set of actions that the organization needs to consider internally, as well as into a set of well-formulated privacy requirements that they will need to impose on their consultancies and/or or software vendors during the analysis, design and implementation phases of the development of the software itself. This is a process that will need the involvement of a wide range of stakeholder (illustrated in Fig. 1).

When analyzing the seven PbD principles (from a software engineering point of view) and the different documents that were reviewed in Sections II-III, we have concluded that there are four distinct viewpoints of PbD,

 ISBN: 978-1-61208-458-9

which is top-down in the sense of involving both the organization as well as the actual software engineering process and that will require the involvement of the stakeholders identified in Fig. 1. These four viewpoints will be presented in the next section, along with an introduction to the accompanying PbD self-assessment tool that we have created.

V. A SELF-ASSESSMENT METHOD FOR PBD

The self-assessment method that we propose consists of four different viewpoints. First and foremost, we maintain that *privacy must be acknowledged in the organization*. This viewpoint implies that privacy must be taken seriously by the management and that a privacy mindset should be adopted by those who are responsible for the systems that process personal data. In our view, acknowledging privacy means, for example, that the organization has appointed a privacy officer who is accountable for privacy protection, a privacy policy has been established and approved by the management and that PIAs, or privacy risk assessments, are regularly performed within the organization.

Secondly, organizations need to be transparent about their privacy practices; any organization that processes personal data needs to inform the data subjects about the processing of their personal data. The *privacy policy* (or set of privacy policies) is the statement that discloses the details of what data will be collected, how it will be used and with whom it may be shared. The organization's privacy policy must be compliant with data protection legislation and it must be actively enforced in all its IT systems, including the software that is to be developed. Unfortunately, due to their complexity, difficult language and sheer length, users tend to neither read nor understand the policies prior to acceptance [8][13]. Having adopted a PbD mindset, the organization's privacy policy should therefore be clearly written and easy to access, contain no ambiguous language, and be as restrictive as possible in terms of how much data that will be collected and how it will be used.

Having acknowledged privacy in the organization and having a proper privacy policy in place are two fundamental cornerstones that the organization needs to have in place before the software system procurement phase starts. The former will ensure that sufficient attention and resources are put in place to protect privacy and the latter will serve as a basis for deriving appropriated privacy requirements when the software development process starts. These two viewpoints will need the involvement of business owners, regulators, 3rd parties and application developers.

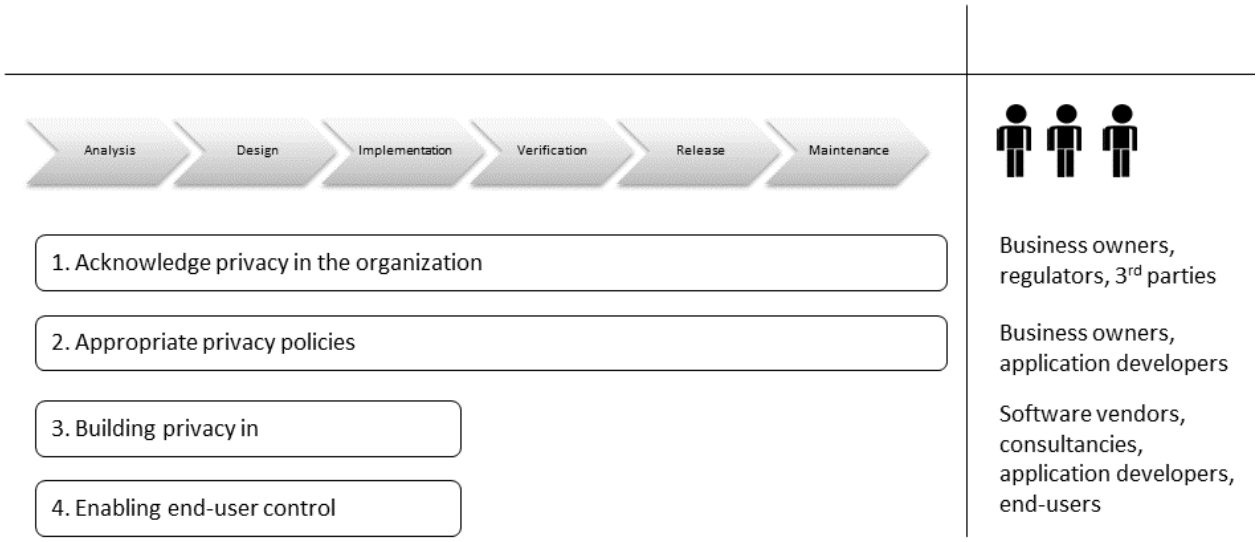


Figure 2. The role of the four viewpoints in the different phases of the standard waterfall software development process.

Once the software development processes has started, the third viewpoint, *building privacy in* is invoked. This viewpoint aims to ensure that privacy is integrated into the early phases of the software engineering process, in particular during the analysis, design and implementation phases. Software specific privacy requirements will be elicited from relevant stakeholders (business and application owners, regulators and the intended end-users), the privacy requirements must validated towards the organization's privacy policy and existing PbD best-practices are incorporated into the code by the software development team.

Finally, the fourth viewpoint *enabling end-user control* will ensure that the intended users of the software (i.e., the individuals who will be the data subjects of the personal data that will be collected) will be in control over his/her personal data. This viewpoint will ensure that the users are empowered with mechanisms to change their privacy settings, give and withdraw consent, and view, correct and delete personal data that have already been collected.

 ISBN: 978-1-61208-458-9

Fig. 2 illustrates how these four viewpoints relate to a standard waterfall software development process and what stakeholders that will be involved in each of the viewpoints. As indicated in the figure, acknowledging privacy and appropriate privacy policies are continuous processes that need to be in place before the software development activities starts and that will persist during the lifetime of the software. These processes will involve business owners, regulators, 3rd parties (with whom the data may be shared) and application developers in the organization. On the contrary, building privacy in and enabling end-user control consist of activities that will be accomplished during the analysis, design and implementation phase and that will involve the software vendors and prospective consultancies, the application developers in the organization and representatives of the end users who will be data subjects when the software is operating.

In the rest of this section, we present the self-assessment method, which has been organized as a checklist (Table I-IV) that has been derived from the four viewpoint introduced in the previous section. The checklist has gone through several iterations with security and privacy experts, before converging to 43 questions to be treated as recommendations, (i.e., answering "yes" is better than answering "no"). We then introduce a simple tool for analyzing the results of applying the checklist to an ongoing or finalized software project. Note that the tool itself is an adapted version of the security checklist for water network operators, originally developed by Jaatun et.al [23].

In our checklist, we have prepared three possible answers; "yes", "partly" and "no", however, it is of course also possible to use for example a sliding scale to indicate to what degree the organization that is being assessed is compliant with the different statements. We do not stipulate what methods the organization should apply to answer the individual checkpoints, but envision a combination of interviews, document analysis and testing as being an appropriate approach.

TABLE I. ACKNOWLEDGING PRIVACY IN THE ORGANISATION

Checkpoint	Yes	Partly	No
The organization has appointed a privacy officer, who is accountable for privacy protection			
A privacy policy has been established and approved by the management			
PIAs, or privacy risk assessments, are regularly performed within the organization			
Privacy audits are regularly performed within the organization			
Notice of personal data processing has been given to all the relevant DPAs			
Data processing agreements have been established with all 3rd parties that will process personal data			
The organizations' software and infrastructure regularly undergoes security risk and threat analysis			
The organization has a privacy education/awareness training program			
The organization is prepared to handle security incidents affecting personal data			

TABLE II. APPROPRIATE PRIVACY POLICIES

Checkpoint	Yes	Partly	No
The amounts of personal data that can be collect have been minimized			
The purpose for data collection has been defined to be as specific as possible			
Any sharing of personal data to 3rd parties has been clearly specified			
The retention date is no longer than necessary to fulfil the purpose of data collection (or to comply with existing legislation)			
The privacy policy clearly states who are responsible for the personal data and how they can be contacted			
The privacy policy is clearly written, to make it easy to understand by the intended end-users			
The length of the privacy policy is not excessive, but kept to a minimum			
The privacy policy can easily be retrieved by customers and end-user at all times			

TABLE III. BUILDING PRIVACY IN (SOFTWARE SUPPORT)

Checkpoint	Yes	Partly	No
Software specific privacy requirements have been elicited from relevant stakeholders (business and application owners, regulators and the intended end-users)			
The privacy requirements are consist with the organizations' privacy policy			
The privacy requirements have been incorporated in code developed by the software engineers			
The software only collect the personal data necessary to deliver its intended functionality			
The software includes appropriate mechanisms for obtaining end-user consent			

 ISBN: 978-1-61208-458-9

Checkpoint	Yes	Partly	No
The software has mechanisms in place to limit the use of personal data to the specific purpose for which it was collected			
The software has mechanisms in place to avoid future data linkage			
The software will encrypted all personal data by default using standardized encryption mechanisms with securely managed encryption keys			
All personal data are anonymized whenever possible			
There is an expiry date associated with all personal data that are collected			
All collected personal data will be properly deleted after they expire			
The software provides audit trails showing how personal data have been collected, processed and deleted			
The software has been subject to a thorough security risk and threat assessment			
The focus on privacy has not been traded against functionality			

TABLE IV. ENABLING END-USER CONTROL

Checkpoint	Yes	Partly	No
The default privacy settings in the software are as restrictive as possible			
The user can change the settings that control what kind of personal data are collected			
The user can change the settings that control for what purpose personal data are collected			
The user can view what personal data have been collected			
The user can view who has access to the personal data that will be collected			
The user can view who has accessed the personal data that have been collected			
The user can make corrections to personal data that have been collected			
The user can export a copy of all personal data that have been collected			
The user can request personal data to be immediately deleted			
The user's personal data is not shared with 3rd parties, unless the user specifically agrees to this ("opt-in")			
The user can choose not to share personal with 3rd parties ("opt-out")			
The user's privacy settings are valid across different platforms and persist over time			

If the checklist is used to evaluate a software engineering process that has already started, or software that is already operating, the answers can be visualized in order to show to what degree the PbD concept has been adopted. We have implemented a simple Excel-based tool and applied it to a case study that we are working on. The case study involves a public organization in Scandinavia, which currently is preparing a pilot study of the usage of cloud-based software for remote monitoring of health-care patients in their homes. Security and privacy are high on the agenda for this organization and since the software will collect large amounts of (sensitive) personal data, they need to be compliant to the existing privacy legislation in Europe, as well as to the upcoming privacy regulation, in order to succeed with their project. (For confidentiality reasons we are not allowed to reveal any technical details about the case study.) The result from the first viewpoint for this organization is illustrated in Fig. 3.

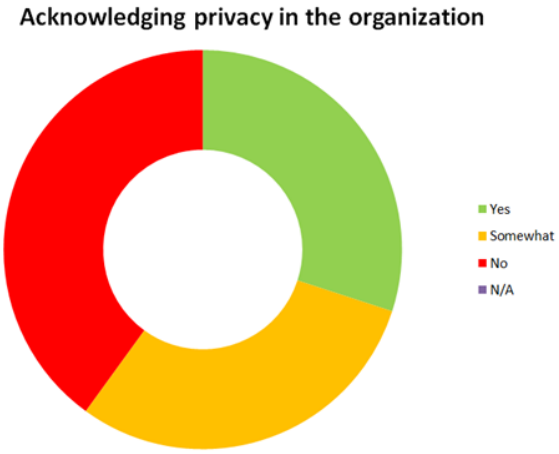


Figure 3. Visualizing to what degree privacy has been acknowledged in the organization

In the figure, the colors green, yellow and orange have been used to visualize the ratio of answers that have been selected as "yes", "partly" and "no", respectively. From the figure, we can see that, even though this particular organization have fulfilled some of the identified checkpoints, they still have a long way to go before privacy has been fully acknowledged.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented four viewpoints of Privacy by Design and our approach to translate these into a list of checkpoints. The intention of our approach is to clarify what the PbD concept means in a software engineering context. We also aim to help organizations that are involved in personal data processing to adopt a privacy mindset and to make sure that their software is compliant with the vision of PbD. In the next step, we will compile a best-practices document that includes existing privacy design patterns, strategies, mechanisms and tools, and map

 ISBN: 978-1-61208-458-9

these to the checkpoints in our self-assessment checklist in order to identify whether there are any gaps that current technology cannot fulfil. We believe that a combination of technical mechanisms (PETs) and organizational measures will be necessary in order to fully adopt the PbD concept.

ACKNOWLEDGMENT

This work has been partly funded from the European Commission's Seventh Framework Programme (FP7/2007-2013) under grant agreement no: 317631 (OPTET) and 317550 (A4CLOUD).

REFERENCES

[1] A. Cavoukian, "Privacy by Design Curriculum 2.0", 2011. [Online]. Available from: https://www.ipc.on.ca/ [retreived: 2015-10-29]

[2] S. Spiekermann, and L. Faith Cranor, "Engineering Privacy". IEEE Trans. Softw. Eng. 35 (1), pp 67-82, January 2009, doi=10.1109/TSE.2008.88

[3] S. F. Gürses, C. Troncoso, and C. Diaz, "Engineering Privacy by Design", Computers, Privacy & Data Protection, 2011. [Online] Available from: http://www.cosic.esat.kuleuven.be/publications/article-1542.pdf [retreived: 2016-01-12].

[4] NIST Privacy Engineering Objectives and Risk Model Discussion Draft, April 2014. [Online]. Available: http://www.nist.gov/itl/csd/upload/nist_privacy_engr_objectives_risk_model_discussion_draft.pdf [retreived: 2016-01-12].

[5] A. Cavoukian, "Operationalizing Privacy by Design: A Guide to Implementing Strong Privacy Practices". Information and Privacy Commissioner, Ontario, Canada, December 2012.

[6] OASIS. "OASIS. Privacy Management Reference Model and Methodology (PMRM) Version 1.0.", March 2012. [Online]. Available from https://www.oasis-open.org/ [retreived: 2016-01-12]

[7] Microsoft. "Privacy Guidelines for Developing Software Products and Services, Version 3.1", September, 2008. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=16048 [retreived: 2016-01-12]

[8] N. Robinson, H. Graux, M. Botterman, and L. Valeri, "Review of the European Data Protection Directive", 2009, RAND Corporation. [Online]. Available: http://www.rand.org/pubs/technical_reports/TR710.html [retreived: 2016-01-12]

[9] Proposal for a Regulation of the European Parliament and of the Council on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data (General Data Protection Regulation), COM (2012) 11 final (25 January 2012).

[10] NOKIA, "Privacy Engineering & Assurance. The Emerging Engineering Discipline for implementing Privacy by Design", NOKIA Position Paper 1/10, 2014-09-08. [Online]. Available: http://www.w3.org/2014/privacyws/pp/Hirsch.pdf [retreived: 2016-01-12]

[11] J-H. Hoepman, "Privacy Design Strategies". ICT Systems Security and Privacy Protection, IFIP Advances in Information and Communication Technology Volume 428, 2014, pp 446-459, 2014.

[12] "PRIPARE: A New Vision on Engineering Privacy and Security by Design", PRIPARE position paper, April 2014. [Online] Available from: http://prepareproject.eu/research/ [retreived: 2016-01-12]

[13] L. Faith Cranor, P. Guduru, and M. Arjula, "User interfaces for privacy agents". ACM Trans. Comput.- Hum. Interact., 13(2):135–178, 2006.

[14] B-J. Koops and R. Leenes, "Privacy regulation cannot be hardcoded. A critical comment on the 'privacy by design' provision in data-protection law". Int. Rev. Law Comput. Technol. 28 (2), May 2014, pp. 159-171. doi=http://dx.doi.org/10.1080/13600869.2013.801589

[15] "CIPHER: Integrated Cybersecurity framework and roadmap". [Online] http://cipherproject.eu/ [retreived: 2016-01-12]

[16] "PRIPARE: Preparing Industry to Privacy-by-design by supporting its Application in Research". [Online] http://prepareproject.eu/ [retreived: 2016-01-12]

[17] "Cloud Accountability project". [Online]. http://www.a4cloud.eu/ [retreived: 2016-01-12]

[18] "USEMP: User Empowerment for enhanced online management". [Online]. http://www.usemp-project.eu/ [retreived: 2016-01-12]

[19] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

[20] European Commission, "Progress on EU data protection reform now irreversible following European Parliament vote", Strasbourg, 12 March 2014. [Online]. Available: http://europa.eu/rapid/press-release_MEMO-14-186_en.htm [retreived: 2016-01-12]

[21] Information Commissioner's Office (ICO), "What is 'privacy by design'?" [Online] Available: https://ico.org.uk/for-organisations/guide-to-data-protection/privacy-by-design/ [retreived: 2016-01-12]

[22] The Norwegian Data Protection Authority. http://www.datatilsynet.no/English/ [retreived: 2016-01-12]

[23] M.G. Jaatun, J. Røstum, S. Petersen, and R. Ugarelli, "Security Checklists: A Compliance Alibi, or a Useful Tool for Water Network Operators?", Procedia Engineering, Volume 70, 2014, Pages 872-876, ISSN 1877-7058, http://dx.doi.org/10.1016/j.proeng.2014.02.096.

[24] "Privacy Online: A Report to Congress. Federal Trade Commission", June 1998. [Online]. Available from: https://www.ftc.gov/reports/privacy-online-report-congress [retreived: 2016-01-12]

 ISBN: 978-1-61208-458-9

End User in Charge - Social Framework for Open Source Development

Kwabena Ebo Bennin, Shahid Hussain, Arif Ali Khan, Solomon Mensah, Ernest Pobee
Department of Computer Science
City University of Hong Kong
Hong Kong, China
{kebennin2-c,shussain7-c,aliakhan2-c,smensah2-c, epobee2-c}@my.cityu.edu.hk

Mohammed Alqadhi
Department of Computer Science
Jazan University
Jazan, Saudi Arabia
malqadhi@jazanu.edu.sa

***Abstract*— Open Source Software (OSS) is often developed in a public collaborative manner. Online OSS repositories such as GitHub, Google Code and SourceForge support collaborative OSS development by offering services such as subversion management, bug tracking and others. However, OSS mostly favors end-users who are programmers or have some pre-requisite programming skills. The normally short README description file provided by the OSS developers does not contain enough information to help the novice end-users who intend to use the software in terms of installation and usage. Also, despite being equipped with social coding feature to support distributed multi-developer work environment, most OSS repositories provide only a storage space for the OSS files and this limits end-users just to their bugs/review comments on a different platform and naturally, people would also like to be key stakeholders like changing the functionality and accessibility of software they could use. Some online OSS repositories do not make provision for users to frequently communicate with the developers of the OSS to discuss about the published content on the repository. In this paper, we propose a social framework for OSS development to address the aforementioned issues. The framework is aimed to allow: (1) knowing the degree of matching between the sought user's requirements and the available OSS by presenting the end-user with the business domain model of a candidate OSS associated to its textual requirements description and (2) a lifetime communication between the users and OSS developers and even inviting other developers out of the OSS development team if needed.**

Keywords-Open Source Software; OSS; End User; Crowdsourcing; Social development.

I. INTRODUCTION

Open Source Software (OSS) development is an approach to the design, development, and distribution of software offering accessibility to software's source code for modification or enhancement. OSS has contributed to software technology by providing end-users from many sectors such as governmental/non-governmental organizations, businesses and individuals around the world leverage to customize OSS for their personal needs (see Table I). Basically, this wide adoption of OSS is because an OSS is a freeware and promotes reuse through code transparency and a quality alternative to close source software [8][9]. Such quality is a result of collaborative efforts of developers from all over the globe and also the flexibility of allowing team members to contribute as much as they can, whenever they want [7]. Therefore, this partially defeats the classic concept that only a centralized management and strong control on the access to the source code produces a good and high quality software product. OSS has also dispelled in practice the view that rigorous management and a clearly defined design is instrumental for a successful software development project because many open source software projects have been successfully completed even without a clear initial design and formal management process [1].

Most OSS developers contribute to development of projects not because of money but as a way of giving to the society freely [6]. There exist many Web-based software repositories for hosting OSS such as GitHub [10] and SourceForge [11] on the world wide Web. They provide the service of social software development by facilitating multi-developer OSS projects and offering subversion control, bug tracking, release management, mailing lists and wikis.

TABLE I. EXAMPLES OF POPULAR OPEN SOURCE SOFTWARE

Usage Domain	OSS Example
Office Productivity Suites	Apache openOffice, libreOffice, Neo Office, Calligra
Finance and Accounting Applications	GnuCash, TurboCASH
ERP Software	ADempiere,OFBiz, OpenERP
CRM Software	SugarCRM, OpenCRX, Fat Free CRM
Communication and Telephony Software	AsteriskNOW, Elastix
Content Management Systems	Drupal, Wordpress, OpenCms, Joomla
E-commerce Tools	OpenCart, PrestaShop

 ISBN: 978-1-61208-458-9

The most often usage scenario for nowadays' Web-based OSS repositories begins with an end-user who looks for an OSS project that satisfies a set of requirements. The presented results based on the end-user's search on the OSS repository Website will be of one of these two possibilities: (1) finding a list of preexisting OSS that possibly partially fulfills the end-user's requirements or (2) finding no candidate OSS that matches or satisfies the end-user's requirement, thus, the end-user would have to create a totally new system from scratch. Also, most available OSS repositories do not provide the user with useful documentation information to help decide the degree to which a stored OSS satisfies a targeted user's requirements.

Mostly, an OSS will be associated only with a README file containing only technical information such as configuration, installation and others. For example, a simple search for a point of sale software may bring results of more than 30 different OSS systems. The end-user will be confused as to which software he/she should select. The end-user could end up installing/testing nearly half of the resulted list of systems before obtaining the desired system.

Another limitation of these OSS repositories is the lack of developer support. This difficulty stems from the lack of frequent communication channels between OSS users and developers. Usually, end-users have to contact developers through their personal homepages and e-mails. For that, Websites for network of questions and answers, such as StackOverflow.com are the popularly targeted venues by OSS end-users. StackOverflow helps end-users to discuss and solve their problems with developers who voluntarily offer help and support. Research works in [3][4] have studied a group of OSS authors (i.e., those who developed the OSS) and the committers (i.e., those who reuse the OSS) who have been identified as active on both GitHub and StackOverflow platforms. Both studies observed that there is a positive connection between participating in StackOverflow and the productivity on GitHub. Also, the study in [5] found that end-users do not follow up with the OSS project within the OSS repository but rather, they go to the community Websites such as StackOverflow [2] when faced with any problem concerning the OSS project. All of the aforementioned studies can be indicatives of the difficulty the end-users endure in trying to communicate with the authors and developers of an OSS project on the OSS repository platform.

To address these two issues, we propose a social framework for OSS projects that will: (1) provide a mechanism for storing software requirements on an easy-to-manipulate format in order to facilitate the process of matching between the functionalities provided by the OSS and the sought end-user's requirements. (2) incorporate social networking feature to frequently connect developers and users. (3) Moreover, end-users are involved in the process of social reviewing and crowd testing of the OSS being developed.

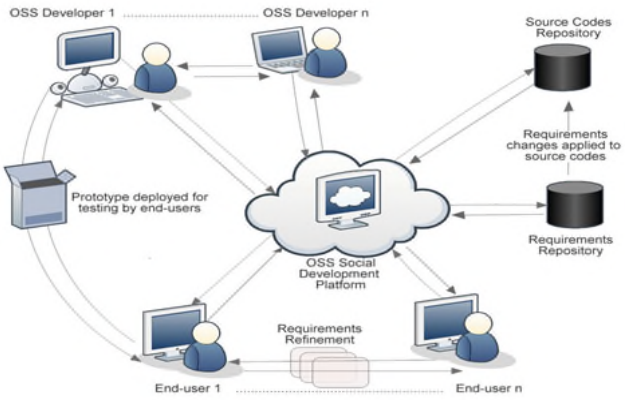


Figure 1. An overview of the proposed framework.

The framework (Figure 1) will be a Web-based service that offers a platform to undertake a multi-developer OSS project. It also provides a social networking facility to ensure all users (i.e., the developers and end-users) are connected.

II. OVERALL FRAMEWORK

A. Framework Components

The framework mainly consists of three components as follows:

1) Source Code Repository: this repository will contain the source code files for all OSS projects on the platform. Also, it will keep track of all changes made by different users by means of subversion management facility to ensure that each user's corresponding changes are linked to the changes in the requirements repository.

2) Requirements Repository: this will store the requirements documentation for all OSS projects available on the repository in XML-based format namely, XML-based Requirements Description Language (XRDL).

3) Social Network: this component will frequently connect all users in the repository (developers and users).

B. XML-Based Requirements Description Language (XRDL)

We propose an XML-based language for providing a well-organized, structured and easy-to-manipulate format for storing requirement documents in the OSS requirements repository. This feature emanates from the inherent dynamicity of XML, as it is a dynamic markup language where one can define his/her own structures and constructs. Dynamic requirements description can be accomplished through user-defined requirements tags that define different requirements-related constructs. The tags may define the following constructs:

- Requirements
- Service type
- System Category
- Language
- User
- User Story
- Use-Case
- Class

 ISBN: 978-1-61208-458-9

```
<?xml version="1.0" encoding="UTF-8"?>
<requirements reqid="001"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="requirement.xsd">
 <system> <service>new</service>
<platform>PHP</platform></system>
<preference> <category>Information
systems</category></preference>
<story><user>tenant</user>
<body>user should get management fee receipt, report incidents, check
incident handling     progress</body>
</story>
<story><user>Security Guard</user>
<body>update fee payment for tenants, update incident report status,
generate incident report</body></story>
</requirements>
```

Figure 2. A sample XRDL document mapped from requirements collection form.

An XRDL document (Figure 2) will be generated automatically by mapping the information retrieved from a form filled by the user at the beginning of the OSS development cycle. State-of-the-art techniques like textual analysis already employed within contemporary CASE tools will help in extracting artifacts such as actors, use-cases and classes from the user entries in the form. By utilizing XML (Figure 3) hierarchical power, these artifacts information will be transformed into different representations. We care about creating different representations of the requirements due to the fact that the platform will bring together volunteering developers from different schools of thought on software development. For instance, if a developer adopts Scrum development method then, he/she will be interested in viewing requirements as user stories. In contrast, a developer who follows a systematic software engineering approach needs to deal with use-cases and class diagrams which are doable by applying textual analysis to the user stories.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="requirement">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="system"><xs:complexType>  <xs:sequence>
 <xs:element name="service" type="xs:string"/>
<xs:element name="platform" type="xs:string"/>
</xs:sequence></xs:complexType></xs:element>
<xs:element name="preference"><xs:complexType>
<xs:sequence>
<xs:element name="category" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="story" maxOccurs="unbounded">
<xs:complexType><xs:sequence>
<xs:element name="user" type="xs:string"/>
<xs:element name="body" type="xs:string"/>
</xs:sequence></xs:complexType></xs:element>
</xs:sequence> <xs:attribute name="reqid" type="xs:string"
use="required"/>
</xs:complexType></xs:element>
</xs:schema>
```

Figure 3. An XML schema to capture requirements information in XRDL document shown in Figure 4.

III. ILLUSTRATIVE OSS DEVELOPMENT STORY

The story begins with an end-user who has a set of business requirements and searches for an OSS that satisfies the requirements. The user will go to the platform and fill in a form that is designed in a very simple and easy-to-understand display to end-users (Figure 4). Upon submission of the form, all developers on the platform will be notified of the newly created requirements record.

Interested developers will pursue and provide feedback to the end-user. Other end-users who have similar requirements or needs can also join thus, they end up in a single development group within the social software development platform.

In the first project's requirements collection form, the collected requirements are mostly composed of user stories. With this proposed framework, since there will be a number of end-users on the same platform, it will be easy to interact and get a clearer picture of the requirements through a social requirements refinement process. That is, the developers and other end-users who are subscribed to the social development platform can help in breaking down and refining the collected requirements (Figure 5).

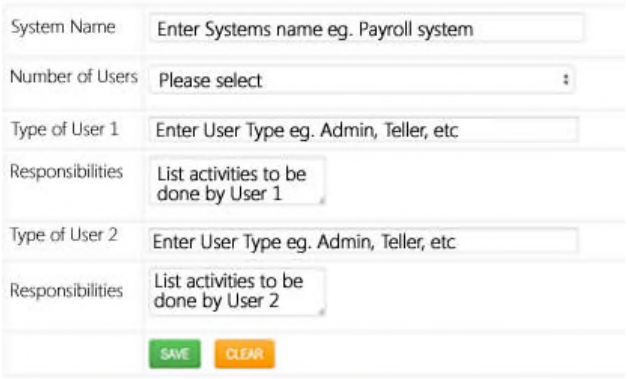


Figure 4. A sample project's requirements collection form.

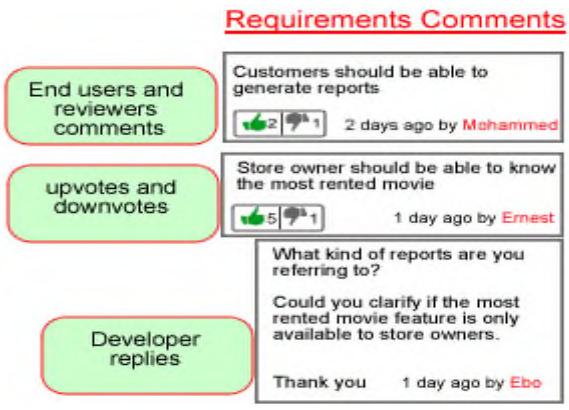


Figure 5. A sample social refinement for OSS requirements.

 ISBN: 978-1-61208-458-9

Should there be more than two end-users and developers on the platform, as end-users keep on refining the requirements, the developers develop and design prototypes. After each prototype, it can be pushed to the end-users for testing. This cycle will be repeated until a fully functional application is developed and all stakeholders are satisfied with the results. Another alternative scenario is when the end-user logs in to the OSS social development platform and searches the software repository for a possible OSS that matches his/her requirements. For the search, a set of indicative keywords related to the users of the systems and usage scenarios to search the repository could be used. The output of the search will be one of the following two scenarios:

a) Returning a list of preexisting OSS that possibly partially fulfills the end-user's requirements.
b) Finding no candidate OSS prompting the end-user to create a totally new system from scratch by filling in the requirements collection form. In case of availability of a match, then the platform will display the candidate list of OSS projects for the user associated with its most updated requirements. The presented requirements will be in the form of a business domain model associated with text-based requirements description (Figure 6). Through the presented business domain model, the end-user can quickly understand the functionality provided by the available OSS projects. The end-user will go through these listed OSS and check whether it matches his requirements either partially or fully.

If an OSS matches the user's requirement fully, then he/she will proceed to use it. However, it is more likely the OSS will match the end-user's requirements partially, so there will be a need to slightly change requirements and consequently, the corresponding OSS code accordingly. Therefore, the user will fill a change request form stating the minor changes to be implemented upon submission to the social development platform. Members who are experienced in the same programming language in which the selected OSS was written or domain or who worked on the previous OSS application will be notified with a new user request. This therefore will ensure there is still continuity of the project and communication with developers.

All changes or updates made to the source code will be saved in subversions. Similarly, the requirements documents will be updated with the latest refined requirements and stored in the repository in XRDL.

Figure 6. Part of the business domain model for a video rental system.

Finally, after completion of development of software, should the end-user encounter any challenges in the usage of the software, they could always return to the framework and interact with the developers to get their bugs fixed or request any update(s).

IV. BENEFITS

We list the potential benefits of using this framework.

Social Requirements Refinement: All active stakeholders (including end-users) who are experts in their domain together with the developers can collaborate and agree on the best requirement data and hence generate the best and necessary requirement for the developers to work with. There will be no misunderstanding with the output at the end of the project since requirements were clearly defined so many times during each information session.

Social Review of Source Code: Since the development is a crowd effort, other developers have the opportunity to review the source codes and make contributions in the areas of bug detection, refactoring and efficient and smart algorithms to make the system as robust as required.

Crowd Testing: A unique robust testing technique is applied in this framework. Testing is not only done by developers but end-users also participate in this process. Hence, bugs hidden during white box testing by programmers will be exposed by end-users. Also, with a large community, a rigorous stress testing approach is applied to the system to ensure the system is stable and can withstand heavy load.

Reuse and sharing of components: The source code repository will be available for reuse and therefore shared among the community.

V. CONCLUSION

The Web is now proliferated with so many open source projects or applications which are at the disposal of end users. The success of OSS on the Web is a clear indication of how end-users would like to be involved in the development of software. However, the platforms for OSS development mostly favor users who are programmers or have some pre-requisite programming skills and provide limited or no technical support. In this paper, we have addressed the challenges faced by end-users in finding and using OSS in the current Web-based OSS repositories. We proposed a social framework that involves end-users during the development of an OSS project.

End-users who need software will present their problems to an open community of developers and end-users and people who have the same or similar interest will collaborate to provide solution for the end-users. In this framework, contributors can willingly divide themselves into groups such as requirements analyst, developers and testers. The framework encourages crowdsourcing and crowd testing to support social end-user development by

 ISBN: 978-1-61208-458-9

which people can easily share problems and associated solutions together with the underlying rationale that the completed project can be used by all contributors.

REFERENCES

[1] J. M. Gonzalez-Barahona and C. Daddara "Free software/open source: Information society opportunities for europe?" Working group on Libre Software, http://eu. conecta. it/paper/cathedral_bazaar. html (2000).

[2] StackOverflow: Q&A Website. Available from: http://www.stackoverflow.com. [accessed December 2014]

[3] B. Vasilescu et al., StackOverflow and GitHub: Associations between Software Development and Crowdsourced Knowledge, In Social Computing (SocialCom), 2013 International Conference on (pp. 188-195). IEEE.

[4] B. Vasilescu, Human aspects, gamification, and social media in collaborative software engineering, in Companion Proceedings of the 36th International Conference on Software Engineering. 2014, ACM: Hyderabad, India. pp. 646-649.

[5] C. Ayala et al., OSS Integration Issues and Community Support: An Integrator Perspective, in Open Source Systems: Long-Term Sustainability, I. Hammouda, et al., Editors. 2012, Springer Berlin Heidelberg. pp. 129-143.

[6] A. Westenholz, (Ed). The Janus face of commercial open source software communities: An investigation into institutional (non) work by interacting institutional actors. Copenhagen Business School Press DK, 2012.

[7] A. Hemetsberger, and C. Reinhardt. "Collective development in open-source communities: An activity theoretical perspective on successful online collaboration." Organization studies 30.9 (2009): pp. 987-1008.

[8] A. Boulanger,. "Open-source versus proprietary software: Is one more reliable and secure than the other?." IBM Systems Journal 44.2 (2005): pp. 239-248.

[9] B. Fitzgerald,. "A critical look at open source." Computer 37.7 (2004): pp.92-94.

[10] C. Gandrud,. "Github: A tool for social data set development and verification in the cloud." Available at SSRN 2199367 (2013).

[11] SourceForge Software repository. https://www.sourceforge.net [accessed December 2014]

 ISBN: 978-1-61208-458-9

Sequence Data Mining Approach for Detecting Type-3 Clones

Yoshihisa Udagawa and Mitsuyoshi Kitamura
Computer Science Department, Faculty of Engineering,
Tokyo Polytechnic University
Atsugi-city, Kanagawa, Japan
e-mail: {udagawa, kitamura}@cs.t-kougei.ac.jp

Abstract— **Code clones are introduced to source code by changing, adding, and/or deleting statements in copied code fragments. Thus, the problem of finding code clones is essentially the detection of strings that partially match. The proposed algorithm is based on the well-known apriori principle in data mining and is tailored to detect code clones represented as sequences of strings. However, the apriori principle may generate too many sequential patterns. The proposed algorithm finds a compact representation of sequential patterns, known as maximal frequent sequential patterns, which is often two orders of magnitude smaller than frequent sequential patterns. Early experiments using the *Java SDK 1.7.0.45 lang* package demonstrate the number of extracted patterns and elapsed time in several contexts.**

Keywords-component; Code clone; Maximal frequent sequence; Longest common subsequence(LCS) algorithm; Java source code.

I. INTRODUCTION

Copying and pasting similar code (or code clones) is very common in large software since it can significantly reduce programming effort and time. However, code clones complicate software maintenance. For example, when an error is identified in one copy, the same error can occur in the code clones. Thus, a maintenance programmer must check all code clones to ensure parallel changes.

Generally, the detection process for code clones comprises two phases, i.e., transformation and matching [1].

(1) Transformation phase: parts of interest of the source code are transformed to another intermediate representation for ease of matching.
(2) Matching phase: the intermediate representation units are compared to find a match.

Because copying and modifying statements are common programming practices, finding code clones that partially match is a challenging task from both practical and technical perspectives. Partially-matching code clones are referred to as type-3 clones, gapped clones, and near-miss clones in code clone detection literature [1]. The term "gap" refers to nothing-match or non-match elements that comprise two code clone candidates.

A number of approaches have been developed for detecting code clones. State-of-the-art research is divided into two categories. The first category is dedicated to code clone detection. Ducasse et al. [2] defined and assessed six degrees of transformation with regard to varying gap sizes of zero, one, and two. Because of limitations of scalability, they restricted themselves to a gap size of zero in some case studies. Roy et al. [3] proposed a near-miss clone detection method called Accurate Detection of Near-miss Intentional Clones (NICAD). NICAD combines language-sensitive parsing with language-independent similarity analysis using an optimized longest common subsequence (LCS) algorithm [4] to detect code clones. Murakami et al. [5] proposed a new token-based method that detects gapped code clones using a local sequence alignment algorithm, i.e., the Smith–Waterman algorithm. They discussed a sophisticated trace back algorithm tailored for code clone detection.

The other category focuses on frequent sequence mining techniques to detect code clones and code change patterns. CP-Miner [6] employs an extended version of CloSpan [7] to support gap constraints in frequent subsequences. It tolerates one to two statement insertions, deletions, or modifications in copy-pasted code. Negara et al. [8] developed a sophisticated data mining algorithm that effectively detects frequent code change patterns. They also identified 10 types of popular high-level code change patterns from mined code change patterns.

The main idea of the proposed approach is a combination of frequent sequence mining and the LCS algorithm to detect type-3 clones.

A sequence is called a frequent sequence if it appears in a given sequence database with a frequency no less than a user-specified threshold (i.e., minSup). Although several algorithms have been proposed for frequent sequence mining, such as CloSpan, ClaSP, and CM-ClaSP [9], one of the drawbacks of these algorithms is that they can present a very large number of sequential patterns. A sequential pattern is maximal if immediate super-sequences are frequent [10]. The maximal sequential patterns are generally a small subset of frequent sequential patterns. The proposed approached employs maximal sequential pattern mining to discover a compact set of clone candidates.

The main contributions of this paper are as follows: (1) development of a code transformation parser that extracts code matching statements; (2) development of a matching algorithm that efficiently detects type-3 clones using a tailored sequential pattern mining algorithm; (3) evaluation of the proposed algorithm using the *Java SDK 1.7.0.45 lang* package with several parameters; and (4) performance comparison of the proposed algorithm to previous methods.

The remainder of this paper is organized as follows. Section 2 presents an overview of the proposed approach. Section 3 describes the proposed algorithm, which discovers clone candidates using maximal frequent sequence mining. Section 4 presents the results of an experimental study using the *Java SDK lang* package. Finally, Section 5 concludes the paper and provides suggestions for future work.

 ISBN: 978-1-61208-458-9

II. OVERVIEW OF OUR APPROACH

Our goal is to detect method pairs or sets that share common code fragments. In the proposed approach, Java source code is initially partitioned into methods. Then, code matching statements are extracted for each method. The extracted statements comprise class method signatures, control statements, and method calls [11]. Our approach consists of the following four steps (Figure 1).

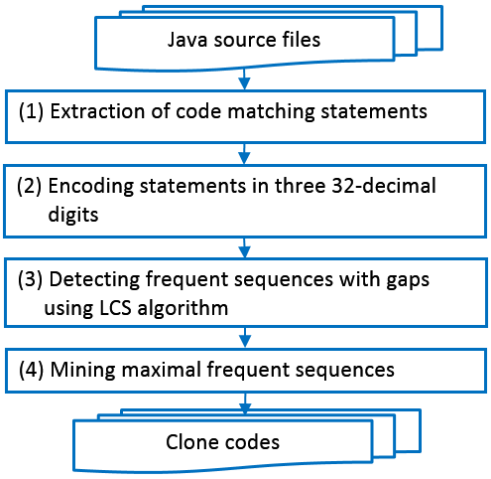


Figure 1. Overview of the proposed approach.

A. *Extraction of code matching statements*

Under the assumption that a method call characterizes a program, the proposed parser extracts a method identifier called in a Java program. Generally, the instance method is preceded by a variable whose type refers to a class object to which the method belongs. The proposed parser traces a type declaration of a variable and translates a variable identifier to its data type or class identifier as follows.

<variable>.<method identifier>
is translated into
<data type>.<method identifier>
or
<class identifier>.<method identifier>.

We have developed a parser that extracts control statements with various levels of nesting. A block is represented by the "{" and "}" symbols. Thus, the number of "{" symbols indicates the number of nesting levels. The following Java keywords for 15 control statements are processed by the proposed parser.

if, else if, else, switch, while, do, for, break, continue, return, throw, synchronized, try, catch, finally

We selected the *Java SDK 1.7.0.45 lang* package as our target. The number of total lines is 67,677. Figure 2 shows an example of the extracted structure of the *encode(char[] ca, int off, int len)* method in the *StringCoding.java* file of the *java.lang* package. The three numbers preceded by the # symbol are the number of comments, and blank and code lines, respectively. The extracted structures include control statement nesting depth; thus, they provide sufficient information for retrieving methods using the structure of the source code.

```
StringEncoder::encode(char[] ca, int off, int len)
#     2     0     25
{
scale()
        if{
        return
        }
        if{
        return
        }
        else{
        CharsetEncoder.reset()
        ByteBuffer.wrap()
        CharBuffer.wrap()
                try{
                CharsetEncoder.encode()
                        if{
                        CoderResult.throwException()
                        }
                CharsetEncoder.flush()
                        if{
                        CoderResult.throwException()
                        }
                }
                catch{
                Error()
                }
        return
        }
}
```

Figure 2. Example of the extracted structure.

In this study, we only deal with Java. However, extraction of code matching statements can allow our approach to be independent of programming languages, such as C/C++ and Visual Basic.

B. *Encoding statements in three 32-decimal digits*

The conventional LCS algorithm takes two given strings as input and compares each character of the strings. However, the length of statements in program code differs; thus, the conventional LCS algorithm does not work effectively. In other words, for short statements, such as *if* and *try* statements, the LCS algorithm returns small LCS values for matching. For long statements, such as *synchronized* statements or a long method identifier, the LCS algorithm returns large LCS values.

We have developed an encoder that converts a statement to three 32-decimal digits, which results in a fair base for a similarity metric in clone detection. Figure 3 shows the encoded statements that correspond to the code shown in Figure 2. Figure 4 shows the mapping table between three 32-decimal digits and a code matching statement extracted from the original source files.

```
StringEncoder::encode(char[] ca, int off, int len)→001→
13V→005→004→003→005→004→003→00C→14F→
141→142→00V→14G→005→144→003→14H→005→
144→003→003→011→07F→003→004→003→003
```

Figure 3. Encoded statements corresponding to Figure 2.

 ISBN: 978-1-61208-458-9

001, {	13V, scale()
002, super()	...
003, }	141, ByteBuffer.wrap()
004, return	142, CharBuffer.wrap()
005, if{	143, CharsetDecoder.decode()
...	144, CoderResult.throwException()
00C, else{	...
...	14F, CharsetEncoder.reset()
00V,try{	...
...	

Figure 4. Mapping table between three 32-decimal digits and a code matching statement.

C. *Detecting frequent sequences with gaps*

We have developed a mining algorithm to find frequent sequences based on the apriori principle [12]. The proposed algorithm is designed to find a set of frequently occurring sequences. Note that several matches can be detected in a sequence for a subsequence given as a matching condition. For example, the proposed algorithm detects the two matches of subsequence A→B in sequence A→B→A→C→A→B→D.

The LCS algorithm is tailored to match three 32-decimal digits as a unit. The LCS algorithm can match two given sequences even if "gaps" (nothing-match or non-match elements) exist. Given two sequences of matching strings S1 and S2, let |lcs| be the length of their longest common subsequence, and let |S1| and |S2| be the length of S1 and S2, respectively. The "gap size" gs is defined as gs= |lcs| − min(|S1|, |S2|).

D. *Mining maximal frequent sequences*

Frequent sequences mining can result in a very large number of sequential patterns, which makes it difficult for users to analyze the results. Mining maximal frequent sequences addresses a drawback of frequent sequences mining [10]. We have developed an algorithm to discover maximal frequent sequences. Note that our approach deals with gapped sequences; thus, it requires a tailored technique to filter non-maximal frequent sequences.

III. PROPOSED FREQUENT SEQUENCE MINING

This section outlines the proposed frequent sequence mining algorithm and shows some examples that demonstrate how the algorithm works.

A. *Proposed Frequent Sequence Mining Algorithm*

The proposed approach is based on frequent sequence mining. A subsequence is considered frequent when it occurs no less than a user-specified minimum support threshold (i.e., minSup) in the sequence database. Note that a subsequence is not necessarily contiguous in an original sequence.

We assume that a sequence is a list of items, whereas several algorithms for sequential pattern mining [9] deal with a sequence that consists of an ordered list of "itemsets." Our assumption is rational because we focus on detecting code clones. In addition, the assumption simplifies the implementation of the proposed algorithm, which makes it possible to achieve high performance (Section 4).

The proposed frequent sequence mining algorithm comprises two methods, i.e., GProve (Figure 5) and Retrieve_Cand (Figure 6). It follows the key idea behind apriori; if a sequence S in a sequence database appears at least N times, so does every subsequence R of S.

```
1  GProve(String[] args){
2     k= 1;
3     Initialization: Set the 15 control statements of Java to LinkedList<String> Sk
4     do {
5        Retrieve_Cand();   // Find a set of sequences of length k+1 that matches Sk.
6                           // Store the set of sequences in Ck.
7        k= k+1;
8        Sk.clear();        // Clear Sk in order to store frequent sequences of length k.
9        while( For all elements e in Ck ){
10          if ( Frequency of  e  >= minSup ){
11                Print e and sequence ids of e in the database to output file.
12                Add e and the sequence ids to Sk;
13                Scan the database to find a set of gap synonyms of e;
14                Add each gap synonym and the sequence ids to Sk;
15                }
16          }
17       }
18    } while (Sk.size() > 0);
19 }
```

Figure 5. Frequent sequence detection of the proposed algorithm.

```
1  Retrieve_Cand(){
2     Ck.clear();
3     for (each element s in Sk) {
4        for (each element t in the sequence database) {
5           for ( each position p that s matches in t){
6              Compute LongestCommonSubsequence between s and t at position p;
7              if (match count >= k && gap count  <=  maxGap ){
8                 Put s and frequency of s to Ck;
9                 Extract gap synonym g of s from t.
10                Put g and frequency of s to Ck;
11                }
12             }
13          }
14       }
15 }
```

Figure 6. Candidate sequences retrieval for the next repetition.

The variable k indicates the count of the repetition (line 2, Figure 5). LinkedList < String > Sk is initialized to hold 15 control statements. The Retrieve_Cand method (line 5, Figure 5) discovers a set of sequences of length k+1 from a sequence database that matches statement sequences in Sk. The while loop (lines 9–17) finds frequent sequences and sequence IDs in a sequence database.

Lines 12–14 maintain the frequent sequences. Note that the proposed algorithm handles gapped sequences, and both a frequent sequence and its "gap synonyms" are prepared for the next repetition. Here, "gap synonyms" means a set of sequences that match a given subsequence under a given gap constraint.

Generally, the Retrieve_Cand() method in Figure 6 works as follows. HashMap <String, Integer> Ck holds a sequence (String) and its frequency (Integer). First, Ck is cleared (line 2, Figure 6). The three for loops examine all possible matches between an element in Sk and sequences in a sequence database. The longest common subsequence algorithm is tailored to compute the match count and gap

 ISBN: 978-1-61208-458-9

count (line 6, Figure 6). The if statement, (line 7, Figure 6) screens a sequence based on the match count and gap count. Lines 8–10 maintain the frequency of sequences and its "gap synonyms."

B. Extracting Frequent Sequences

In our approach, we assume a program structure is represented as a sequence of statements preceded by a class-method ID. Each statement is encoded to three 32-decimal digits so that the LCS algorithm correctly works regardless of the length of the original program statement. The proposed algorithm is illustrated for the given sample sequence database in Figure 7. MTHD# is an abbreviated notation for a class-method ID.

```
MTHD1→005→003
MTHD2→005→00A→003→003
MTHD3→005→003→00F→006→005→003
MTHD4→005→006→003→005→00C
```

Figure 7. Example sequence database.

Figure 8 shows the result of the frequent sequences for a gap of 0 and minSup of 50%, which is equivalent to a minSup count that equals 2. "005" is a frequent sequence with a minSup count of 6 because "005" occurs once in the first and second sequences and twice in the third and fourth sequences. The proposed algorithm maintains an ID-List, which indicates the positions a frequent sequence appears in a sequence database. The ID-List for "005" is 1|2|3+3|4+4.

Similarly, 005→003→ is a frequent sequence with a minSup count of 3, i.e., the ID-List for 005→003→ is 1|3+3.

```
005→          N=6（1|2|3+3|4+4）
005→003→      N=3（1|3+3）
```

Figure 8. Result of the frequent sequences (gap, 0; minSup, 50%).

Figure 9 shows the result of the frequent sequences for a gap of 1 and minSup of 50%. "005" is a frequent sequence with a minSup count of 6, which is the same in the case of a gap of 0.

Similarly, 005→003→ is a frequent sequence with a minSup count of 5. In addition to the consecutive sequence 005 → 003 → , the proposed algorithm detects gapped sequences. In the case of 005→003→, the algorithm detects 005→00A→003→ in the second sequence and 005→006→003→ in the fourth sequence. Thus, the ID-List for 005→003→ is 1|2|3+3|4.

```
005→          N=6（1|2|3+3|4+4）
005→003→      N=5（1|2|3+3|4）
```

Figure 9. Result of the frequent sequences (gap, 1; minSup, 50%).

Figure 10 shows the result of the frequent sequences for a gap of 2 and minSup of 50%. In addition to 005→ and 005→003→, 005→006→ is detected as a frequent sequence because 005→003→00F→006→ in the third sequence matches 005→006→ with a gap of 2, and 005→006→ in the fourth sequence with a gap of 0. Thus, the ID-List for 005→006→ is 3|4.

```
005→          N=6（1|2|3+3|4+4）
005→003→      N=5（1|2|3+3|4）
005→006→      N=2（3|4）
```

Figure 10. Result of the frequent sequences (gap, 2; minSup, 50%).

C. Extracting Maximal Frequent Sequences

A frequent sequence is a maximal frequent sequence and no supersequence of it is a frequent sequence. The set of maximal frequent sequence is often several orders of magnitude smaller than the set of all sequential patterns. In addition, it is representative because it can be used to recover all frequent sequences. Several algorithms for finding maximal frequent sequences and/or itemsets employ sophisticated search and pruning techniques to reduce the number of sequence and/or itemset candidates during the mining process.

However, we wish to measure the effects of a maximal frequent sequence; therefore, the proposed algorithm first extracts a set of frequent sequences and then detects a set of maximal frequent sequences.

Note that, since we deal with a gapped sequence, screening a maximal frequent sequence is required to check that none of the immediate super-sequences of gap synonyms, which are a set of sequences that match a given subsequence under the gap constraint, is a frequent sequence.

IV. Experimental Results

We present some measures on frequent sequences or candidate clones, time analysis, and findings relating to the *Java SDK 1.7.0.45 lang* package. Some features of the code are as follows.

1. After screening methods without control statements or method calls, the normalized code consist of 2,522 methods, 18,205 identifiers, and 1,286 unique identifiers.
2. The maximum length method is *isCallerSensitiveMethod()* (127 lines), which is obtained from the *java.lang.invoke.MethodHandleNatives.java* file.
3. The maximum method nesting level is seven, which is obtained from the *getEnclosingMethod()* method of the *java.lang.Class.java* file.

A. Maximum Length of Retrieved Sequences

The proposed algorithm can retrieve sequences that satisfy an arbitrary gap size specified by the user. Figure 11 summarizes the maximum lengths of the retrieved sequences for each minSup and gap size. As minSup decreases, the filtering condition lessens; thus, the maximum lengths increase. As the gap size increases, the matching condition lessens; thus, the maximum lengths increase. The results in Figure 11 show that the maximum

 ISBN: 978-1-61208-458-9

lengths reach 120 when minSup is 2 and gap size is no less than 2.

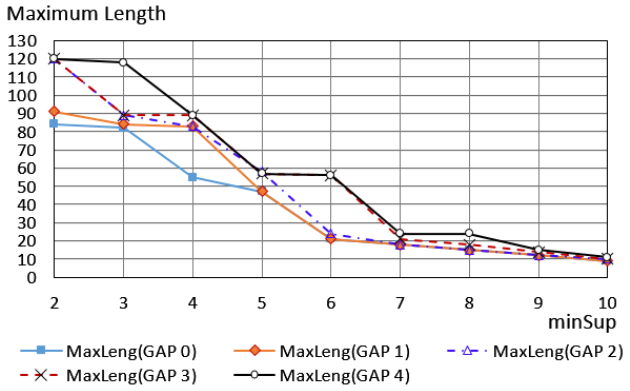


Figure 11. Maximum length of retrieved sequences for each minSup and gap size.

B. *Numbers of Retrieved Sequences*

Figure 12 shows the number of retrieved frequent sequences with respect to gap (0 to 4) and minSup (2 to 10). As expected, the number of retrieved frequent sequences increases as gap increases and minSup decreases. The proposed algorithm can find frequent sequences that occur at least twice in the sequence database, which is necessary for finding all possible code clones. Note that the numbers of retrieved frequent sequences for a gap of 0 are plotted on the right secondary axis because they are 1/7 to 1/60 of the numbers of retrieved frequent sequences for a gap of 1 to 4.

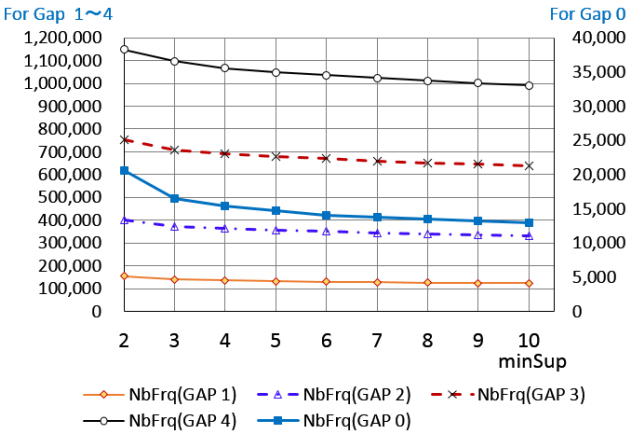


Figure 12. Numbers of retrieved frequent sequences (gap size, 0 and 1-4; minSup, 2-10).

Figure 13 shows the number of maximal retrieved frequent sequences with respect to a gap of 0 to 4 and minSup of 2 to 10. As expected, the number of maximal retrieved frequent sequences is a compact representation of the set of frequent sequences, which is approximately one to two orders of magnitude smaller than that of frequent sequences. The ratio of the number of frequent sequences to the number of maximal frequent sequences increases as gap increases. For example, the ratio is approximately 100, which is the largest obtained ratio, when gap is 4 and minSup is 2.

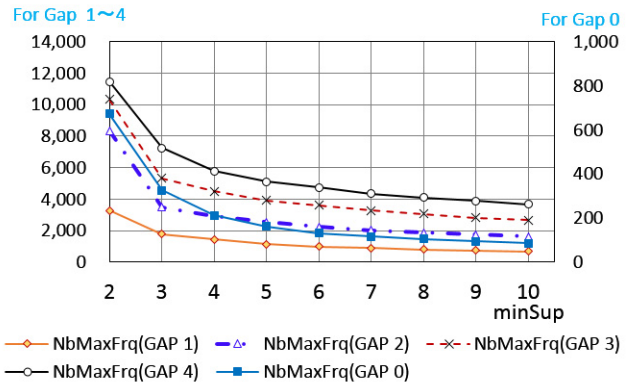


Figure 13. Numbers of retrieved maximal frequent sequences (gap size, 0 and 1-4; minSup, 2-10).

C. *Time Analysis*

Figure 14 shows the elapsed time in milliseconds for retrieving frequent sequences. The x-axis indicates minSup. Note that the elapsed time for a gap of 0 is plotted on the right secondary axis. We measured elapsed time using the following experimental environment.

CPU: Intel Core i3-540 3.07 GHz
Main memory: 8 GB
OS: Windows 7 64 Bit
Programming Language: Java 1.7.0

The proposed algorithm can retrieve frequent sequences fairly efficiently. For example, it takes 289,481 milliseconds to identify 154,789 frequent sequences for a gap of 1 and minSup of 2. Note that elapsed time increases as the gap increases. The results show that the elapsed time is approximately 4.8 × N × t, where N is the number of gaps, and t is the elapsed time for a gap of 0.

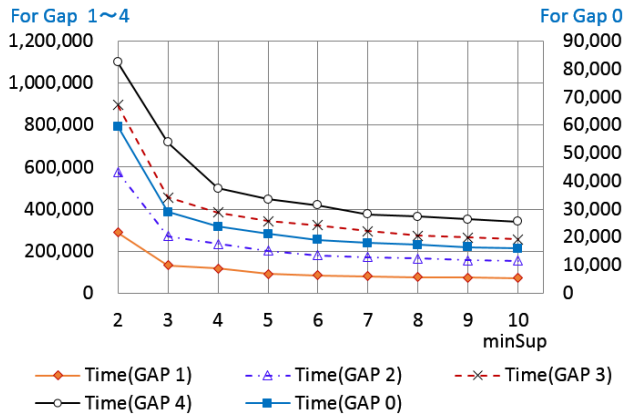


Figure 14. Elapsed time (milliseconds) for retrieving frequent sequences.

 ISBN: 978-1-61208-458-9

The maximal sequential pattern (MaxSP) and vertical mining of maximal sequential patterns (VMSP) produce a set of maximal sequential patterns. However, the runnable code of MaxSP downloaded from an open-source data mining library [9] fails to process the sequence database due to an overly long process time. For VMSP, it processes in less than 100 ms with a very small set of maximal sequential patterns that is approximately three orders of magnitude smaller than the expected set of patterns.

Figure 15 shows a comparison of elapsed time for the proposed algorithm with respect to gap 0, as well as ClaSP and CM-ClaSP, which produce the best available results. Note that ClaSP and CM-ClaSP terminate with a stack overflow error when minSup is less than 7.

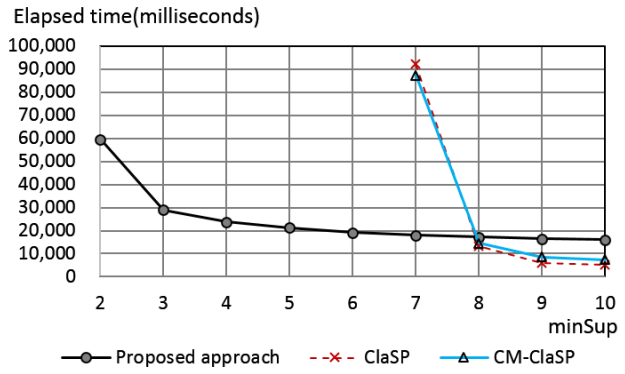


Figure 15. Comparison of elapsed time.

Figure 16 shows elapsed time in milliseconds for retrieving maximal frequent sequences. The input is a list of retrieved frequent sequences, and the output is a list of maximal frequent sequences. The elapsed time is nearly proportional to the number of maximal retrieved frequent sequences in Figure 13. The elapsed time for extracting a list of maximal frequent sequences is 1/160 of that for retrieving a set of frequent sequences and is nearly independent of gap size.

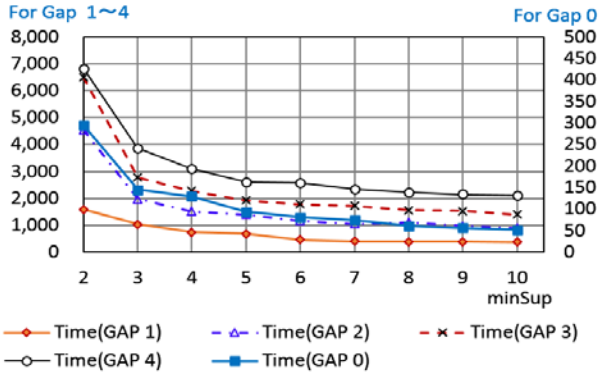


Figure 16. Elapsed time for retrieving maximal frequent sequences.

D. *Source Code Findings*

By increasing gap size to greater than 2, we can relax the gap constraint; however, this is detrimental to the relevance of retrieved sequence occurrences. We limit ourselves to a gap size of 1 to simplify analysis.

1	StringDecoder::decode(byte[] ba, int off, int len)→001→13V→005→004→003→005→004→003→00C→140→141→142→00V→143→005→144→003→145→005→144→003→003→011→07F→003→004→003→003
2	StringDecoder::decode(Charset cs, byte[] ba, int off, int len)→001→13T→13V→005→004→003→005→005→0E0→003→003→14B→005→004→003→00C→141→142→00V→143→005→144→003→145→005→144→003→003→011→07F→003→004→003→003
3	StringEncoder::encode(char[] ca, int off, int len)→001→13V→005→004→003→005→004→003→00C→14F→141→142→00V→14G→005→144→003→14H→005→144→003→003→011→07F→003→004→003→003
4	StringEncoder::encode(Charset cs, char[] ca, int off, int len)→001→14E→13V→005→004→003→005→005→0E0→003→003→14J→005→004→003→00C→141→142→00V→14G→005→144→003→14H→005→144→003→003→011→07F→003→004→003→003

Figure 17. Four clone candidate methods.

Figure 17 shows a set of four methods that match a sequence 005 → 004 → 003 → 00C → 141 → 142 → 00V → within a gap of 1. These four methods are defined in the *StringCoding.java* file of the *java.lang* package. They are considered clones because the arguments in the *encode* and *decode* methods differ only slightly in order to implement method overloading. In addition, they share the sequence 005→004→003→00C→141→142→00V→. Note that only the *encode(char[] ca, int off, int len)* method (Figure 2) matches the sequence 005→004→003→00C→141→142→00V→ with one gap (i.e., "14F" or *CharsetEncoder.reset()*), as shown in the third row of Figure 17. This difference is considered to be worthy of checking by the implementer of the methods.

V. CONCLUSINONS AND FUTURE WORK

We have presented an approach to identify type-3 clones using a source code parsing technique to extract matching code statements, a maximal frequent sequence mining algorithm, and a modified LCS algorithm for computing the matching degree and gaps of corresponding code segments.

Early experiments using the *Java SDK 1.7.0.45 lang* package indicate that the algorithms can identify type-3 clones in a reasonable elapsed time. The experimental results show that the ratio of the number of the frequent sequences to the number of maximal frequent sequences reaches approximately 100. However, the proposed algorithm still generates thousands of maximal frequent sequences. Therefore, we plan to improve the proposed algorithm in future.

 ISBN: 978-1-61208-458-9

REFERENCES

[1] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," Queen's Technical Report:541. Queen's Uni-versity at Kingston, Ontario, Canada, Sep. 2007, pp.1-115.

[2] S. Ducasse, O. Nierstrasz, and M. Rieger, "On the effectiveness of clone detection by string matching," Journal of Software Maintenance and Evolution Research And Practice, Jan. 2006, pp.37–58.

[3] C. K. Roy and J. R. Cordy, "NICAD: Accurate detection of near-miss intentional clons using flexible pretty-printing and code normalization," Proc. 16th IEEE International Conference on Program Comprehension, June 2008, pp.172-181.

[4] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," Comm. ACM, Vol.20, Issue.5, May 1977, pp.350-353.

[5] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Gapped code clone detection with lightweight source code analysis," May 2013, pp.93-102.

[6] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: A tool for finding copy-paste and related bugs in operating system code," Proceedings of the 6th Symposium on Operating System Design and Implementation, Dec, 2004, pp.289-302.

[7] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining closed sequential patterns in large datasets," Proc. 3rd SIAM International Conference on Data Mining (SDM'03), May, 2003, pp.166-177.

[8] S. Negara, M. Codoban, D. Dig, and R. E. Johnson: Mining Fine-Grained Code Changes to Detect Unknown Change Patterns, Proc. 36th International Conference on Software Engineering(ICSE 2014), May 2014, pp.803-813.

[9] "An Open-Source Data Mining Library," http://www.philippe-fournier-viger.com/spmf/index.php, v0.96r20, August 2015.

[10] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng, "VMSP: Efficient vertical mining of maximal sequential patterns," Proc. 27th Canadian Conference on Artificial Intelligence (AI 2014), Springer, Advances in Artificial Intelligence, Lecture Notes in Computer Science,Vol. 8436, May 2014, pp. 83-94.

[11] Y. Udagawa, "A novel technique for retrieving source code duplication," Proc. 9th International Conference on Systems (ICONS 2014), Vol. 9, Feb. 2014, pp.172-177.

[12] R. Agrawal and R. Srikant, "Fast algorithms for mining associa-tion rules," Proc. 20th International Conference on Very Large Data Bases(VLDB), 1994, pp.487-499.

 ISBN: 978-1-61208-458-9

A Multi-Agent System for Expertise Localization in Software Development

José Ramón Martínez García[1], Ramón René Palacio Cinco[2], Joel Antonio Trejo-Sánchez[3]
Luis-Felipe Rodríguez[1], Joaquin Cortez[1]

[1]Instituto Tecnológico de Sonora, Unidad Nainari,
Antonio Caso 2266, Ciudad Obregón, Sonora, México.
e-mail:joseramonmg26@gmail.com, luis.rodriguez@itson.edu.mx
joaquin.cortez@itson.edu.mx

[2]Instituto Tecnológico de Sonora, Unidad Navojoa,
Ramón Corona S/N, Colonia ITSON, Navojoa Sonora, México.
e-mail:ramon.palacio@itson.edu.mx

[3]Universidad del Caribe, Dpto. Ciencias Básicas e Ingeniería,
Esquina Fraccionamiento Tabachines, Cancún, Q.R.
e-mail:jtrejo@ucaribe.edu.mx

***Abstract*— In software industry, *expertise* is fundamental to the timely ending of projects, since tasks are solved in a more efficient manner. The *expertise* of an organization exists in people or artifacts and it is not easy to identify when required because this information does not reside in a repository to facilitate its management in terms of storage, consultation and distribution. This causes uncertainty among members of the organization to determine the appropriate *expertise* to solve a project activity. The aim of this paper is to present a multi-agent system that supports the *expertise* location in software development. Using a knowledge flow methodology the barriers that prevent the flow of knowledge and interaction in software development activities were identified and using this information the requirements were elicited. The architecture provides information concerning the location of the appropriate *expertise* to solve a problem posed by a user making use of the artifacts and experts available in the organization.**

Keywords-knowledge; expertise; agents; software development.

I. INTRODUCTION

Today, organizations have a special interest in treating knowledge as an organizational resource. This type of resource represents a change regarding how to manage information that generates different processes inside the organizations. Using this information, organizations promote the exchange of knowledge among its members, so that through the current knowledge, it will increase and improve their work practices in order to create organizational knowledge. For this, organizations need to promote internal support among members to generate and transfer knowledge that will later help with better decision making [1].

According to [2], knowledge can be found in persons (individual or group) and artifacts (e.g., business practices and daily routines, technologies, physical or digital documents repositories, such as books, manuals, videos, and so on) by which it can produce wealth, multiply the production of physical goods, and create competitive advantage [3][4]. New knowledge always stems from an individual in the organization, so that knowledge passed from individual to organizational. This means, an organization cannot create knowledge without an individual initiative and the interaction that occurs within teams. Knowledge can increase or consolidate in the group through dialogue, discussion, exchange of experience and observation. So the members of an organization generate new points of view through dialogue and discussion. This dialogue can include considerable conflicts and disagreements, but it is precisely such conflict that pushes employees to question existing premises and give new meaning to their experiences [5].

In many situations, when developers encounter difficulties with an activity, they usually searches for knowledge. This is because the goal is to find the *expertise* (better quality knowledge) to solve the problem or the difficulty, demanding a greater degree of knowledge [6]. This article addresses the problem to locate the appropriate *expertise* to solve problems in an organization, particularly in software development. This is, because the software industry generates different type of artifacts during the development process (e.g., system requirements, modules, components software, manuals, etc.) and such artifacts are connected or related to the creator/s (programmer, software architect, analyst, etc.) [7]. During the activities of software development, project activities are distributed to members of the working group and they work individually, at a certain times come together to integrate their products to achieve the project deliverables or artifacts. For this, they require existing *expertise* in the organization so they turn to their colleagues to share knowledge in order to solve obstacles. So, the challenge for these organizations is to obtain adequate *expertise* in a timely manner in order to maintain the competition level of the company to win more contracts and fulfill their commitments on time with customers, and for this the organization requires its members to be effective in generating artifacts [8].

In software development, a lot of knowledge can be shared among members of development either between analysts and programmers or between programmers and testers, but much of this knowledge remains tacit and depends heavily on the interaction between members to

 ISBN: 978-1-61208-458-9

obtain or share that knowledge. This, combined with the global trend of the software industry [9] placing the different team members working in different geographical locations, limited to communication [10] and activities coordination [11], which is reflected in misunderstandings, errors and waste of resources [12]. Therefore, the main objective of this article is to design and model an agent based system to support the localization of *expertise* in the development of software architecture, which makes use of existing knowledge and experience, so that, in this way, they can anticipate problems, innovate and generate new knowledge between the working groups and could potentially help these organizations to improve their development process. The remainder of the paper is organized as follows Section II outlines the related work, after which, in Section III, the phases of the methodology used in this work are described. Section IV explains the multi-agent architectural propposal and describes the process of *expertise* location. Finally, Section V presents our concluding remarks and the directions of our future work.

II. RELATED WORK

The location of *expertise* in software development is a complex and little discussed issue, since there is a tacit knowledge in people. In order to acess that knowledge, software engineers generally interact with each other to find out who is the best person to answer any questions or help with a project task. For this, efforts have been made to manage the location of *expertise* by using an agent-oriented approach, as in [13], where a multi-agent architecture designed to manage information and knowledge generated during maintenance of software is presented, using Web technologies to support interaction between players using different reasoning techniques to generate new knowledge from prior information and learning from their own experience. This reasoning is based on the different cases that happened in a previous project or previously in the same project, as well as the experience documented by the stakeholders. Another proposal that works with the agent-oriented paradigm [14] presents an initial implementation of a system that works with agents that help localization *expertise* (experts) under the theme 'Java programming language', where agents are responsible for scheduling appointments for the exchange of knowledge, proactively detecting when help is needed, providing additional information during the interaction and adjusting their own mechanisms of profiles according to user feedback.

Systems that focus to a particular kind of knowledge (artifacts) have also been developed, such as BluePrint [15], which describes the design, implementation and evaluation of a web search interface integrated with the development environment Adobe Flex Builder that helps users locate code examples of previous projects using keywords (e.g., programming language, framework, class name and/ or method). SNIFF [16] works with the location of artifacts, which facilitates the search for existing libraries through a plugin for the Eclipse development environment for Java programming language. It is based on the premise that libraries are fine match documented facilitating the search for the library with the available domain Java programming. Similarly, Exemplar [17] is a tool for searching software projects of great importance for source code reuse. It uses the keywords and the words in the description of the project to infer on the needs of the user.

Finally, there are many works about locating the knowledge of the people (experts), such as QuME [18], which is a prototype of a personalized Web interface for users of online communities requiring help with Java programming language. It has a mechanism to infer the level of knowledge of a java programming language user, calculated using parameters such as the questions being asked in the forum, the response frequency, the keywords in the user profile and other aspects that help determine the level of *expertise* of person. Expertise Recommender [19] is an experts recommendation system using a general recommendation architecture based on a study of location field experience, which is adjusted according to the needs of the user and the field of the related experts. This work has led to locate the level of knowledge of people based on specific parameters and the history of knowledge that has been shared in a specialized forum.

This section shows how researchers have made many efforts to locate the *expertise* in software development environments. These studies have suggested reasoning to generate new knowledge from a knowledge base. They have also proposed software platforms to locate artifacts in development projects and systems to allow the location of experts on specific topics. However, none of the works we studied has integrated artifacts and experts. These works are limited to collecting the knowledge for use at a particular time, without sharing it, so that no one else can access it later or find out who is the supplier. The most cited papers collect the individual *expertise* of users but do not integrate and share it to make it accessible to all developers in an organization. This is a key element, since software development experience is an important factor for on-time deliveries, as highlighted in [2]. It is important to support the reuse of organizational knowledge [6].

III. KNOWLEDGE FLOW IDENTIFICATION

To identify barriers to the flow of knowledge and interaction in software development activities KoFI Methodology (Knowledge Flow Identification) [20][21] was used. This methodology consists of four phases. Phase 1 consists in the identification of different sources that generated or stored knowledge; Phase 2 identifies the types of knowledge used and generated in the main processes of the organization, while Phase 3 identifies how knowledge flows within the organization. Finally, Phase 4 consists in identifying the main problems that hinder the flow of this knowledge.

This methodology identified the sources of knowledge, knowledge entities that are used in the activities of these organizations, how knowledge is distributed and the problems that arise during the activities of the members in order to solve their doubts or complete their deliverables. This methodology helps the elicitation of the requirements

 ISBN: 978-1-61208-458-9

to support the *expertise* location in the software development.

A. Phase 1: Knowledge Sources

The sources of knowledge that exist in software development organizations were validated through a focus group. This focus group was aimed at knowing how workers of software development perform *expertise* search and how they manage their knowledge, in order to validate whether the sources of knowledge that are listed in the software engineering literature are consistent with the daily developer activity. The focus group was held on the premises of the software company UXLAB [22] and lasted 90 minutes. The group involved 11 members of the organization, 7 developers and 4 designers. The focus group subjects were asked how they obtain knowledge individually, where they stored it, how they transfer it and how they infer the level of *expertise* of a colleague. The result of this phase is summarized in Table I.

TABLE I. SOURCE KNOWLEDGE

Sources	Description
Artifacts	*Bookmarks:* When developers find a place of interest or it helped them to solve a problem, they usually save the address in the browser.
	Projects: Developers usually save previous projects source code for reuse in some cases.
	Manuals: In some cases the acquired knowledge is stored as a user manual.
	Official documentation: In some cases, developers often visit the official site of some technology to obtain information about it.
Persons	Developers often seek experts, people with some degree of knowledge of a specific topic or area that could be particularly useful to solve some problems presented by development activities, so it involves all people within the organization represent a source of knowledge.

B. Phase 2: Knowledge Topics

In order to categorize and organize knowledge identified in software development organizations a plot showing the three scenarios in which the *expertise* is involved (see Table II) was performed.

The scenario of *Individual Knowledge* is characterized by a *knowledge necessity,* this consists of a developer looking for information on forums, official documentation and in his own bookmarks. Once the developer finds useful information, the *transition* (*tacit to explicit*) starts. This happens when the developer stores this information.

The scenario *Knowledge Exchange* is characterized by the way knowledge is transfered, which can be obtained through training *directly* with an expert (e.g., advice and/or training) or *indirectly* performed by using some explicit action recommended by experts (e.g., manuals, specialized forums, etc.).

Finally, the scenario *Expert Search* relates to the identification of the experience and knowledge that colleagues have about a particular query to acquire new knowledge or solve an obstacle concerning any work activity. Once a developer finds an expert, they exchange *Knowledge Exchange*.

TABLE II. KNOWLEDGE REPRESENTATION

Scenario	Characteristics	Representation
Individual Knowledge	Knowledge necessity	Forums
		Official Documentation
		Video Tutorials
		Bookmarks
		Projects
	Transition (tacit to explicit)	Format
		Author
		Thema
		Folder name
		Notes
		Language
		Source
Knowledge Exchange	Direct	Training courses
		Advisery
	Indirect	Keywords
		Bookmarks
		Web sites
		Manuals
Expert Search	Expertise Level	Experience
		Work area
		Availability
		Programming languages
		Recommendations
		Contributions

C. Phase 3: Knowledge Flow

The flow of knowledge to locate *expertise* starts with a developer that needs to acquire new knowledge to solve any difficulties in an activity. Then he/she choose between doing a search in artifacts or search for an expert.

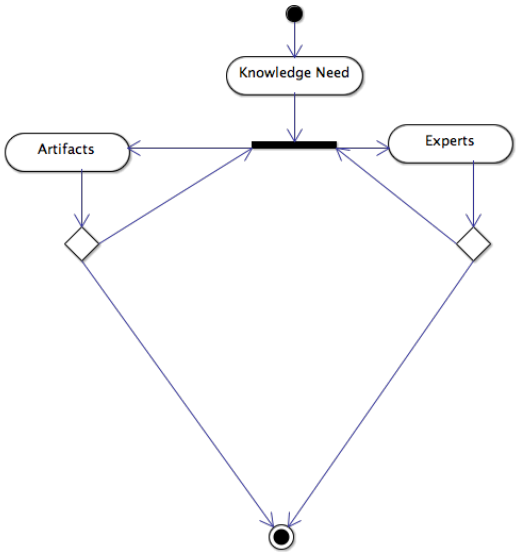


Figure 1. Knowledge Flow Activity Diagram.

 ISBN: 978-1-61208-458-9

In the case of choosing a search of artifacts, searches should be performed in all available artifacts (pages, manuals, videos, reused code). When an artifact is found it is checked whether this helped achieve the objective or still need to find more artifacts. If not necessary to search for more artifacts then the objective was met, otherwise, the search for more artifacts continues. An artifact may suggest to seek expert help. In the case of choosing an expert search, start looking for experts with the degree of knowledge to solve the difficulty presented in the activity. Once an expert is found, one needs to check their availability to start an interaction with the expert. Later, one must check if expert consultation was sufficient or if there is a need to consult another expert. It may also happen that an expert suggests the consultation of an artifact. If the objective was fulfilled, the process ends (see Figure 1).

D. Phase 4: Problems in the Knowledge Flow

With the information obtained from the focus group, it was possible to identify the problems that prevent the distribution of knowledge in the activities of support software localization *expertise*. These are:

1) *Administration of the artifacts (individual or group):* In some cases, the knowledge supplier is known, but one does not have access to its artifacts (e.g., blogs, manuals, reused code).
2) *Management Experts:* Sometimes it is difficult to find the person with the appropriate level of *expertise* to consult if there is any doubt on how to solve a problem or to perform an activity.
3) *Availability of the Experts:* In some cases, it is not known if the *expertise* or expert is available to the person who needs them.
4) *Timely resolution of difficulties*: In some cases, a lot of time is wasted in finding *expertise* that does not have the knowledge needed to solve an issue.

IV. SUPPORTING EXPERTISE LOCATION

With the information obtained from KoFI methodology, it was possible to elicit the system requirements for the *expertise* location (see Table III). With these requirements, a model of a multi-agent architecture is needed. This multi-agent system is composed of virtual modules each dedicated to managing artifacts and experts within the organization.

TABLE III. EXPERTISE LOCATION REQUIREMENTS

Requirements
R1. The system will have an interface where an information query can be performed and also new knowledge can be registered.
R2. The system is responsible for capturing new knowledge repositories and will perform searches of users.
R3. The system is responsible for making decisions to find the best resources for the user according to their needs.
R4. The system is responsible for making the calculation of potential experts that have the knowledge and availability to provide knowledge to the user.

The purpose of ExLoc (Expertise Location) system is to provide appropriate resources to the users needs, collect all the knowledge that is available within the organization, provide contact information with experts who can help solve some problems. There are 4 basic ExLoc agents: User Agent (UA), Central Search Agent (CSA) and Fuzzy Expert Agent (FEA). Working together for *expertise* location, either to capture or search (see Figure 2).

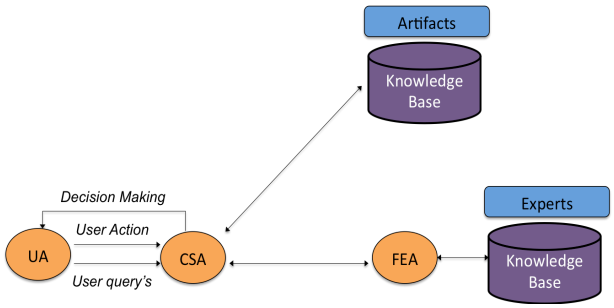


Figure 2. ExLoc Architecture.

The CSA Agent receives the search parameters sent by the UA Agent. For example, when a user needs to know *Web backend* programming, in *PHP* language to make a *database connection* in *MYSQL*, this parameters trigger a set of rules to identify artifacts or experts who have knowledge related to the search. Later the results of this process are sent to the UA agent presented as a list of reources to the user. The CSA Agent sends a series of actions that the user can perform to satisfy their need for knowledge. At the end of this process the systems ask the user if such knowledge was useful for him, using a ranking score to mark the usefulness of the resource in the case of those who have been helpful to the user. The results of this questions are used to update the knowledge repositories.

A. User Agent (UA)

The main objective of User Agent (UA) is to provide the user an interface that allows to search *expertise*, as well as to capture new knowledge.

Task (*R1*)

- Receive data capture of the new knowledge
- Perform Search of *expertise*
- Send the information to the Central Search Agent (CSA)
- Update repositories

There is an UA which interacts with each user allowing customization of searches through the interest or the history of users. Each UA communicates with the CSA, so that, it sends alerts or actions performed by the user and also receives the information obtained as a result of these actions and later shows it to the user.

B. Central Search Agent (CSA)

The main objective of CSA is to act as an expert in *expertise* location strategies in software development.

 ISBN: 978-1-61208-458-9

Task (*R3*)

- Decision making
- Run the search or capture
- Distribute knowledge

There is a CSA around the multi-agent system responsible for processing all actions and inferring what is the best option according to user needs.

The CSA Agent receives as input the user query. According to the need of the user, the CSA Agent creates a strategy to search in the knowledge base (KB) of artifacts and experts for the right tools to support the user. The CSA has an *inference engine* that interacts with the artifacts and experts KB. Next we describe the inference engine.

The knowledge base (KB) contains all the knowledge of the artifacts and the experts respectively. The system starts with the knowledge provided by the registered users. Through the use of such knowledge, this will increment. The semantic network of Figure 3 represents the knowledge. The expert system can search the *expertise* in any of two knowledge repositories: the artifacts contain knowledge from a number of tutorials, papers, web sites and forums. Such knowledge is classified as in the previous example where the platform was *web*, type *backend,* language *PHP* and the subject was a database *connection* in *MYSQL*. Then, the classification of the expert knowledge includes the *project* where the expert is currently working, *experience* of topics the expert has been working on (e.g., Android Projects, Web pages etc.), the *schedule* of the user to know if the expert is available and the *personal data*.

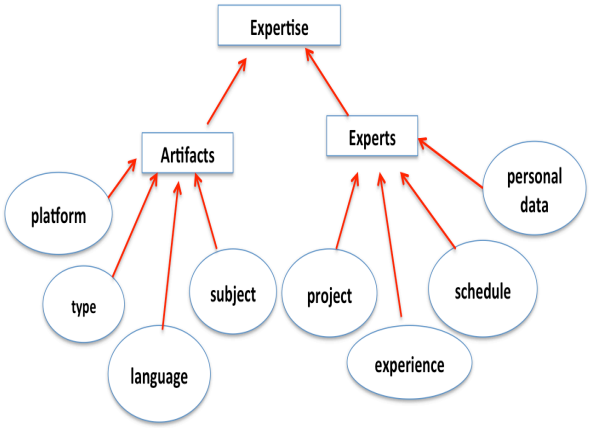


Figure 3. Semantic Network of the Knowledge Representation

The inference engine is able to explore through all the knowledge from artifacts and experts.

The inference engine begins the decision making by exploring the user query input and comparing it with the KB of artifacts and with the available experts. Figure 4 presents the inference engine implicit in the CSA agent. The user introduces the input query to the CSA. The input query is processed by the inference engine and returns a suggestion to the CSA. The inference engine uses the *expertise* stored in the knowledge base to support in the decision making process.

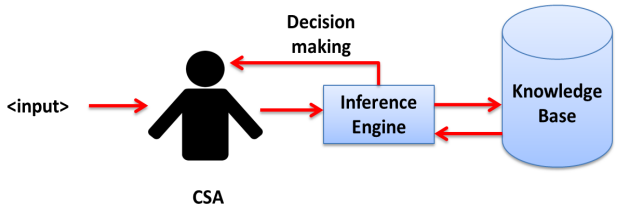


Figure 4. Inference Engine

C. *Fuzzy Expert Agent (FEA)*

FEA has the main objective to make the calculation of available experts who could help the user based on the requirements of the problem.

Task (*R4*)

- Finding the right persons according the user needs

There is a FEA responsible for calculations of the experts using the repository with information from members of the company. Later, candidates found are sent to CSA.

V. CONCLUSION & FUTURE WORK

This article addresses the issue of *expertise* location in the software development with a multi-agent system that makes use of the experts and artifacts available in the organization. Because of that, both artifacts and experts play an important role, since, in some cases, there is not enough knowledge stored in the system, but some expert may be able to help and an artifact can lead to such an expert. The requirements of the system were elicited with the information obtained from the KoFI methodology. This methodology was used to know the sources that are located in the software development software, and the topics or the way they store and transfer. Then, with this information, the knowledge flow was built, to identify the problems that the developer currently faces.

As future work, we intend to work in the semantic network presented as the basis of a language through wich the agents will comunicate. Also, we intend to extend the functionality of CSA and FEA to interact with a wider range of knowledge and perform the validation of ExLoc in the work context of software development.

REFERENCES

[1] E. Pasher and T. Ronen, The Complete Guide to Knowledge Management: a Strategic Plan to Leverage Your Company's Intellectual Capital, John Wiley & Sons, 2011.

[2] I. Becerra-Fernandez and R. Sabherwal, Knowledge management: systems and processes, ME Sharpe, 2010.

[3] E. Fragouli, "Intellectual Capital & Organizational Advantage: an economic approach to its valuation and measurement," Business and Management, vol. 7, no. 1, 2015, pp. 36-57.

 ISBN: 978-1-61208-458-9

[4] N. Bontis, “Assessing knowledge assets: a review of the models used to measure intellectual capital,” International Journal of Management reviews, vol. 3, no. 1, 2001, pp. 41-60.

[5] I. Nonaka and H. Takeuchi, The Knowledge-creating Company: How Japanese Companies Create the Dynamics of Innovation, Oxford University Press, 1995.

[6] K. A. Ericsson, M. J. Prietula and E. T. Cokely, “The Making of an Expert,” Harvard Business Review, vol. 85, no. 7/8, 2007, pp. 114.

[7] R. S. Pressman, Software Engineering: A Practitioner’s Approach, 7/e, Mc Graw-Hill, 2009.

[8] I. Rus and M. Lindvall, “Guest editors' introduction: Knowledge Management in Software Engineering,” IEEE software, vol. 19, no. 3, 2002, pp. 26-38.

[9] D. Damian and D. Moitra, “Guest Editors' Introduction: Global Software Development: How Far Have We Come?,” Software, IEEE, vol. 23, no. 5, 2006, pp. 17-19.

[10] R. E. Jensen, “Communication Breakdowns in Global Software Development Teams: Is Knowledge Creation the Answer?,” Proc. 17th ACM International Conference on Supporting Group Work, ACM, 2012, pp. 289-290.

[11] J. D. Herbsleb and D. Moitra, “Global Software Development,” Software, IEEE, vol. 18, no. 2, 2001, pp. 16-20.

[12] H. Van Vliet, “Knowledge Sharing in Software Development,” Proc. 10th International Conference on Quality Software (QSIC), IEEE, 2010, pp. 2-2.

[13] O. M. Rodríguez, A. Vizcaino, A. I. Martínez, “Using a Multi-agent Architecture to Manage Knowledge in the Software Maintenance Process,” Knowledge-Based Intelligent Information and Engineering Systems: 8th International Conference, KES 2004, Wellington, New Zealand, September 20-25, 2004, Proc., Part I, M. G. Negoita, et al., eds., Springer Berlin Heidelberg, 2004, pp. 1181-1188.

[14] A. Vivacqua, “Agents for Expertise Location,” Proc. AAAI Spring Symposium Workshop on Intelligent Agents in Cyberspace, 1999, pp. 9-13.

[15] J. Brandt, M. Dontcheva, M. Weskamp and S. R. Klemmer, “Example-Centric Programming: Integrating Web Search into the Development Environment,” Proc. SIGCHI Conference on Human Factors in Computing Systems, ACM, 2010, pp. 513-522.

[16] S. Chatterjee, S. Juvekar and K. Sen, “Sniff: A Search Engine for Java Using Free-Form Queries,” Fundamental Approaches to Software Engineering, Springer, 2009, pp. 385-400.

[17] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk and C. Cumby, “A Search Engine For Finding Highly Relevant Applications,” Proc. Software Engineering, 2010 ACM/IEEE 32nd International Conference on, IEEE, 2010, pp. 475-484.

[18] J. Zhang, M. S. Ackerman, L. Adamic, K. K. Nam, “QuME: A Mechanism to Support Expertise Finding in Online Help-seeking Communities,” Proc. 20th annual ACM Symposium on User Interface Software and Technology, ACM, 2007, pp. 111-114.

[19] D.W. McDonald and M.S. Ackerman, “Expertise Recommender: A Flexible Recommendation System and Architecture,” Proc. ACM Conference on Computer Supported Cooperative Work, 2000, pp. 231-240.

[20] O. M. Rodríguez-Elias, A. Vizcaíno, A. I. Martínez-García, J. Favela and M. Piattini, “Knowledge Flow Identification,” Encyclopedia of Information Science and Technology, 2009, pp. 2337-2342.

[21] O. M. Rodríguez-Elias, A. Vizcaíno, A. I. Martínez-García, J. Favela and M. Piattini, “Studying Knowledge Flows in Software Process,” Software Engineering and Development, Nova Publishers, 2009, pp. 37-68.

[22] UXLAB, 2015; http://www.uxlab.mx/

 ISBN: 978-1-61208-458-9

Exploring the Scala Macro System for Compile Time Model-Based Generation of Statically Type-Safe REST Services

Filipe R. R. Oliveira, Hugo Sereno Ferreira, and Tiago Boldt Sousa

Department of Informatics Engineering
Faculty of Engineering, University of Porto, Portugal
Email: {filipe.rroliveira, hugo.sereno, tiago.boldt}@fe.up.pt

***Abstract*—Representational State Transfer (REST) is a prolific architectural style among modern Web services, mainly due to its better performance, scalability and simplicity. A common usage of the style includes services that implement CRUD (Create, Read, Update, and Delete) operations for entities of a model. Most frameworks that do this automatically, apply the models logic in run-time usually using reflection or in-memory data structures, and are written in dynamically typed programming languages. Such technical choices usually hinder two software quality attributes: performance (due to run-time adaptation) and maintainability (due to absence of compile-time guarantees). This paper studies the impact of interpreting the models at compile-time with statically typed programming languages, using Scala as a representative. Based on a generic architecture, we implemented a proof of concept, called the Metamorphic framework, which uses a Domain Specific Language (DSL), supported by a macro system, to generate entire applications. Evaluation was executed by performing both quantitative benchmarks and qualitative analysis of Metamorphic against other frameworks.**

Keywords–Model-Driven Engineering; REST; Internal DSL; Scala Macros.

I. INTRODUCTION

The number of Internet users has tripled in the last decade [1] mainly due to the appearance and growth of mobile devices. These users and devices stay connected and explore their potentialities through the consumption of Web services, such as, static or dynamic Web pages, mobile applications content, and real-time services. Two common architectures for implementing these services were the Remote Procedure Call (RPC) and the Service-Oriented Architecture (SOA) [2], mainly explored through the Simple Object Access Protocol (SOAP).

In the same time-frame of SOAP's specification, Roy Fielding defined the Representational State Transfer (REST) architectural style [3] to be applied in distributed hypermedia systems. The style defines a set of six constraints: client-server, stateless, cache, uniform interface, layered system, and code-on-demand. The uniform interface constraint uses the concept of resource, around which communication is built. A resource is an abstract instance of any concept that can be uniquely identified. All these constraints enable scalability, portability, visibility, and simplicity in exchange for some degraded efficiency and reliability. It is normally preferred to the SOAP approach as it achieves better performance most of the time [4]. In practice, REST is usually implemented using URI (Uniform Resource Identifier) for resource identification, and HTTP for stateless client-server cacheable communication.

The need to implement more complex and robust Web services led to the development of frameworks, that provide solutions for recurrent problems and enable better structured implementations. Some of these use model-driven engineering [5], i.e., they can deliver CRUD operations for a set of model entities, reducing repeated code when compared with most traditional frameworks. This approach has the following advantages: short-time-to-market, fewer bugs, increased reuse, and easier-to-understand up-to-date documentation [6].

In general, current model-driven REST frameworks do in fact reduce repetition of code but due to implementation decisions there are two main problems.

Firstly, they are mostly implemented in dynamically typed languages [7], such as Python and JavaScript. These kind of languages don't require the use of explicit types, in which case type-related errors are more susceptible to happen. This fact combined with dynamic typechecking delays resolution of these errors to run-time, suggesting longer debugging sessions. Strongly and statically typed languages reduce substantially this problem and consequently may enable better performed services, due to compiler optimizations based on types.

Secondly, these frameworks implement model-based generation through the inspection of variables or introspection [8] for collecting the schema, and through parameterized functions or intercession [8] for responding to requests. All this work is done at run-time, increasing the program's setup time or even the response time to requests.

Following such logic and considering a statically type-safe programming language that enables generation of REST services in compile time, a question can be raised:

> *Can a model-driven REST framework written in that language improve the development process and performance when compared to current ones?*

In this research, Scala [9] was used as proof of concept to answer this question. This language, that was built with scalability in mind, offers a strong static type system, and an easy capacity for compile-time generation, through macros [10]. These characteristics promise that developers may be able to implement their model-driven REST services even faster and with more robustness, as type errors may be identified sooner.

The paper is organized as follows. Firstly, in Section II the most relevant model-driven REST frameworks are briefly presented and compared. Secondly, in Section III, we describe a generic architecture for model-driven REST applications and how that was translated into the Metamorphic framework. Thirdly, in Section IV the validation criteria is defined as well

 ISBN: 978-1-61208-458-9

as the steps executed to verify that, which include framework benchmarks and a synthetic environment experiment. At last, the conclusion of the work is presented followed by future work that can be explored.

II. Model-Driven REST Frameworks

Model-driven REST frameworks usually consider two types of resources: collections of entities which may have create and read operations; and instances of entities which may have update, replace, and delete operations.

A. Django REST Framework

Django REST framework [11] is an open source framework in Python to build Web APIs (Application Program Interfaces), and is built on top of the Django framework [12], a tool that enables fast development of Web applications, including model-driven development. The framework is funded in: *views* that given requests perform necessary actions and prepare responses; *serializers* that define the structure of object's data; and *urls* that connect URLs (Uniform Resource Identifiers) with views.

```
class Category(models.Model):
    name = models.CharField(max_length = 50)
    description = models.CharField(max_length = 100)

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category

class CategoryViewSet(viewsets.ModelViewSet):
    queryset = Category.objects.all()
    serializer_class = CategorySerializer

router = routers.DefaultRouter()
router.register(r'categories', CategoryViewSet)
urlpatterns = router.urls
```

Listing 1. Example of a simple API with the Django REST framework

The support for model-driven development is delivered by subclassing a base model, a base model serializer, and a base generic view, as shown in Listing 1. These subclasses override the interface methods and use reflection in order to implement the intended functionality. This adds an overhead on responses when compared with manual implementations that directly access variables without having to inspect their name in the beginning.

B. Eve

Eve [13] is also an open source framework in Python, and is built on top of the Flask microframework [14] that supports HTTP (Hypertext Transfer Protocol) I/O (Input/Output) operations and routing. In contrast to Django REST that supports four types of SQL (Structured Query Language) databases, Eve only supports non relational MongoDB databases.

```
DOMAIN = {
'categories': {
  'schema': {
    'name': {
      'type': 'string', 'maxlength': 50, 'required': True
    },
    'description': {
      'type': 'string', 'maxlength': 100, 'required': True
    }
  }
}}
```

Listing 2. Example of a simple API with the Eve framework

It is more based in specification rather than writing code requiring only the initialization of the *DOMAIN* variable, as shown in Listing 2.

C. LoopBack

LoopBack [15] is an open source Node.js framework [16], which means that is written in JavaScript. It is built on top of the Express framework [17] that provides a thin layer of Web application features. It considers relations between entities as resources, besides instances and collections of entities.

```
{
  "name": "Category",
  "plural": "categories",
  "base": "PersistedModel",
  "idInjection": true,
  "properties": {
    "name": { "type": "string", "required": true },
    "description": { "type": "string", "required": true }
  },
  "validations": [],
  "relations": {},
  "acls": [],
  "methods": []
}
```

Listing 3. Example of a simple API with the LoopBack framework

The framework tries to hide its inner workings by providing a command-line tool through which model entities are specified. In fact, this tool generates JSON (JavaScript Object Notation) files with the provided specification which may be edited, as shown in Listing 3. When the server application is started the model is interpreted and the correct dispatch functions are dynamically generated, similar to Eve and contrary to Django REST.

D. Sails

Sails [18] is an open source Node.js framework [16], and is also built on top of the Express framework [17] providing a Model-View-Controller (MVC) development architecture. It enables model-driven development by providing entity scaffolding (generation of code templates) using *sails generate api* <*entity_name*>.

```
module.exports = {
  attributes: {
    name: {
      type: "string", maxLength: 50, required: true },
    description: {
      type: "string", maxLength: 100, required: true }}};
```

Listing 4. Example of a simple API with the Sails framework

The developer must then complete the generated files with a specification of the entities, just like in Listing 4. Just like the previous examples the model is only known by the framework in run-time by importing the modules.

E. Conclusion

The identified frameworks are implemented in a dynamically typechecked language, either Python or JavaScript. Each of them implements model-based services with different approaches: class specialization in the case of Django REST; variable initialization in the case of Eve and Sails; and command-line interaction in the case of LoopBack.

 ISBN: 978-1-61208-458-9

III. Proposed Framework

Building high-quality frameworks is usually the result of many design iterations [19] using: either a *bottom-up* approach that starts with concrete applications and iteratively abstracts concepts into the framework; or *top-down* which relies on domain knowledge. The development of Metamorphic used a bottom-up approach. Considering architectures proposed by the online community, we built a non model-driven base application that followed a supposedly ideal architecture. In order to generate such kind of applications, we designed an internal DSL that relied on the use of macros. Macro architecture and tests were designed and implemented iteratively.

A. Application Architecture

Generated applications can either be *synchronous* or *asynchronous* but they have only one architecture (Figure 1). The architecture is not model centered, allowing generation of generic Web applications. This fact assures better software quality as components have to be less decoupled from their real use. Influenced by this decision, the architecture is composed of a mandatory layer, the *application logic*, and an optional layer, *data storage*, which may be used by the first layer.

The application logic is implemented by an *App* object that initiates services. These services may require access to data storage through repositories. A repository must have an entity associated with it and the whole application has access to a set of developer settings.

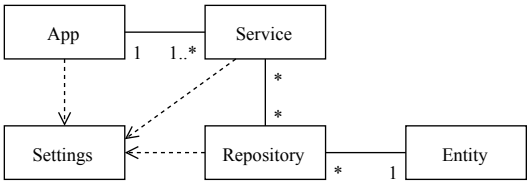


Figure 1. Architecture of a generated application.

To enable greater flexibility of generation, the application logic is defined by a model, as shown in Figure 2. The model allows the specification of services, which may have dependencies and a set of operations. Each operation implements an HTTP method for a path, expects the request body to be serializable for a specified class, and contains a body. It is at the operations level that one of the possible programming styles is applied, by using the *isAsync* flag.

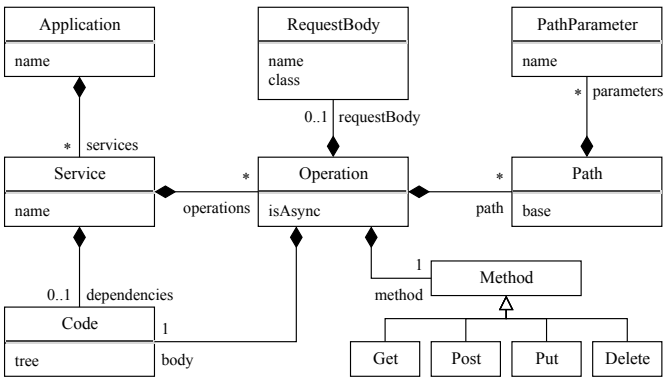


Figure 2. Application logic model.

To enable configurations, the *Config* [20] library was used, which enables the use of one file for setting configurations of all dependencies of a project. This means that besides Metamorphic's configurations, developers can still configure underlying libraries.

```
metamorphic {
  host = "111.111.111.111"
  port = 9000
  databases.default.name = "file.db"
}
```

Listing 5. Example of configuration file with a SQLite database

The configurations (Listing 5) are defined inside the *metamorphic* scope and shall be either host ("localhost" as default), port (8080 as default) or databases. Scopes inside *databases* may specify a name, an user, a password, an host, a port, a number of threads (numThreads) or a maximum queue size (queueSize).

B. Internal DSL

Scala macros enables generation of classes, traits and objects either through type providers or macro annotations [21]. The first discourages reuse of types in the scope calling the macros, while the second despite some limitations allows reuse. Through an internal DSL the framework makes use of these annotations.

Applications are identified by *@app* annotations in objects (Listing 6), which may have a set of entity definitions, a list of default operations, and a set of service definitions.

```
import metamorphic.dsl._
@app object PersonApp {

  @entity class Person {
    def fullname = StringField()
    def birthdate = DateField()
  }

  class PersonService extends EntityService[Person] {
    val operations = List(GetAll)

    def create(person: Person) = {
      if (person.fullname.length < 5)
        Response("Name is too short.", BadRequest)
      else
        super.create(person)
    }
  }
}
```

Listing 6. Example of simple API with the Metamorphic framework

The model specification follows the metametamodel in Figure 3 which is independent of any specification source such as this DSL. In this case entities can be defined using the *@entity* annotation in a class. The macro annotation expansion adds a companion object with replicated content, which helps to identify types errors as the compiler will typecheck the result after expansion. Entities are composed by fields which are case classes that accept a variable number of values to enable configuration and may be of type: *IntegerField*, *DoubleField*, *StringField*, *BooleanField*, *DateField*, *DateTimeField*, *ObjectField*, *ListField*, *ReverseField*.

All fields accept an *Option* argument, meaning that the property is not required. The first argument of a *ListField* or an *ObjectField* is the companion object of an entity definition. These two types of fields are by default mapped to many-to-many and one-to-many relations, respectively. The use of

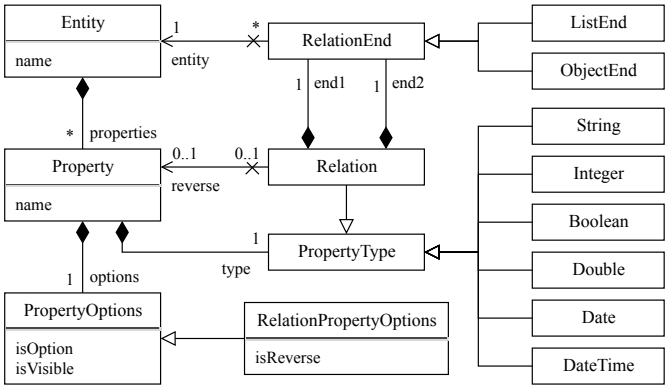


Figure 3. Framework's metametamodel.

R.Object as argument changes this behavior to many-to-one and one-to-one relations, respectively.

The generated entity operations can be the ones identified in Table I, which are implemented using entities plural as the base path. By default, each entity has all operations automatically implemented. Declaring the *operations* variable in the *@app* scope changes the set of default operations to be implemented. Operations are identified using the following objects: *Create*, *GetAll*, *Get*, *Replace*, and *Delete*.

TABLE I. ENTITY OPERATIONS SPECIFICATION

Operation	HTTP method	Path	Success code	Error codes
Create	POST	basePath/	201 (Created)	400 (Bad Request)
GetAll	GET	basePath/	200 (Ok)	-
Get	GET	basePath/:id	200 (Ok)	404 (Not Found)
Replace	PUT	basePath/:id	200 (Ok)	400 (Bad Request), 404 (Not Found)
Delete	DELETE	basePath/:id	204 (No Content)	404 (Not Found)

Changes to the default implemented operations are done via services, which can override the set of default operations for a particular entity. Operations implementations return a *Response* if synchronous or *Future[Response]* if asynchronous and can also be overridden. Customized operations can use a *repository* variable for accessing storage and use the keyword *super* for applying the default implementation.

C. Implementation

The classes and traits required by the DSL were created in the package *metamorphic.dsl* (Figure 4), including the *@app* annotation. The annotation implementation depends on the package *matcher* for translating the code tree into a metametamodel instance and an application logic model instance. There is also a dependency of package *generator* for mapping those models into the final application tree.

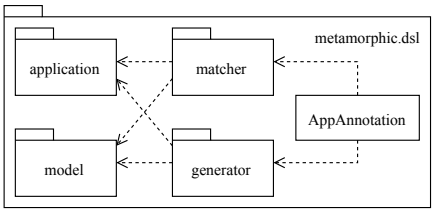


Figure 4. Diagram of packages for the Metamorphic framework.

The framework was designed to have a flexible and loosely decoupled architecture that allows long-term maintainability. With that in mind the *generator* package doesn't provide any concrete generation of applications, building them instead using dependency injection. The framework requires for a project to provide two dependencies/generators: a *repository generator* and a *service generator*.

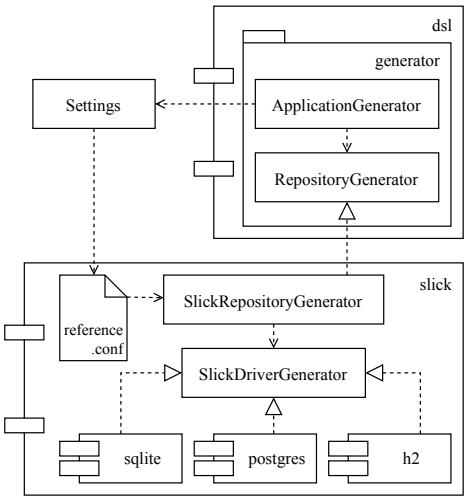


Figure 5. Dependency injection of a RepositoryGenerator that uses Slick.

To test this proof of concept, two repository generators were implemented based in the Slick library [22], one for synchronous applications and the other for asynchronous applications. Figure 5 illustrates how dependency injection is performed for a RepositoryGenerator that uses Slick, which also uses dependency injection to implement different database systems. We also implemented a service generator that uses components from the Spray toolkit [23].

IV. VALIDATION

It was expected that the developed framework would have the following characteristics:

- *Quick and easy to use*. Developers that look for these kind of frameworks want to have a Minimum Viable Product (MVP) as soon as possible without having to explore all the framework's documentation.
- *Error preventive*. Statically typechecked programs reassure developers about their code quality and reduce frustration when debugging.
- *Better response times*. Due to its programming language origin, improvements in performance should be noted when compared with existing model-driven frameworks.

The first and second characteristics were validated using *synthetic environment experiments* [24], in the form of an academic quasi-experiment. The last characteristic was validated through *dynamic analysis* [24], in the form of benchmarks.

A. Benchmarks

Comparing response times of different REST frameworks shall not be generic, i.e., comparison must be executed between services with the same characteristics. The categorization of compared services followed two properties: the type of entities and the type of operations. Entities may be: (i) *simple entities* which don't have navigable relations with other entities; (ii)

 ISBN: 978-1-61208-458-9

entities with objects which have a navigable relation with multiplicity one; (iii) or *entities with lists* which have a navigable relation with an infinite upper bound.

The same scenario was compared using seven different implementations: (i) synchronous version of Metamorphic; (ii) asynchronous version of Metamorphic; (iii) LoopBack; (iv) Sails; (v) Django REST in Python 2.7; (vi) Django REST in Python 3.4; (vii) Eve. All of these used a PostgreSQL 9.3.8 database (except Eve that used MongoDB) and were installed in production environments to guarantee maximum performance.

The experiment was executed in a portable computer Lenovo Thinkpad T430 with the following specifications: Ubuntu 14.04 LTS 32-bit; Intel Core i5-3210M @ 2.5GHzx4; 4GB Sodimm DDR3 Memory (1600 MHz); and 500GB 7200 RPM 32 MB Cache SATA Hard Drive. The tests were locally executed in an environment without Internet connection, all non-essential programs closed, and two terminals in foreground: server application and benchmark application.

All batches of operations were performed with a maximum of 10 concurrent requests. For each tuple, (Framework, Operation, Entity Type) 5000 requests were executed and, to discard any possible setup effects, 1000 requests were executed before testing each framework.

TABLE II. FRAMEWORK'S RANK BY ENTITY TYPE AND RANK SUM

Framework	Create	GetAll	Get	Replace	Delete	Sum
LoopBack	1 | 1 | 1	3 | 2 | 1	1 | 1 | 1	2 | 2 | 2	1 | 1 | 1	21
Metamorphic Async	2 | 2 | 2	1 | 1 | 2	3 | 2 | 2	1 | 1 | 1	2 | 2 | 2	26
Sails	3 | 3 | 6	2 | 6 | 4	2 | 3 | 3	3 | 3 | 6	3 | 3 | 5	55
Django REST 3.4	5 | 5 | 4	5 | 4 | 5	6 | 5 | 5	5 | 4 | 3	4 | 4 | 4	68
Django REST	4 | 4 | 3	6 | 5 | 6	5 | 6 | 6	4 | 5 | 4	5 | 5 | 3	71
Metamorphic	6 | 6 | 5	4 | 3 | 3	4 | 4 | 4	6 | 6 | 5	6 | 6 | 6	74
Eve*	1 | 1 | 1	4 | 3 | 2	4 | 3 | 3	1 | 1 | 2	3 | 3 | 3	35

* pseudo-rank; not comparable.

The frameworks were compared using rank sums. Table II presents the ranking of the frameworks by entity type in the following order: simple entities; entities with objects, and entities with lists. Considering the sum of these rank sums it is possible to conclude that, in spite of not being the best performant framework, the asynchronous version of Metamorphic already achieves performances better than most model-driven frameworks. In fact, without almost no effort to optimize framework's implementation, its results are close to the most performant framework, LoopBack, and it is the best solution to implement GetAll and Replace operations.

B. Academic Quasi-Experiment

8 MSc students in their 5^{th} year of the Master in Informatics and Computing Engineering, from the Faculty of Engineering of the University of Porto, were asked to participate. The experiment tested only one of the current model-driven frameworks against the synchronous version of Metamorphic.

All subjects started by answering a questionnaire and reading a problem guide. The subjects were split in two groups with two different treatments and had to perform the same set of tasks (Round 1). Then, each subject performed the same set of tasks as before with another treatment (Round 2). The test finished by answering another questionnaire.

The treatments were: *baseline treatment* - a default ready-to-use Django REST project; and *experimental treatment* - a default ready-to-use Metamorphic synchronous project. In both treatments, a guide about the framework and the language syntax were handed to the subjects. The experiment consisted in three tasks: (i) modeling using a UML diagram and the entities schema; (ii) creation of services operations for the entities; and (iii) customization of the defined operations.

Each subject executed the test in an isolated area of a low noise laboratory, with a single portable computer with Internet access mimicking a real programming situation. The subjects could only use a text editor of their choice and clarify any doubts they had. Application running and testing had to be done using the terminal. A *screencast* program was used to correctly measure time and development metrics in a non-intrusive way.

The questionnaires were designed with a five-point Likert scale [25]. Their responses were compared using the non-parametric, two-sample, rank-sum Wilcoxon-Mann-Whitney [26] test, with $n1 = n2 = 4$ and significance level of 5%. The results revealed statistical validity of the experiment and that implementing model-based operations is easier and more intuitive to do using Metamorphic ($\rho = 0.014$ in both rounds). Also, there is an high chance that implementations of models and customizations are easier and more intuitive ($\rho_1 = 0.043, \rho_2 = 0.200$ and $\rho_1 = 0.014, \rho_2 = 0.100$ respectively).

The framework may be considered quicker to use as time measurements indicate that development time may decrease 35%, and lines of code measurements indicate that the quantity of code may decrease 28%. As can be seen in Table III, modeling (Task 1) and operations definition (Task 2) may be implemented faster even after having knowledge about the problem (Round 2). The results of Task 3 were unexpected and may be explained by the reduced amount of requested customizations, that was 2.

TABLE III. STATISTICS OF TIME MEASUREMENTS (MINUTES)

Measurement	Round	$\bar{x}_E$	σ_E	$\bar{x}_B$	σ_B	$\bar{x}_B - \bar{x}_E$	$(\bar{x}_B - \bar{x}_E)/\bar{x}_B$
Task 1	1	11.44	03.11	20.13	09.18	08.69	43.2%
	2	07.94	01.48	08.81	03.06	00.87	09.9%
Task 2	1	08.31	06.46	22.94	13.73	14.63	63.8%
	2	10.69	05.76	17.25	01.49	06.56	38.0%
Task 3	1	05.06	02.12	03.92	02.67	-1.14	-29.1%
	2	05.31	01.91	03.44	00.97	-1.87	-54.4%
Total	1	45.81	08.19	70.56	12.14	24.75	35.1%
	2	31.50	01.86	51.38	08.29	19.88	38.7%

As illustrated in Table IV, applications built with the Metamorphic framework barely have runtime errors with measurements indicating the contrary for the baseline treatment. It should still be noted that the amount of non-runtime errors using the experimental framework are slightly lower than runtime errors using the baseline. This corroborates the error preventive nature as there are less errors that when occur are detected faster.

At last, the number of test executions measurement indicates that applications built with Metamorphic require less iterations to validate all the tests.

The results of this experiment should be carefully considered, as the number of subjects may not be representative of the developer community. In order to diminish this threat

 ISBN: 978-1-61208-458-9

TABLE IV. STATISTICS OF ERROR MEASUREMENTS

Measurement	Round	$\bar{x}_E$	σ_E	$\bar{x}_B$	σ_B
# Non-runtime errors	1	02.8	2.06	01.5	1.29
	2	04.3	1.71	00.3	0.50
# Runtime errors	1	00.0	0.00	05.8	4.99
	2	00.3	0.50	05.5	1.73

each subject executed the tasks for each of the frameworks increasing the number of data points.

V. CONCLUSION AND FUTURE WORK

This work required a full and extensive review on REST frameworks, specially model-driven REST frameworks, as very little is documented scientifically. With that knowledge a meta-architecture was developed, through modeling, that is independent of any programming language. A framework that follows such meta-architecture was also developed. This was designed to be as modular as possible by enabling the use of components through cross-project dependency injection.

Validation of the proof of concept revealed that the response to the research problem may be positive. This means that it is possible to improve the development process and execution performance of model-based REST services, through a framework that is written in a statically type-safe programming language that enables code generation in compile time. Improvements in the development process are backed by a reduced number of lines of code, a reduced number of runtime errors, and a reduced development time when using the proof of concept. Despite unexpected execution performances in some cases the authors believe the premise may hold for a more mature framework.

The only identified disadvantage of use of the framework is the additional development time required for compiling applications before testing. For big applications this may be critical as for any small change the entire code will be regenerated.

The source code of the framework can be found in https://github.com/frroliveira/metamorphic and more details about this research, such as the experiments can be found in http://paginas.fe.up.pt/ ei10038/dissert. Further research of the identified problem would be connected with Metamorphic as it is not yet a full featured framework. Future work could be:

1) *Test models flexibility.* Knowledge on Metamorphic's flexibility to other generators is empirical. To ensure such information another repository and service generators could be implemented based in other libraries.
2) *Extension of models.* For a framework to be useful, it may contain most features developers will need. Metamorphic has the basic features, so others would be welcome such as entity inheritance, authentication, database migrations, filtering, and pagination.
3) *Profiling.* Understanding any possible bottleneck in generated applications could be done trough profiling. This would aim to fully validate the performance goal initially established.
4) *Experiments.* Having a new and more mature version of the framework, its validity should be tested in industrial environments with samples that are more size significant. This time both versions, synchronous and asynchronous, should be tested.
5) *Swagger definition.* Swagger [27] defines a "standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code". This would allow faster testing of the generated services.

REFERENCES

[1] I. Society, "Global Internet Report 2014," 2014, URL: http://internetsociety.org/sites/default/files/Global_Internet_Report_2014_0.pdf [accessed: 01, 2016].

[2] J. Kopecký, P. Fremantle, and R. Boakes, "A history and future of web apis," it - Information Technology, vol. 56, no. 3, 2014, pp. 90–97.

[3] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[4] P. K. Potti, S. Ahuja, K. Umapathy, and Z. Prodanoff, "Comparing performance of web service interaction styles: Soap vs. rest," in Proceedings of the Conference on Information Systems Applied Research ISSN, vol. 2167, 2012, p. 1508.

[5] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," Computer, vol. 39, no. 2, 2006, pp. 25–31.

[6] D. Riehle, S. Fraleigh, D. Bucka-Lassen, and N. Omorogbe, "The architecture of a uml virtual machine," SIGPLAN Not., vol. 36, no. 11, Oct. 2001, pp. 327–341.

[7] L. Cardelli, "Type systems," ACM Computing Surveys, vol. 28, no. 1, 1996, pp. 263–264.

[8] D. G. Bobrow, R. P. Gabriel, and J. L. White, "Clos in context-the shape of the design space," Object Oriented Programming: The CLOS Perspective, 1993, pp. 29–61.

[9] Ecole Polytechnique Fdrale de Lausanne - EPFL, "The Scala Programming Language," URL: http://scala-lang.org/ [accessed: 01, 2016].

[10] E. Burmako, "Scala macros: Let our powers combine!: On how rich syntax and static types work with metaprogramming," in Proceedings of the 4th Workshop on Scala, ser. SCALA '13. New York, NY, USA: ACM, 2013, pp. 3:1–3:10.

[11] T. Christie, "Django REST framework," URL: http://django-rest-framework.org/ [accessed: 01, 2016].

[12] D. S. Foundation, "Django," URL: https://djangoproject.com/ [accessed: 01, 2016].

[13] N. Iarocci, "Python REST API Framework — Eve 0.5 documentation," URL: http://python-eve.org/ [accessed: 01, 2016].

[14] A. Ronacher, "Flask," URL: http://flask.pocoo.org/ [accessed: 01, 2016].

[15] StrongLoop, "LoopBack," URL: http://loopback.io/ [accessed: 01, 2016].

[16] N. Foundation, "About — Node.js," URL: https://nodejs.org/about/ [accessed: 01, 2016].

[17] Express, "Express - Node.js web application framework," URL: http://expressjs.com/ [accessed: 01, 2016].

[18] M. McNeil, "Sails.js — Realtime MVC Framework for Node.js," URL: http://sailsjs.org/ [accessed: 01, 2016].

[19] R. J. Wirfs-Brock and R. E. Johnson, "Surveying current research in object-oriented design," Communications of the ACM, vol. 33, no. 9, 1990, pp. 104–124.

[20] Typesafe Inc., "typesafehub/config," URL: https://github.com/typesafehub/config [accessed: 01, 2016].

[21] E. Burmako, M. Odersky, C. Vogt, S. Zeiger, and A. Moors, "Scala macros," November 2013, URL: http://scalamacros.org/paperstalks/2013-11-25-ScalaMacrosPoster.pdf [accessed: 01, 2016].

[22] T. Inc, "Slick," URL: http://slick.typesafe.com/ [accessed: 2016-01-05].

[23] Typesafe Inc, "spray — REST/HTTP for your Akka/Scala Actors," URL: http://spray.io/ [accessed: 01, 2016].

[24] M. V. Zelkowitz and D. R. Wallace, "Experimental models for validating technology," Computer, vol. 31, no. 5, 1998, pp. 23–31.

 ISBN: 978-1-61208-458-9

[25] R. Likert, “A technique for the measurement of attitudes.” Archives of psychology, 1932.

[26] M. Hollander, D. A. Wolfe, and E. Chicken, Nonparametric statistical methods. John Wiley & Sons, 2013.

[27] SmartBear, “Swagger — The World’s Most Popular Framework for APIs.” URL: http://swagger.io/ [accessed: 01, 2016].

 ISBN: 978-1-61208-458-9

Migration from Annotation-Based to Composition-Based Product Lines: Towards a Tool-Driven Process

Fabian Benduhn[1,2], Reimar Schröter[1], Andy Kenner[2], Christopher Kruczek[2],
Thomas Leich[2], and Gunter Saake[1]
University Magdeburg[1], METOP GmbH[2], Germany
[1]email:{fabian.benduhn, reimar.schroeter, gunter.saake}@ovgu.de,
[2]email:{andy.kenner, christopher.kruczek, thomas.leich}@metop.de

***Abstract*—Software product lines allow a developer to produce similar programs based on a common code base. Two main techniques exist: composition-based and annotation-based approaches. Although composition-based approaches offer potential advantages such as maintainability, in practice mostly annotation-based approaches are used. The main reason hindering the migration of existing projects is the difficulty of the transformation process that can take a lot of time in which maintenance and evolution of the system are put on hold. Thus, for a company, it is hard to estimate the transformation costs and the success is uncertain. As already stated in previous work, a hybrid solution using both approaches may be an adequate solution to overcome this problem. Therefore, we propose a migration concept focusing on technical requirements, such as tool- and language support to reduce the risk during the error-prone migration process. We exemplify the concept by considering the partial migration of a real-world system from preprocessor-based variability to an implementation based on feature-oriented programming. We identify conceptual and tool-based challenges that must be addressed for the practical application. We present technical considerations that must be taken into account for step-wise migration and specific challenges related to our case study.**

Keywords–Software Product Lines; Step-wise Migration; Variability Mechanisms; Implementation Techniques.

I. INTRODUCTION

Software product lines are a concept to create similar programs based on a common code base [1][2]. Several implementation techniques with different advantages and disadvantages exist [3]. We distinguish between annotation-based and composition-based approaches. In practice, variability is often implemented by annotating code with preprocessor directives to achieve conditional compilation. In detail, preprocessors are an easily accessible mechanism for fine-grained adaptation of source code to achieve similar programs with low effort. While the use of preprocessors provides an effective way to implement software variability, the technique comes with disadvantages related to code and feature traceability [4][5].

In contrast, composition-based approaches, such as feature-oriented programming (FOP), avoid these problems by providing specialized modularization mechanisms [6][7]. In FOP, each feature, which serves as a configuration option, is encapsulated in a dedicated module so that it can be combined with other features to generate variants. Therefore, compared to annotation-based approaches, the traceability of features is straight forward, which eases maintenance and extension of the source code.

Despite the potential advantages of composition-based over annotation-based approaches, their use has not been widely adopted in industry. There are several reasons for this situation. On the one hand, the usage of FOP in practice is difficult and error-prone and has high requirements regarding sufficient tool support. On the other hand, the process of a complete transformation to these techniques for legacy systems can be time consuming, and it is hard to estimate the transformation costs [8]. Thus, preprocessors remain the dominant approach in industry.

Kästner and Apel formulated the idea to use a combination of annotation-based and composition-based approaches to combine their advantages and to enable a step-wise migration process [9]. We build on this idea of such a hybrid approach and propose a process for its instantiation considering practical concerns, such as required tool support. In a case study, we investigate how Berkeley DB, a database management system using preprocessor directives to implement variability, could be migrated using our migration concept. We present details of the step-wise migration and identify technical and conceptual challenges.

In detail, we make the following contributions:

- We propose a concept for the instantiation of a step-wise migration process based on the combination of annotation-based and composition-based approaches that considers technical concerns.
- We exemplify our concept considering the migration of Berkeley DB, from an implementation in the programming language C with preprocessor annotations to a composition-based implementation using FeatureC, a newly developed extension of FeatureHouse [10].
- We identify technical and conceptual challenges that we have to consider during our migration concept.

In Section II, we introduce the necessary background for the rest of the paper. We propose our tool-driven migration concept in Section III and its practical application to a real-world project in Section IV. In Section V, we discuss several project-specific challenges. We give an overview of related work in Section VI and conclude in Section VII.

II. IMPLEMENTATION OF VARIABILITY IN SOFTWARE PRODUCT LINES

In this section, we give a brief overview on implementation techniques for software product lines. We consider two basic approaches, annotation-based and composition-based [3].

 ISBN: 978-1-61208-458-9

```
1 static int __bam_getbothc(dbc, data)
2   DBC *dbc;
3   DBT *data;
4 {
5   ...
6   return (__bam_getboth_finddatum(dbc, data, ...);
7 }
8
9 #ifdef HAVE_COMPRESSION
10   static int __bam_getlte(dbc, key, data)
11     DBC *dbc;
12     DBT *key, *data;
13   {
14     int ret;
15     ...
16     return (ret);
17   }
18 #endif
```

Figure 1. Conditional compilation using preprocessor directives.

In detail, we focus on the representation of variability in source-code artifacts. However, the distinction between both approaches can be generally applied to many types of variable development artifacts, such as models, specifications, and documentations [11][12][13]. Now, we present advantages of both approaches and their combination.

A. Annotation-Based Implementation of Variability

The idea of annotation-based approaches to implement variability is that certain parts of the development artifacts are mapped to features by annotating them. Individual variants can be generated by removing parts representing undesired features.

As annotation-based technique, preprocessors are commonly used that exist for many languages and tools [14][15][16]. In practice, the usage of preprocessors is widespread, e.g., in open-source projects [17], operating systems, and databases. The mechanism also allows a developer to implement fine-grained variability, including changes to single characters within program statements. In Figure 1, we exemplify the usage of the C preprocessor for conditional compilation. The code in Line 9 is annotated using feature `HAVE_COMPRESSION`. In detail, the beginning of the variable code fragment is marked by the directive `#ifdef`, the end is marked by `#endif`. Thus, if feature `HAVE_COMPRESSION` is not activated, the preprocessor removes the specific code before it is given to the compiler.

Preprocessors support very fine-grained source code variability but this property negatively influences the code comprehension of the system. Thus, alternative annotation-based approaches with more sophisticated tool support and certain restrictions regarding the discipline of annotation usage have been developed [18][19][20]. Similarly, researchers have investigated the code comprehension of preprocessor programs and related problems and proposed several approaches to improve them, e.g., with different background colors [4][18]. Despite these efforts, the general problems with such annotation-based approaches, such as the C preprocessor, are not completely solved. However, preprocessors remain the dominant approach in practice.

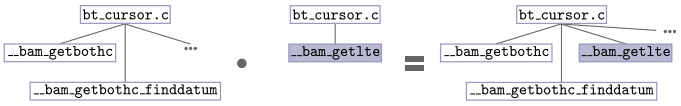


Figure 2. FOP - Structural combination of code artifacts.

B. Composition-Based Implementation of Variability

To overcome the problems of annotation-based approaches, several composition-based implementation techniques have been proposed [6][7][21][22][23][24][25]. In composition-based approaches, variable features of a system are mapped to dedicated modules. The main advantage is that this modularization improves traceability of features and separation of concerns [3].

In this paper, we focus on the composition-based approach FOP. The main idea of FOP is to modularize software into cohesive units — each module encapsulating a particular feature of the software. Individual program variants can be generated by superimposing the desired feature modules based on a hierarchical representation of their structure, called Feature Structure Trees (FST) [10]. In detail, two corresponding inner nodes (i.e., nonterminal) are merged when they have the same type and name. For terminals (i.e. leave nodes, e.g., functions), it depends on the implementation, i.e., whether the node is refined (i.e., extended) or completely overwritten. In Figure 2, we exemplify composition-based variability for FOP. We use the same example as given for the annotation-based approach of Figure 1. In the base feature, the file `bt_cursor.c` with the functions `__bam_getbothc` and `__bam_getbothc_finddatum` exists. However, the file `bt_cursor.c` also exists in feature `HAVE_COMPRESSION` with the function `__bam_getlte`. After the combination, the file includes all functions of both features.

C. Combination of Annotation-Based and Composition-Based Approaches

Despite the potential advantages of composition-based approaches, there are some problems regarding their practical application in industry. First, the migration from legacy systems, which often use preprocessors to implement variability, to composition-based approaches is a difficult and error-prone task. Long migration processes are generally problematic, because they must usually be performed in parallel to the regular development and maintenance of the software, and eventually merged - which is, again, a time-consuming task. Thus, the risk to switch to composition-based approaches is high. Second, composition-based approaches are not suitable for fine-grained variability as used in many preprocessor-based systems. Thus, it is especially difficult to ensure that a migration process results in a well-structured system, which may diminish the potential benefits of the composition-based approach.

As a possible way out of this dilemma, Kästner and Apel proposed the idea to combine annotation-based and composition-based approaches [9]. In detail, they formulated the idea to use this concept to enable a step-wise migration process from annotation-based to composition-based approaches in which intermediate steps use the combined approach and, thus, avoid the usually long, atomic migration process. Kästner and Apel discuss this migration process in a theoretical manner

 ISBN: 978-1-61208-458-9

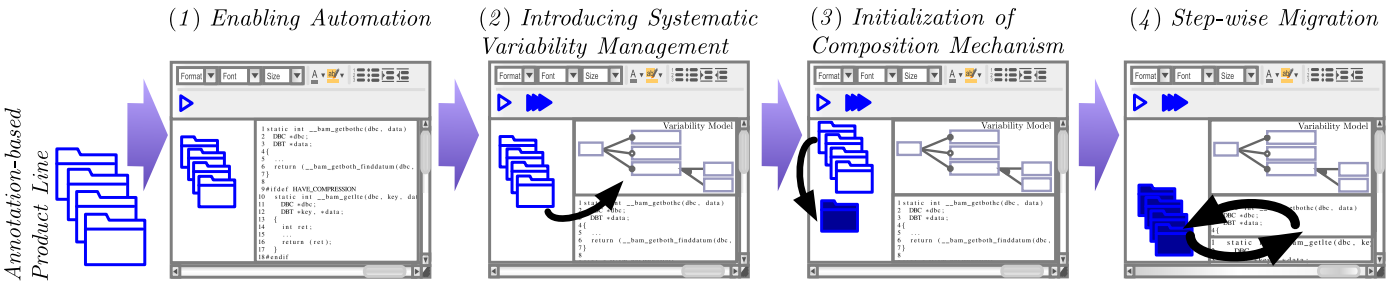


Figure 3. General migration concept for the instantiation of a step-wise migration of annotation-based to composition-based approaches.

and state the need for sufficient tool support as a necessary prerequisite for its practical application. We present a migration concept and discuss practical considerations for companies that are interested in the application of a system transformation.

III. Tool-Driven Migration Concept

In this section, we introduce our concept for the tool-driven migration of software systems with annotation-based implementation of variability to a system using composition-based variability. Our concept is generally applicable in the sense that it is independent of specific annotation-based and composition-based implementation techniques. It describes a process consisting of four steps, as depicted in Figure 3. In particular, we have to (1) enable automation, (2) introduce support for systematic variability management, (3) initialize the desired composition mechanism, and (4) apply the step-wise migration.

One of the goals of our approach is to ensure a consistent state in each step. Thus, a long migration process that hinders regular development of the project is avoided and costs to merge the migration results can be reduced. When applying this to concrete projects, the details on how this consistency is ensured may vary. Ensuring a minimal interruption of the continuous development, requires that the application can be compiled, executed, and tested in each step. In the following, we describe each step of our concept in detail.

(1) Enabling Automation

The first step of our migration concept is to enable automation of important tasks in the software-development process for a given project. This will enable the developer to validate the correctness of each subsequent step. Typically, this includes the capability to automate the process of building, executing, and testing the program as provided by many modern integrated development environments (IDEs). The choice of the specific IDE depends on the later steps, e.g., the IDE's support for variability management tools. Thus, even if the project already provides automation for relevant tasks, it might be still necessary to switch the used IDE in preparation of the next steps.

(2) Introducing Systematic Variability Management

Having ensured that we enabled automation, execution, and testing of our software, a systematic variability management should be introduced. An important aspect of variability management is to provide techniques for the modeling of the variability space and a mapping to code artifacts. Therefore, the variability artifacts must be identified and possibly re-engineered for integration. Furthermore, variability-related tool support must be provided to establish a tool-driven configuration process and the automation of variant generation. In addition, the possibility to execute variants or test them is a frequently desired aspect.

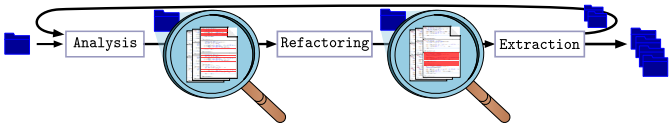


Figure 4. Step-wise migration in detail.

(3) Initialization of the Composition Mechanism

In this step, the technical prerequisites to employ a composition mechanism must be ensured. The result of this step is a trivial decomposition into a single module. However, it must be ensured that the used tools support the desired language, and that modules can be composed accordingly. For instance, if we take FOP as composition-based approach, we have to initialize the project using one base feature module in which we include all code fragments with all preprocessor directives. Afterwards, it must be possible to apply the composition mechanism, which, in this step, results in an output that is equal to the input (i.e., equal to the base feature). Furthermore, we have to ensure that the automated variant-generation mechanism using preprocessors of Step (2) can be used to remove the undesired features of this composed output.

(4) Step-wise Migration

After the execution of the previous steps, the main task of the step-wise migration can start. The step-wise migration consists of small refactoring steps in which we can extract a specific source-code artifact into a module (cf. Figure 4). Therefore, it is necessary to (4.1) analyze the code to identify the next fragment for the modularization, (4.2) refactor the code to improve the structure, so that it is easier to modularize, and (4.3) extract the corresponding code fragment into a module. Since we define each sub-step as a refactoring, the complete product line should be in a consistent state before and after each step. This can be ensured by taking advantage of the support for automated building, executing, and testing, which has been ensured in the first step.

IV. Practical Application to a Real-World Project

In the previous section, we introduced a general process for the step-wise migration of annotation-based to composition-based approaches with the focus on necessary tool support to

 ISBN: 978-1-61208-458-9

enable a successful practical application. Now, we investigate the applicability of this process to initialize the migration of a real-world project. Our focus is to identify technical and conceptual concerns and possible challenges for the process that may be useful to guide companies in their own migration projects.

We consider the application of the proposed migration concept to Berkeley DB, a database management system, from an annotation-based implementation using the C preprocessor to a composition-based implementation based on FOP. In detail, Berkeley DB consists of 229.419 lines of code. Since Berkeley DB is written in C with preprocessor annotations, it is a practically relevant case study with high challenges of an annotation-based product line. Another reason for selecting Berkeley DB is that it was used in several case studies related to research on variability (for more details, see Section VI). According to the properties of Berkeley DB, we have to search for an IDE that allows us to apply our migration concept and that supports variability management for C with preprocessors and a solution for FOP in the programming language C.

Since it supports the required properties for our migration task, we have chosen the IDE Eclipse with the plugin FeatureIDE for variability support. FeatureIDE supports the complete development cycle of software product lines including the modeling step, as well as different implementation techniques, such as annotation-based and composition-based approaches (i.e., also FOP) [26]. Therefore, besides a hybrid solution of annotation- and composition-based approaches, it fulfills all theoretical requirements of our migration concept. In detail, FeatureIDE integrates a set of command line tools that support different implementation techniques. In our application, we extend the command line tool FeatureHouse which implements FOP based on superimposition as introduced in Section II-B [10]. In the following, we describe for each step of the migration process some of the relevant technical decisions and the tool-specific steps that have to be performed. We focus on the description of the applicability of the process in general, as well as on the practicability of the specific tools we have used.

(1) Enabling Automation – Integration into Eclipse

As described before, to enable automation of the build and testing process for Berkeley DB, we use Eclipse because we plan to use FeatureIDE during the subsequent steps. In detail, we create a new Eclipse project for the programming language `C` and include our case study Berkeley DB with all files and additional material. For the automated generation and execution of a specific variant, we administrate further Eclipse settings so that we ensure a correct behavior of Berkeley DB (i.e., we do not consider variability management so far). In detail, we use the C and C++ Integrated Development Environment (CDT) of Eclipse for the configuration and apply the subsequent make process. In our case, this step took several hours. But the necessary time for this step strongly depends on the complexity of the application and on the intended test mechanism that should ensure the correctness of the system. Even if this step may take longer for other projects, it has to be done only once for the complete migration process.

(2) Introducing Systematic Variability Management – Using FeatureIDE

The next step is to introduce systematic variability management for this Eclipse Berkeley DB project. Using FeatureIDE, we can automatically convert the project into a FeatureIDE project, by adding specific configuration properties. As result, it is possible to automate the configuration, generation and execution for each variant of the product line. A central part of this step is to make the variability explicit. Therefore, we need to create a feature model in which we define configuration options in terms of features and their dependencies. This task was easy for us, because Berkeley DB was studied several times and, thus, we reused information of a previous study and selected 10 out of 28 preprocessor variables as configurable features. As result, we use FeatureIDE to create a feature model of Berkeley DB that consists of one root feature and ten optional child features (i.e., they can be combined arbitrarily to variants). Afterwards, we can use FeatureIDE's configuration editor to specify the specific variants of Berkeley DB that we want to create. If the build process of a project is straight forward, FeatureIDE already supports direct compilation and execution. However, in the case of Berkeley DB, the build process involves a specific configuration process. Thus, FeatureIDE needs to start the configuration and make process for which we developed an extension to bridge the gap. Because of our experience in plugin development, it was easy to create a first prototype for our needs. Therefore, this step takes only a few days for us. By contrast, without our experience and without knowledge of the available variability options, this step can also take several weeks. Furthermore, the step can be extended arbitrarily, this depends on the tool support that a company needs. For instance, it is also possible to develop IDE views to give a code outline or editors with specialized visualization techniques to highlight variability. As explained above, we have decided to use FeatureIDE, but there are other tools for variability management that could be used, depending on project-specific requirements [27]. The time that is required for this step depends largely on project-specific infrastructure, but in general this step has to be performed only once per migration process

(3) Initialization of the Composition Mechanism – Using FeatureC

The main advantage of FeatureIDE is that the plugin also supports other implementation techniques. Thus, in this step, we can change the implementation strategy that is used by FeatureIDE. However, no existing tool supports the required hybrid approach. Thus, we have developed FeatureC, a feature-oriented extension of C that additionally supports the use of preprocessors. Therefore, we have extended the existing composition tool FeatureHouse. As mentioned above, FeatureHouse is a command line tool for FOP in which different languages can be integrated using a specialized grammar [10]. On the one hand, this grammar is needed to parse a specific programming language. On the other hand, the grammar encodes the composition rules for this specific language. For a hybrid solution, the already existing C grammar of FeatureHouse has to additionally support annotations. Therefore, we developed the extension FeatureC. With FeatureC, we can create one feature module in which all the existing annotation-based source code is located. Afterwards, we can use the same procedure to configure, compile and execute a specific variant.

 ISBN: 978-1-61208-458-9

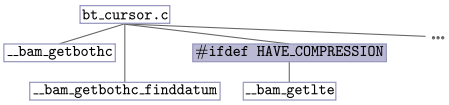


Figure 5. Required grammatical changes to support annotations in feature modules - illustrated by using the example of Figure 1.

For our case study Berkeley DB, several grammar changes were necessary. In detail, we adapted the grammar in a way that directives outside of a function are represented as nonterminal. This change allows us to parse the preprocessor annotations and to produce an output in which this annotation still exists. We depict this fundamental change in Figure 5. Here, we depict the structural representation of the file given in Figure 1 that can be combined with the corresponding nodes of another feature.

By contrast to the mentioned grammatical change, in some cases we were able to avoid grammatical changes through source-code disciplines (see next section for detailed information). However, in hindsight, we spent the most time on grammar changes and code disciplination. All other parts of this step were partly straight forward. In sum, the complete step required several weeks; however it is largely dependent on the project-specific languages as well as tools, and is independent from the actual size of the project or the number of features to be extracted.

(4) Step-wise Migration – Application on Berkeley DB

In this step, the actual migration procedure can start. As stated in the previous section, each refactoring consists of three sub-steps. For our case study, we used a manual approach to extract two features (`HAVE_HASH` and `HAVE_HEAP`) with a focus on the identification of existing challenges. The identified project-specific challenges are discussed in the next section.

While we have used a manual approach, we identified the need for further tool support. We give a short overview on approaches that could aid the migration procedure based on our experience with the manual approach. In future work, we plan to investigate the integration of selected tools. For code analysis, several tools exist that consider variability [28]. On the one hand, it would be possible to use tools that aid the developer to identify potential code fragments representing features. On the other hand, code analysis can be used to ensure the correctness of each sub step. With `Morpheus`, Liebig et al. provide a promising tool for automated refactoring of C code that can cope with preprocessor directives [29]. Kästner et al. provide a concept for automatable refactorings that can be used to aid the migration from annotation-based to composition-based implementations [30]. The approach requires specific disciplined annotations and supports a subset of Java, for which it can be guaranteed that the refactorings can be performed correctly. It should be investigated to which extend this approach can be applied to real-world systems in other programming languages, such as C, for which preprocessor directives already exist. Furthermore, there are several approaches to transform preprocessor-based implementations into aspect-oriented implementations [31][32]. While we used feature-oriented programming for our case study, our general process can also be applied to a migration to an aspect-oriented product line. Therefore, this line of research may be interesting for the practical application.

V. Project-Specific Challenges

In this section, we discuss details of our experiences and challenges during the application of our migration concept to two features of Berkeley DB. Therefore, we present several insights regarding the application of concepts and discuss project-specific challenges regarding the tools and languages used for the migration of Berkeley DB.

A. Interdependence of Process Steps

In our proposed tool-driven migration concept, we propose multiple steps to achieve an environment in which a step-wise and fine-grained migration can be applied to an annotation-based product line. As a result, in Step 4, it should be possible to refactor the code in a fine-grained manner so that we can introduce a composition-based approach in which each intermediate step presents a fully-functional hybrid solution. As a result, the transformation process is much more predictable and less risky, because it is not necessary to perform the complete migration process in one atomic step - avoiding expensive parallel development or delays. Concepts to cope with the interdependency of the individual process steps are required.

Considering the migration of Berkeley DB, we found out that a revision of a previous step can help to ease the current step. For instance, after we started to refactor Berkeley DB, we identified several tool-driven, as well as conceptual challenges that required changes to the initialization phase of Step 3. In detail, it could be beneficial to change the language support of FeatureC instead of adapting Berkeley DB in an awkward manner. This problem is strongly related to our next remarks.

B. Undisciplined Use of Preprocessors

We started by using existing tools for the variability management of our case study. However, it turned out that the language support was not sufficient. In particular, the composition mechanism for FOP (i.e., FeatureHouse) did not support the hybrid strategy in which annotations are completely supported in compositions. Therefore, we introduced FeatureC with some grammatical changes so that it is possible to use FeatureHouse in a straight-forward manner.

Besides the discussed grammar change of Figure 5, we realized further adaptions of the underlying C grammar. However, it was possible to avoid some grammar changes because their necessity was the result of an undisciplined usage of preprocessor annotations. Therefore, we restricted the usage of annotations to certain useful patterns. In Figure 6, we show an example of such a problematic annotation that we have found in the source code of Berkeley DB. On the one hand, the restriction to disciplined preprocessor annotations is a promising approach to increase the understandability of the code. On the other hand, it would have been technically difficult to implement an approach flexible enough for all cases, while still taking advantage of the existing tool infrastructure. As a result, we had to refactor the code to introduce the desired preprocessor discipline. In general, it is advisable to consider the introduction of preprocessor discipline in concert with providing the necessary language support for the related tools.

 ISBN: 978-1-61208-458-9

```
1  __db_errx(env, DB_STR_A("0153",
2       "s(u): host lookup failed: s","s u s"),
3       nodename == NULL ? "" : nodename, port,
4#ifdef DB_WIN32
5       gai_strerrorA(ret));
6#else
7       gai_strerror(ret));
8#endif
```

```
9#ifdef DB_WIN32
10  __db_errx(env, DB_STR_A("0153",
11        "s(u): host lookup failed: s","s u s"),
12        nodename == NULL ? "" : nodename, port,
13       gai_strerrorA(ret));
14#else
15  __db_errx(env, DB_STR_A("0153",
16        "s(u): host lookup failed: s","s u s"),
17        nodename == NULL ? "" : nodename, port,
18       gai_strerror(ret));
19#endif
```

Figure 6. Disciplining of annotation-based approaches.

C. Variability in the Presence of Scope-Sensitive Statements

In our practical application of our migration concept, we adapted existing tool support for FOP to cope with our specific language requirements, i.e., for C code with preprocessor annotations. The migration process to introduce FOP involves the creation of so-called hook functions, which are a mechanism to add code by a refinement in the middle of the function. The reason is that when only a small part of a function is variable, it is often useful to extract this part into a hook function, enabling more fine-grained refinements. While this process may often be straight forward, it may require modifications that may introduce errors. In particular, we experienced that statements that depend on the current scope are potentially problematic in Berkeley DB for instance `goto`, `switch`, and `return` statements. In Table I, we summarize how often these statements exist in the complete code, as well as how often they are part of variable code and compare these values with all cases in which their usage results in a problem for all identified features, selected features, and our two extracted features, `HAVE_HASH` and `HAVE_HEAP`.

goto-Statements

In Figure 7, we show an example of a `goto`-statement of Berkeley DB. Assume, we extract the code related to Feature `HAVE_HASH` (Line 6-12) into a hook function without further considerations. The result would be that the `goto`-statement from Line 9 operates in a new scope and, thus, the corresponding `goto`-label would be out-of-scope, and lead to an error. In the complete code of Berkeley DB we count 4642 occurrences of `goto` statements, whereas only 635 interact with variable code. 20% of these occurrences result in a potential problem considering all features. Several strategies exist to solve this situation. Depending on the exact occurrences of the problem, we have to select the most promising strategy that could lead to new required tool support.

The straight-forward solution is to extract the complete method into the `HAVE_HASH` feature module. This would lead to code clones and diminish the benefit of the modularization. In some cases, it is possible to create a hook method in which we insert the considered variable code part including the code artifacts behind the `goto`-label. This is often possible in case of ordinary error handling. However, this might require further modifications, such as a huge parameter list, in which the current state of all variables in the scope of the `goto`-label must be preserved. A third solution considers a more conceptual mechanism of inlining, which allows us to extract only the variable code into a feature module also with the `goto`-statement. Through the generation process of the combined code, it is possible to achieve a code artifact in which the scope is again in the correct scope and works correctly. However, this solution separates the `goto`-statement from the label and, thus, breaks the convention regarding this mechanism.

switch-Environment

In Figure 7, we illustrate a switch construct that is problematic and cannot be treated in a straight-forward manner. In detail, it is not possible to extract the complete case into a hook method using a 1:1 extraction of the variable parts. Similar to the `goto`-statement, the scope will change and the case cannot be handled. In Table I, we can see that Berkeley DB includes 429 switch statements, whereas 16% are involved in variable code artifacts. Only half of them, exactly 36, result in a case that is not straight-forward manageable. As the problematic statements are relatively few, an expensive adaption of the tool-support may not be advisable in this case. Nevertheless, we discuss multiple solution strategies.

Similar to the `goto`-statement, multiple solution strategies exist for this example. First, it is possible to extract the complete switch into a feature module. However, this is not a viable option if the number of the used variables and, thus, needed parameters is too high. A second option could be a hook method inside of the case. For this, it must be ensured that the value `DB_HASH` (cf. Line 7) is actually defined and this could be error-prone. Third, we also can change the composer strategy so that, for instance, a new keyword allows us to refine the case on this specific line. However, special knowledge is needed to apply such grammar extensions.

return-Statements

Besides the previous problems, we also identified the extraction of `return` statements to hook functions as a challenge. Again, if we extract corresponding code artifacts, we change the scope of these statements. However, the problem occurs if a return statement is only defined for some cases and not for all, such as could appear in an if-else case in which only one case contains a return. In the code of Berkeley DB, there are 7779 return statements while only 1254 are part of variable code. Again, 20% of these variable cases lead to a potential problem that we can solve in multiple ways.

One solution is an additional parameter of the hook function that indicates whether we should utilize the return statement. Afterwards, the caller side needs to evaluate this additional parameter. By contrast to the first solution, in some cases, we can directly insert a new return statement for our purpose. In detail, if we know that, for instance, all original return statements result in a positive integer, we can also use a negative integer to indicate that a return is not valid. Furthermore, we can also use a grammar adaption as third solution. In this case, it is necessary to create a kind of an inlining. Nevertheless, the challenge is that a new function needs a `return` statement in all cases to be compatible to the signature. In the grammar-adaption solution, this can be solved with an additional keyword.

 ISBN: 978-1-61208-458-9

```
1 //db/partition.c
2 switch (new_dbc->dbtype) {
3     case DB_BTREE:
4       ...
5       break;
6 #ifdef HAVE_HASH
7     case DB_HASH:
8       if ((ret = __ham_stat(new_dbc, &hsp, flags)) != 0)
9         goto err;
10      ...
11      break;
12 #endif
13    default:
14      break;
15    }
```

Figure 7. Example of potential problems to extract hook methods.

VI. Related Work

The idea to combine annotation-based with composition-based implementation techniques to facilitate a step-wise migration process has been formulated by Kästner and Apel [9]. In our work, we consider a similar concept and investigate its application to a real-world project with a focus on technical concerns and challenges that must be considered to prepare and perform the actual migration process.

Researchers have investigated advantages and disadvantages of many different techniques to implement variability for product lines [3], and proposed several possible combinations. Aspectual Feature Modules have been proposed to combine the advantages of two different composition-based approaches: aspect-oriented programming and feature-oriented programming [33]. In contrast, we focus on a combination of annotation-based and composition-based approaches to enable the possibility of a step-wise migration of the prevalent preprocessor-based product lines in practice. Similarly, the choice calculus has been proposed to provide a formal basis for the combination of annotation-based and composition-based approaches to combine their benefits [34]. However, the potential application to migration has not been discussed. In order to improve the practicability of approaches that combine annotation-based with composition-based implementation, Behringer proposes a concept to improve the visualization of the underlying representations [35]. This complements our work, in the sense that it could be used to ease the practical application of the migration process.

Rosenmüller et al. also extracted feature modules from an implementation of Berkeley DB [36]. Complementary to our experiences, they have used a step-wise process in which they refactored the C code into C++, and transformed the resulting object-oriented version into a feature-oriented version of C++. In contrast, we have focused on a direct application of FOP to avoid long and costly refactorings and transformations with the goal to minimize the risk of the overall process. Furthermore, they identified the use of local variables as a potential problem that must be considered when extracting hook methods.

Alves et al. present a case study in which they migrated a real-world product line from an annotation-based to an aspect-oriented implementation [37]. They identify several language-specific strategies to transform certain variability patterns in the code into aspects. For the specific case of aspect-oriented programming this work provides complementary insights that can be used as a guide to perform the actual transformation, i.e., Step 4 in our proposed process.

TABLE I. OCCURRENCES OF POTENTIAL PROBLEMS IN BERKELEY DB.

			Number of Problems in		
	Complete Code	Variable Code	All Features	Selected Features	Extracted Features
#switch	429	71	36	19	11
#goto	4642	634	127	58	35
#return	7779	1254	248	34	9

VII. Conclusion and Future Work

Software product line engineering provides different development approaches. Whereas annotation-based approaches are mainly used in industry, compositional approaches promise advantages, such as eased maintainability. However, if a company plans to transform an existing annotation-based into a composition-based product line, a wide range of challenges exist. Whereas complete refactorings are error-prone and can take a lot of time, existing work proposed to use a hybrid solution that combines both approaches. We build on this idea and propose a detailed instantiation that allows us to transform the system in a step-wise manner. This tool-driven concept is based on intermediate results and ensures that the complete product line is in a consistent state at all times. To exemplify the application of our approach, we developed FeatureC, a hybrid implementation technique, and applied it to Berkeley DB as a case study. We identified challenges regarding the composition-based concept as well as the granularity of modularizations. In future work, we plan to apply our migration concept to further case studies. Furthermore, we aim to use the concept for projects of our industrial partners.

Acknowledgments

This work is partially funded by the BMBF project NaVaS (grant number 01IS14017A and 01IS14017B).

References

[1] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns. Boston, MA, USA: Addison-Wesley, 2001.

[2] K. Pohl, G. Böckle, and F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Berlin, Heidelberg: Springer, 2005.

[3] S. Apel, D. Batory, C. Kästner, and G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation. Berlin, Heidelberg: Springer, 2013.

[4] J. Feigenspan, C. Kästner, S. Apel, J. Liebig, M. Schulze, R. Dachselt, M. Papendieck, T. Leich, and G. Saake, "Do Background Colors Improve Program Comprehension in the #Ifdef Hell?" Empirical Software Engineering (EMSE), vol. 18, no. 4, 2013, pp. 699–745.

[5] D. Le, E. Walkingshaw, and M. Erwig, "#ifdef Confirmed Harmful: Promoting Understandable Software Variation," in Proceedings of the International Symposium on Visual Languages and Human-Centric Computing (VL/HCC). Washington, DC, USA: IEEE Computer Science, 2011, pp. 143–150.

[6] C. Prehofer, "Feature-Oriented Programming: A Fresh Look at Objects," in Proceedings of the European Conference on Object-Oriented Programming (ECOOP). Berlin, Heidelberg: Springer, 1997, pp. 419–443.

[7] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling Step-Wise Refinement," IEEE Transactions on Software Engineering (TSE), vol. 30, no. 6, 2004, pp. 355–371.

[8] P. Clements and C. Krueger, "Point / Counterpoint: Being Proactive Pays Off / Eliminating the Adoption Barrier," IEEE Software, vol. 19, no. 4, 2002, pp. 28–31.

[9] C. Kästner and S. Apel, "Integrating Compositional and Annotative Approaches for Product Line Engineering," in Proceedings of the GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering (McGPLE). Passau, Germany: Department of Informatics and Mathematics, University of Passau, 2008, pp. 35–40.

[10] S. Apel, C. Kästner, and C. Lengauer, "Language-Independent and Automated Software Composition: The FeatureHouse Experience," IEEE Transactions on Software Engineering (TSE), vol. 39, no. 1, Jan. 2013, pp. 63–79.

[11] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela, "Software Diversity: State of the Art and Perspectives," International Journal on Software Tools for Technology Transfer (STTT), vol. 14, 2012, pp. 477–495.

[12] F. Benduhn, T. Thüm, M. Lochau, T. Leich, and G. Saake, "A Survey on Modeling Techniques for Formal Behavioral Verification of Software Product Lines," in Proceedings of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS). New York, NY, USA: ACM, 2015, pp. 80:80–80:87.

[13] M. Alférez, R. Bonifácio, L. Teixeira, P. Accioly, U. Kulesza, A. Moreira, J. a. Araújo, and P. Borba, "Evaluating Scenario-Based SPL Requirements Approaches: The Case for Modularity, Stability and Expressiveness," Requirements Engineering, vol. 19, no. 4, 2014, pp. 1–22.

[14] GCC Development Team, "The C Preprocessor," Website, 2015, available online at http://gcc.gnu.org/onlinedocs/cpp/index.html; visited on October 29th, 2015.

[15] Munge Development Team, "Munge: A Purposely-Simple Java Preprocessor," Website, 2015, available online at http://github.com/sonatype/munge-maven-plugin; visited on October 29th, 2015.

[16] J. Pleumann, O. Yadan, and E. Wetterberg, "Antenna: An Ant-to-End Solution For Wireless Java," Website, 2015, available online at http://antenna.sourceforge.net/; visited on October 29th, 2015.

[17] J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze, "An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines," in Proceedings of the International Conference on Software Engineering (ICSE). Washington, DC, USA: IEEE Computer Science, 2010, pp. 105–114.

[18] C. Kästner and S. Apel, "Virtual Separation of Concerns – A Second Chance for Preprocessors," Journal of Object Technology (JOT), vol. 8, no. 6, 2009, pp. 59–78.

[19] Big Lever Software Inc., "Gears: A Software Product Line Engineering Tool," Website, 2015, available online at http://www.biglever.com/solution/product.html; visited on October 29th, 2015.

[20] pure::systems, "pure::variants," Website, 2015, available online at http://www.pure-systems.com/pure_variants.49.0.html; visited on October 29th, 2015.

[21] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," in Proceedings of the European Conference on Object-Oriented Programming (ECOOP). Berlin, Heidelberg: Springer, 1997, pp. 220–242.

[22] I. Schaefer, L. Bettini, V. Bono, F. Damiani, and N. Tanzarella, "Delta-Oriented Programming of Software Product Lines," in Proceedings of the International Software Product Line Conference (SPLC). Berlin, Heidelberg: Springer, 2010, pp. 77–91.

[23] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, Jr., "N Degrees of Separation: Multi-Dimensional Separation of Concerns," in Proceedings of the International Conference on Software Engineering (ICSE). New York, NY, USA: ACM, 1999, pp. 107–119.

[24] Y. Smaragdakis and D. Batory, "Implementing Layered Designs with Mixin Layers," in Proceedings of the European Conference on Object-Oriented Programming (ECOOP). London, UK: Springer, 1998, pp. 550–570.

[25] A. Bergel, S. Ducasse, and O. Nierstrasz, "Classbox/J: Controlling the Scope of Change in Java," in Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA). New York, NY, USA: ACM, 2005, pp. 177–189.

[26] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "FeatureIDE: An Extensible Framework for Feature-Oriented Software Development," Science of Computer Programming (SCP), vol. 79, no. 0, 2014, pp. 70–85.

[27] J. Pereira, K. Constantino, and E. Figueiredo, "A Systematic Literature Review of Software Product Line Management Tools," in Software Reuse for Dynamic Systems in the Cloud and Beyond, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8919, pp. 73–89.

[28] J. Meinicke, T. Thüm, R. Schöter, F. Benduhn, and G. Saake, "An Overview on Analysis Tools for Software Product Lines." New York, NY, USA: ACM, 2014, pp. 94–101.

[29] J. Liebig, A. Janker, F. Garbe, S. Apel, and C. Lengauer, "Morpheus: Variability-aware Refactoring in the Wild," in Proceedings of the International Conference on Software Engineering (ICSE). Piscataway, NJ, USA: IEEE Computer Science, 2015, pp. 380–391.

[30] C. Kästner, S. Apel, and M. Kuhlemann, "A Model of Refactoring Physically and Virtually Separated Features," in Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE). New York, NY, USA: ACM, 2009, pp. 157–166.

[31] B. Adams, W. De Meuter, H. Tromp, and A. E. Hassan, "Can We Refactor Conditional Compilation into Aspects?" in Proceedings of the International Conference on Aspect-Oriented Software Development (AOSD). New York, NY, USA: ACM, 2009, pp. 243–254.

[32] A. Reynolds, M. E. Fiuczynski, and R. Grimm, "On the Feasibility of an AOSD Approach to Linux Kernel Extensions," in Proceedings of the AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software. New York, NY, USA: ACM, 2008, pp. 8:1–8:7.

[33] S. Apel, T. Leich, and G. Saake, "Aspectual Feature Modules," IEEE Transactions on Software Engineering (TSE), vol. 34, no. 2, 2008, pp. 162–180.

[34] E. Walkingshaw and M. Erwig, "A Calculus for Modeling and Implementing Variation," in Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE). New York, NY, USA: ACM, 2012, pp. 132–140.

[35] B. Behringer, "Integrating Approaches for Feature Implementation," in Proceedings of the International Symposium Foundations of Software Engineering (FSE). New York, NY, USA: ACM, 2014, pp. 775–778.

[36] M. Rosenmüller, S. Apel, T. Leich, and G. Saake, "Tailor-made data management for embedded systems: A case study on berkeley DB," Data and Knowledge Engineering, vol. 68, no. 12, 2009, pp. 1493–1512.

[37] V. Alves, A. Costa Neto, S. Soares, G. Santos, F. Calheiros, V. Nepomuceno, D. Pires, J. Leal, and P. Borba, "From Conditional Compilation to Aspects: A Case Study in Software Product Lines Migration," in Workshop on Aspect-Oriented Product Line Engineering (AOPLE), 2006.

 ISBN: 978-1-61208-458-9

Executable Testing Based on an Agnostic-Platform Modeling Language

Concepción Sanz[1], Alejandro Salas[1], Miguel de Miguel[1,2], Alejandro Alonso[1,2], Juan Antonio de la Puente[1,2]
[1]Center for Open Middleware, Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo, Pozuelo de Alarcón - Madrid, Spain
[2]DIT, Universidad Politécnica de Madrid (UPM), Madrid, Spain
Email: {concepcion.sanz, alejandro.salas, miguelangel.demiguel, alejandro.alonso}@centeropenmiddleware.com,
jpuente@dit.upm.es

***Abstract*—Among the different approaches to deal with testing, model-based techniques come up as promising solutions due to their independence from testing platforms, their quick adaptation to changes during the modeling process, and the use of abstract concepts, which make testers focus only on the business. However, most of the solutions are proprietary or based on the complexity of the Unified Modeling Language (UML) models. We propose a testing framework based on a non-UML test-centred modeling language which is agnostic about any execution platform. The framework also allows to increase the expressivity of the test models by designing execution flows, which connect to elements existing in test models. Apart from design tests, the proposed framework deals with execution, including the necessary mechanisms to transform models, on demand and in an automatic way, into executable tests for a variety of testing platforms. In order to show the flexibility of the proposed approach, we introduce as case study two testing platforms which manage tests in different ways.**

***Index Terms*—Model-based testing; Reuse.**

I. INTRODUCTION

Research on testing activities has provided along time a wide variety of approaches to deal with more and more demanding needs in terms of quality assurance, time-to-market, devices, test automation, productivity and maintenance, among others. Among the existing approaches, it can be found mechanisms which were initially considered for purposes different from testing, as it happens with the approximations based on models. Models were only considered for documentation at first, later were included in software development mechanisms and now can be found in testing frameworks as promising alternatives for increasing the automation and the abstraction level of the testing activities.

Most of the non-model-based testing frameworks offer solutions which are highly coupled to the specific platform where tests will be finally executed. This makes necessary having a deep technical knowledge about the target platform, preventing non-expert people to take advantage of the testing frameworks. The absence of models also makes difficult the migration of tests between different testing frameworks. A significant effort in time and work is necessary in order to adapt the entire collection of tests to a new framework. Currently, as software applications increase their complexity and extend the list of devices and platforms where being executed, the variety of frameworks that can be involved during the testing process also increases. Such diversity leads to more complex testing environments and requires specialized testers to manage them. Some examples of testing frameworks are QuickTest Professional(QTP-HP)[1], JUnit [2] and Selenium [3].

In contrast, approaches based on models offer an intermediate layer between the tester and the final testing platform, allowing users to design tests based on abstract concepts, which are independent from any platform. Once tests are described as models, they are translated into executable tests for a specific platform. This is the approach followed by commercial frameworks, such as Conformiq [4], which translates models into JUnit, Selenium or HP Functional Testing [5], among others. The level of abstraction existing in a model-based approach provides a faster mechanism to design tests, since testers only need to focus on the business domain instead of technology. Besides, the conversion from test models to executable tests is usually made in an automatic way, being less error-prone than applying non-modelled approaches directly.

Regarding model-based approaches, most of commercial and open source solutions are usually based on UML, a general-purpose language whose extension and lack of a precise semantic has given rise to the development of more specific languages for testing. Among these, it can be mentioned the specific UML Testing Profile (U2TP) [6], or domain specific languages (DSLs) for easing the use of UML diagrams for testing purposes [7]. These alternatives are still dependent on UML and usually need to be combined with action languages in order to provide behaviour and execution features. Regarding functionality, commercial model-based frameworks offer integrated proprietary solutions where users can design models; execute them in their own platform; or allow the transformation of the models into tests for specific testing languages. However, it is difficult to find all these features in open solutions, so users usually need to integrate and adapt different tools by themselves in order to get similar functionalities.

Consequently, two are the goals to achieve in this work. Firstly, developing an open framework to enable the design of platform-independent test models using abstract concepts ease to manage by non-expert testers and non-UML based. Secondly, enabling the integration of the independent test models with different testing platforms, so users do not require deep knowledge of those platforms to get executable tests.

There has been developed an Eclipse-based open-source testing framework which allows the design of platform-independent test models and their integration with different execution platforms by means of customized transformations.

 ISBN: 978-1-61208-458-9

Thus, one test model can be used in different testing environments just applying, in an automatic way, the right transformation. These transformations enable users to make use of different testing environments simultaneously without being experts on all of them, and ease the migration among platforms. The only expert needed is the one that builds the transformation rules. Among the testing environments involved in the proposed framework, JUnit and Selenium are the ones selected as open-source tools.

The proposed framework is based on a non-UML based modeling language which manages high-level concepts related to testing. These concepts isolate test models from specific testing frameworks and allow the reuse of test elements among models, easing the design of tests to non-experts testers. The framework not only allows to design tests but also establishes platform-independent execution flows using the elements existing in the test models.

The rest of the paper is structured as follows. Sections I and II motivates this work and provides an overview of the state-of-the-art respectively. Section III gives an overview of the testing framework and introduces the way execution flows are set. Section IV explains how the proposed framework connects models to different testing platforms giving rise to test executions. Finally, Section V summarizes the main results of this work and sketches the lines for future work. From the development point of view, a fully functional prototype of the entire proposed framework has been implemented, providing details about it along the different sections of the paper.

II. Related work

The need for producing reliable applications and ensuring the functionality in a scenario of short time-to-market, tight budgets and complex applications, puts pressure on the testing process that the companies carry out. Research on this topic is increasing, giving rise to a large variety of testing approaches, and thus, to a large variety of platforms for testing. This variety requires testers with a large experience and a deep knowledge in each platform in order to take advantage of all the existing features. This expertise prevents other potential users, such as developers, from making or at least sketching their own tests.

Among the most promising approaches applied to testing, we can find solutions based on models. According to this paradigm, test cases can be designed using concepts with low or none technical knowledge. Through these concepts, the functional behaviour of the system under test can be described, giving rise to test models, which can be later transformed into tests which are executable in a specific testing platform. The functional behaviour can be described using concepts based on formal specifications (e.g., B, Z) [8][9], diagrams of any kind - state charts, use case, sequence diagrams [10], (extended) finite state machines [11][12] -, or graphs, among others. Among all, one of the most popular ways to describe tests is based on diagrams, being the UML modeling language [13] the most extended language to design them. UML is a general-purpose modeling language and owns a wide and ambiguous semantic. For this reason, and due to its extended use in testing activities, it was developed a specific UML profile, UML 2.0 Testing Profile (U2TP) [6], which has been used in different research [14][15][16]. However, the ambiguity and complexity still continue in U2TP, motivating the development of alternative DSLs, although still linked to UML [7]. Once models are described, independently of the applied modeling language, its execution is not straightaway due to the absence of an execution engine in the framework used for the design process. Even when an engine is included, models need to be manually adapted or customized in order to be executed.

The proposal described in this work differs from the existing approaches based on models in the fact that the design of test models is completely independent of UML or any other of its associated DSLs. The models proposed are based on a different modeling language, which involves abstract test concepts in order to increase the number of potential test designers. This is possible since models are independent from any testing platform, so no technical knowledge is required from users. The resulting models can be also automatically derived to executable tests for specific testing platforms, once again without requiring technical knowledge from the final user. In summary, the management of abstract concepts, the absence of technical knowledge, and the simplicity and reduced size of the modeling language, compared to UML, can help to reduce the learning curve and increase the potential users of the framework.

Considering model-based testing tools which include the design and execution of tests, these tools are mostly proprietary, being difficult to adapt or extend, and being limited to manage specific testing platforms for execution. On the other hand, open source testing solutions are usually incomplete in terms of functionality. In this proposal, the entire testing process, based on the design, derivation and execution of tests, is integrated in an open source platform. This platform can be easily extended to manage a large variety of testing platforms according to user's needs.

III. Testing framework: Introducing execution flows

The testing framework proposed in this work is outlined in this section, focusing on the way test models can be enhanced by adding execution flows which allow testers to select, from test models, the elements which will be executed on a platform and the flow that these elements will follow during execution.

The aim of the proposed testing framework is to isolate testers from specific platforms as much as possible, allowing them to focus on the design of the tests by means of modeling techniques. The framework will provide the appropriate mechanisms to create test models, and transform them into executable tests with the minimum interaction of the user. Figure 1 summarizes the testing framework developed.

A. Framework overview

The core of the framework relies on the testing modeling language described in [17], which covers abstract concepts related mainly to structural and basic behavioural aspects neces-

 ISBN: 978-1-61208-458-9

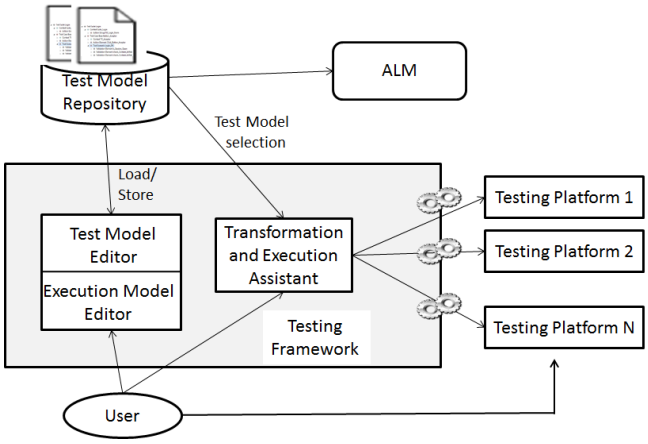


Fig. 1. The proposed framework allows users to focus the effort on the design of the test, being agnostic about any testing platform.

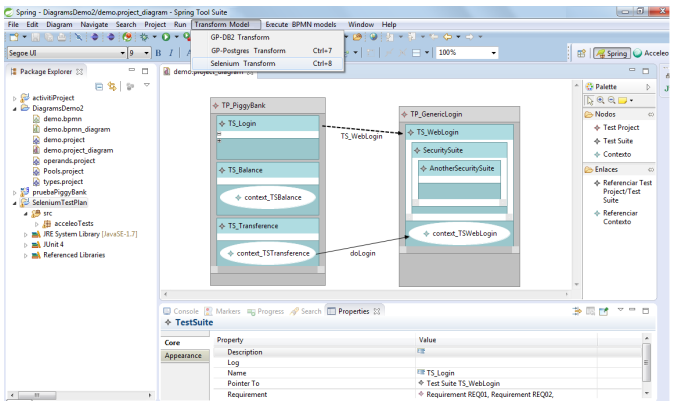


Fig. 2. Example of the developed Test Model Editor building a test model.

sary to design tests in a platform-independent way. An important part of the language is related to the reuse of test elements, structural and behavioural, in order to reduce design time avoiding repetitions and minimizing mistakes or omissions due to those repetitions. This reuse is carried out by pointing directly to the wanted elements (structural or behavioural). A specific GMF (Graphical Modeling Framework)[18] editor, Test Model Editor, encapsulates the language and allows to build the test models graphically. The way the test elements are graphically represented is related to their definition in the defined modeling language. Thus, structural elements are components of different complexity that can be nested in order to provide the test plan with a hierarchy of test elements. An example of structural elements being graphically represented in the editor can be seen in Figure 2. Behavioural elements are inside structural elements and represent basic execution units which will have a direct translation in each target testing platform. Sequences of these elements provide the behaviour of the tests.

Once models are created and the user wants to get executable tests, it is time for transforming the model. The user selects the target platform for executing the tests and the assistant for that platform starts working. This assistant applies a collection of transformation rules, which automatically translate the initial agnostic model into tests which can be executed in the chosen platform. The independent nature of the test models and the existence of specific transformation assistants let the same model be executed in several testing platform, saving time at test design.

Finally, the test models designed can be stored in a regular repository in order to be accessible from the testing framework and from outside. Inside the framework, the models can be accessed from the editor and the transformation assistants for modifications or transformations respectively. From outside, the repository could be accessed by Application Lifecycle Management (ALM) tools for supervision activities. These ALM tools, such as HP-ALM[19] or IBM Rational Team Concert[20], using suitable automatic interpreters, could get the required information directly from test models to fill in their own structures. In this way, migrations from one management tool to another would require less effort since test models are independent from those tools, remaining unaltered during the entire migration process.

This initial framework has been extended with execution models in order to provide more expressivity to the test models. These models are also platform-independent and allow to select which elements from a specific test model will be executed and establish their execution flow. The built of such models is enabled by an Execution Model Editor, while the transformation of these models into executable flows is performed by specific assistants, similarly to the way tests models are managed.

In both cases, the assistants bridge the gap between the agnostic models and the chosen target testing platform, and hide the complexity of the platform. They are also responsible for retrieving relevant information during the execution of the test elements and provide appropriate reports to users.

B. Execution modeling language

The test modeling language used in this framework allows to design hierarchies of structural test elements as a way to organize the test model. However, the language does not provide a way to select which of the test elements will be finally executed in the target platform and the order of that execution. Thus, the user will have to rely on the features included in the target testing platform related to the specification of execution order, if any. This is an important feature since all the executable tests do not need to be executed at once and it is usually interesting to have different execution flows during the testing process. Besides, all the testing platforms do not provide the option to create these flows.

The functionality of the proposed framework has been extended to include the design of execution flows. The aim is to provide more expressivity of test models without increasing the complexity of the test modeling language. For this reason, a new language has been introduced in the framework, the Business Process Model and Notation (BPMN) [21], giving rise to execution models which are still independent from any target testing platform.

 ISBN: 978-1-61208-458-9

BPMN is a standard defined by Object Management Group (OMG) and originally developed to provide a modeling notation comprehensible for all business users, from purely business people to analysts and technical developers. Nowadays, BPMN is widely used in academia and industry due to the large set of concepts managed, independent from any specific domain, and its flexibility to be adapted to other scenarios different from business. This flexibility is due to the fact that BPMN enables the extension of its specification in order to include custom concepts, which represent characteristics of particular domains. The standard is also interesting in terms of execution, since there is a variety of runtime execution engines, which allow the execution of BPMN models almost effortless.

Although the number of concepts managed in the standard is huge (activity, flow, event, gateway, etc.), in this work, we only make use of a subset of them. The chosen elements are available through the developed GMF Execution Model Editor, represented in Figure 1. This editor acts as a filter for the allowed elements, since the standard has not been cut back. The number of BPMN elements managed in the initial subset can be easily increased by simply updating the editor.

Making use of the extensibility feature existing in BPMN to adapt the standard to particular domains, it has been possible to link specific elements in BPMN models to elements defined in agnostic test models. The BPMN element chosen to establish this connection has been the *ServiceTask* element. It represents atomic activities in a process flow, in particular, activities which can be seen as an individual service and can be automated. This description fits well with the purpose of the proposed methodology.

An example of the way these *ServiceTasks* are linked to agnostic test models can be seen in Figure 3, where each of the *ServiceTasks* included in the execution model is pointing to structural elements of different complexity (*TestSuite*, *TestProject*, *TestCase*, etc.) in the test model. Thus, in the figure, the execution model establishes that among all the structural elements existing in the test model for PiggyBank (a simplified online banking application used as case study in this research), only the *TestProject Login* and the *TestSuite Balance* will be executed, showing also the execution order of those elements. The rest of the elements existing in the test model are not required for execution and then, they are not included in the execution model.

An actual example of the Execution Model Editor related to PiggyBank is shown in Figure 4, pointing out the way test elements are linked to BPMN elements.

The execution model shown in Figures 3 and 4 represents a very simplified flow using BPMN. However, the number of concepts and features defined in BPMN, combined with the different concepts defined in the test language allow to introduce different levels of expressivity to the proposed framework.

A first type of expressivity included in the framework, shown in the figures, allows to link *ServiceTasks* only to structural test elements - TestProjects, TestSuites, etc. -. In

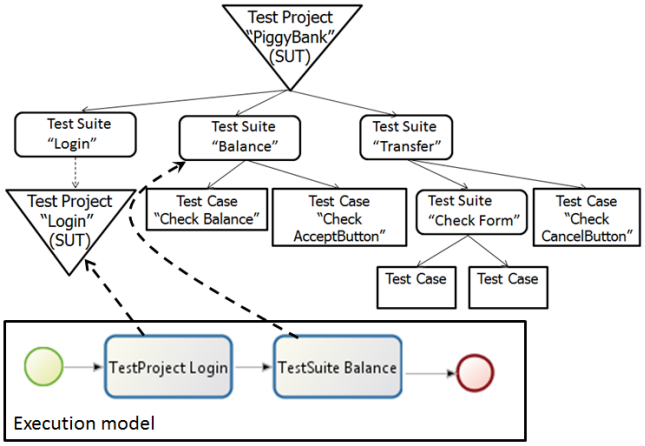


Fig. 3. Extended BPMN specification in order to link to test elements.

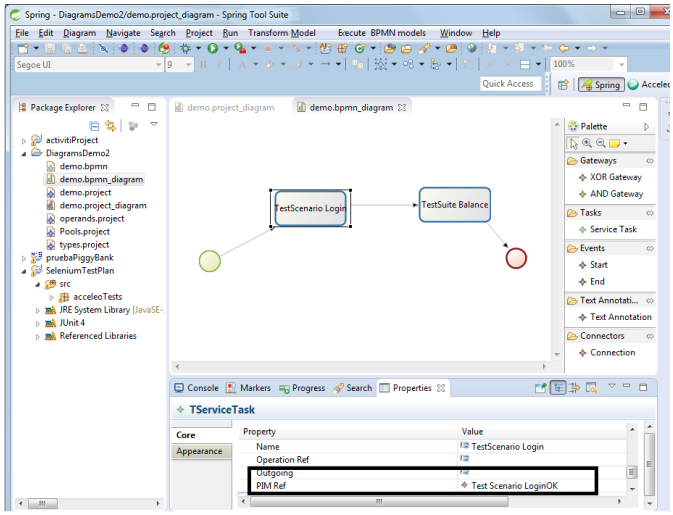


Fig. 4. Example of the developed Execution Model Editor building an execution model related to Piggybank.

these BPMN models, the test elements linked are independent among them in the flow, i.e., the failure of a *ServiceTask* does not imply the suspension of the remaining elements in the model, unless there is a branch in the flow considering that case. In these models, conditional flows can depend on the result of the execution of *ServiceTasks*, seeing this execution as a black box which generates a single result for each *ServiceTask*, independently of the structural test elements involved.

IV. From Modeling Languages to Testing Platforms: A Practical Case

This section shows the way all the elements existing in the proposed framework work together in order to evolve from agnostic test and execution models to executable tests in different platforms.

As shown in Figure 1, users start designing test models for specific applications, ignoring the final platform where tests will be finally executed. These platform-agnostic test models can be enriched with BPMN models, which help users to

 ISBN: 978-1-61208-458-9

define the execution flow of test elements defined in test models. Given the agnostic approach considered in the modeling languages and editors involved in the proposed framework, users can completely ignore any execution platform during the design of test models and execution flows. Only when users want to turn a model into executable tests, test execution platforms are required. Even then, users do not need any technical knowledge about the target platform or the way final tests are built. When a target platform is selected from the proposed framework, the transformation and execution assistants are the only elements responsible for automatically generating tests which are executable in the chosen platform. Both, test models and execution flows, can be transformed to be executed in any available platform.

The mechanisms used to perform the transformation into executable tests are explained next. Before that, the testing platforms managed in the proposed framework are explained.

A. Selenium and GP: testing platforms as case study

Two testing platforms with different characteristics have been considered for test execution. Both suitable for testing web applications, since this is the kind of applications managed in this work. The considered platforms are: Selenium[3], as open source solution; and GP, a proprietary tool suite developed by Santander Group[22].

- Selenium is the selected platform for testing web applications using an open source solution. It lacks of a way to organize tests, being the tester entirely responsible for providing a structural organization. Tests for execution can be written entirely in Java using Selenium WebDriver [23] or as a combination of JUnit tests and Selenium Server [3]. In this proposal, we use the second option in order to simplify the generation of the tests files.
- GP is a proprietary testing tool, which integrates different open source technologies in a common working environment in order to homogenize their usability. It allows the design and automatic execution of tests for web and Eclipse-based applications. Selenium and SWTBot are some of the technologies included.
  From the structural point of view, GP manages elements which provide hierarchical information to organize the tests, allowing individual tests and collections of tests. According to the functionalities included in the platform, it provides a repository of basic behaviours, which can be grouped appropriately to design the each individual test. The technology involved in these behaviours is completely transparent to testers.
  The internal structure of GP tool suite is based on a database technology as storage mechanism for all the elements managed in the platform. This approach can be also found in other testing platforms, such as HP QuickTest Professional (QTP) [1] or IBM Rational Functional Tester (RFT) [24].

Since both platforms build and organize tests in a different ways, the approach to build the transformation engines needed in each case will be also distinct.

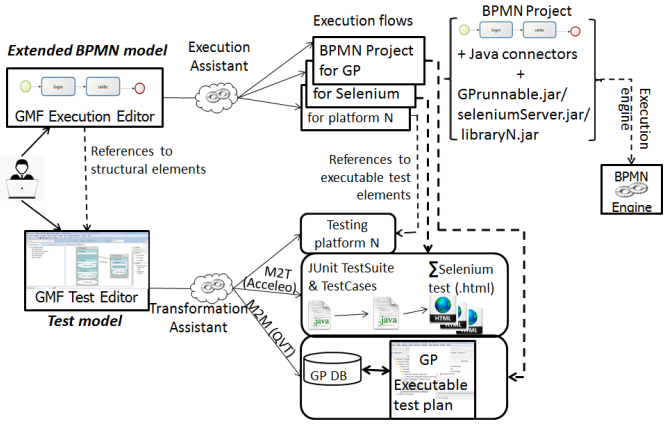


Fig. 5. Transformation and execution assistants link models to testing platforms.

B. Getting executable tests

Following the proposed workflow, and assuming that test models are ready to be transformed into executable tests, users have two ways to continue working. Both are based on transformation and execution assistants, whose role in the testing framework is summarized in Figure 5. This figure also sums up the different technologies and languages used in the implementation of the developed prototype.

1) Transform an entire test model: The user can opt for transforming an entire agnostic test model into a collection of equivalent tests to be executed in a specific testing platform. In this use case, generically represented in the lower part of Figure 5, the user selects the agnostic test model that needs to be transformed and decides the specific testing platform among the options where tests will be executed. Having this information, the assistant delegates the task to the specific transformer for the chosen platform. This assistant provides flexibility to the proposed framework, isolating the editors and users from any testing platform.

Each transformer knows the internal structures of its target platform and the agnostic testing language, establishing the correspondence between the elements in the target platform and the agnostic elements. The existence of this correspondence makes possible that, in an automatic way, the transformer applies to the test model the appropriate rules to generate an equivalent collection of tests in the target testing platform. In this process, the elements existing in the test model (structures, behaviours, data, etc.) are mapped into their corresponding elements, creating all the necessary structures to reproduce the test model in the executable target platform. The transformer not only translates test models, but also interprets them. This is specially relevant referring to data, which potential complexity (unique/multiple values, complex structures, etc.) can generate a single test or a collection of tests as a result of all the possible combinations.

The implementation of each transformer will depend on how the target platform can be represented. If it can be represented as an Eclipse Modeling Framework (EMF) model, model-to-

 ISBN: 978-1-61208-458-9

model transformations can be performed. Otherwise, model-to-text transformations or other alternatives will be necessary. Considering GP tool, based on a database, the tool can be represented by its own modeling language inferred through the database schema. According to this, the content of the database in a particular moment is the state of the system, a state that can be represented as a model. This model enables the use of model-to-model transformations to implement the transformer to GP, using the operational Query/View/Transform (QVT) language. Instead, transforming to Selenium requires model-to-text transformations, using Acceleo, due to the lack of a modeling language to represent Selenium and JUnit.

2) Execute a BPMN flow: Users that want to focus just on the execution of a portion of an agnostic test model can design execution flows (upper part of Figure 5). Before executing the flow, the testing platform where tests will be executed has to be selected. Similarly to test models, BPMN models are not executable straightaway, being also necessary an assistant. When the assistant knows the target platform, it delegates the task to the appropriate interpreter, which will check whether the agnostic model associated with the BPMN model has already been transformed and exists in the chosen platform. Otherwise, the corresponding transformation assistant will be performed first. After that, the interpreter will face two tasks before triggering the execution of the BPMN model automatically. First, connecting the BPMN model to a BPMN engine, which reads, interprets and executes the model. This connection requires a suitable infrastructure (called BPMN project in Figure 5), which varies depending on the chosen BPMN engine. Second, since *ServiceTasks* in the proposed extended BPMN models point to agnostic test elements, it is mandatory their translation to references to executable tests which exist in the target testing platform. For this second task, it is worth to mention that during the transformation process of a test model, some extra data can also be generated if needed. The need for this auxiliary information depends on how costly is to associate in a BPMN model an agnostic test element to its executable tests once the test model has been transformed. This association can be based on one-to-one relationships, but it is mostly based on one-to-many relationships because of two reasons. First, since the structural element referenced in a *ServiceTask* can be a container of other tests elements, the BPMN engine will need to call for execution to each of the individual tests that the transformation process generates from the container element. For instance, *ServiceTask* in Figure 3 is referencing the complex element *TestProject Login*, represented in Figure 6.a. All the tests contained need to be known. Second, the agnostic models manage collections of data for specific inputs of the tests. These collections give rise to a set of individual tests where each input only has one value associated with it. This is the case represented in Figure 6.b, where one single test element, *TestCase Button_Accept*, is transformed into a variable number of executable tests depending on the data defined in it.

According to this, executing BPMN models in the GP platform, requires an intermediate model as extra information in order to associate the structural tests elements in the agnostic model to the identifiers of their corresponding executable tests generated in the target platform. However, using Selenium as target platform does not require extra information, since the association between test elements and executable tests is easier to obtain.

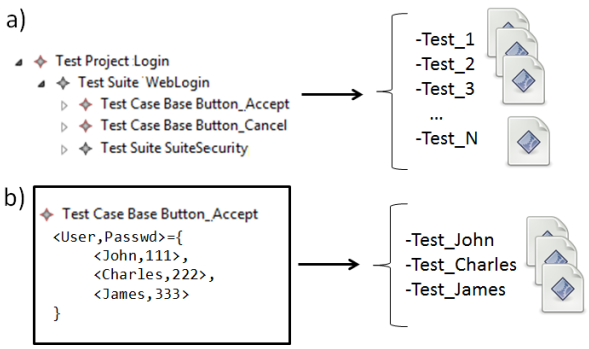


Fig. 6. Model-to-model transformations usually turn into one-to-many relationships that are managed through the intermediate model.

Once the BPMN model points to executable tests, it is time to establish the infrastructure to make the flow executable. This infrastructure depends exclusively on the BPMN engine chosen for execution. In this work, it has been used the BPMN engine provided by Activiti [25], which requires an Eclipse project to execute the BPMN models. Apart from the designed BPMN model, the project must contain information to deploy it for execution and set up the BPMN engine. It also includes the way to implement the execution of the *ServiceTasks* existing in the BPMN model. This implementation codifies the way all the executable tasks associated to a particular *ServiceTask* are identified and called for their automatic execution one by one. In this process, it is used any extra information that might have been generated during the transformation of its corresponding test model. As can be seen in Figure 5, the execution of the tests is provided by platform-dependent libraries (GP runnable, selenium-server), which allow an automatic interaction with each platform. The Activiti-based project is the output of the interpreter when a user wants to execute a BPMN model, and the rules to generate the involved Java classes are based on model-to-text transformations, using as model the test model itself (for Selenium platform) or the intermediate model (for GP platform). Once the building of the Eclipse project is complete, the interpreter initiates its execution, triggering the BPMN engine which reads the BPMN model, interprets each one of the existing elements, and performs the actions associated with each one. When the BPMN engine finds a *ServiceTask*, all its associated tests are launched for execution before continuing with the flow.

V. Conclusion and Future Work

The proposed model-based testing framework, the prototype of which has been entirely implemented, provides a infrastructure to design graphically test and execution models

 ISBN: 978-1-61208-458-9

independent from any testing platform. The execution models represented as BPMN flows, increase the expressivity of the test models by extending BPMN elements to include custom concepts related to the testing language.

The proposed framework allows a quick starting of the testing phase, being able to work in parallel to the development process, and a quick adaptation to changes in the requirements. It also enables the reuse of test elements among models. The models are only attached to an application, but can be applied to very different platforms for test execution due to the assistants provided in the framework, which automatically transform models to executable tests in each particular platform. The use of graphical editors and assistants allow to hide the complexity of testing execution platforms, increasing the number of potential users of the framework.

As future work, the expressivity of the tests will be increased by including more BPMN concepts to the ones already managed by the editor, and the implementation of complex behavioural elements based on BPMN flows. The platforms managed currently in this framework are Selenium as open source solution, and GP as proprietary tool. Other platforms can also be integrated building their corresponding assistants. Thus, the integration of tools, such as HP Quality Center (HP QC), also database-based as GP platform, could be affordable in the near future.

ACKNOWLEDGEMENT

The work for this paper was partially supported by funding from ISBAN and PRODUBAN, under the Center for Open Middleware initiative [26].

REFERENCES

[1] A. Rao, HP QuickTest Professional WorkShop Series: Level 1 HP Quicktest. Outskirts Press, 2011.
[2] P. Tahchiev, F. Leme, V. Massol, and G. Gregory, JUnit in Action. Greenwich, CT, USA: Manning Publications Co., 2010.
[3] A. J. Richardson, Selenium Simplified: A Tutorial Guide to Selenium RC with Java and JUnit. Compendium Developments, 2012.
[4] A. Huima, "Implementing Conformiq Qtronic," in TestCom/FATES, ser. Lecture Notes in Computer Science, A. Petrenko, M. Veanes, J. Tretmans, and W. Grieskamp, Eds., vol. 4581. Springer, 2007, pp. 1–12.
[5] Hewlet-Packard Company, "HP Functional Testing," 2012.
[6] P. Baker et al., "The UML 2.0 Testing Profile," Sep. 2004.
[7] J. Iber, N. Kajtazovic, A. Holler, T. Rauter, and C. Kreiner, "Ubtl - UML Testing Profile based Testing Language," in Model-Driven Engineering and Software Development (MODELSWARD), 2015 3nd International Conference on. SciTePress, February 2015, pp. 99–110.
[8] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.
[9] M. Cristiá and P. R. Monetti, "Implementing and Applying the Stocks-Carrington Framework for Model-Based Testing," in ICFEM, 2009, pp. 167–185.
[10] L. C. Briand and Y. Labiche, "A UML-Based Approach to System Testing," in Proc. of the 4th Int. Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. London, UK, UK: Springer-Verlag, 2001, pp. 194–208. [Online]. Available: http://dl.acm.org/citation.cfm?id=647245.719446
[11] C. Pedrosa, L. Lelis, and A. Vieira Moura, "Incremental testing of finite state machines," Software Testing, Verification and Reliability, vol. 23, no. 8, 2013, pp. 585–612. [Online]. Available: http://dx.doi.org/10.1002/stvr.1474
[12] K. Karl, "GraphWalker," URL: www.graphwalker.org [accessed: 2015-12-18].
[13] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
[14] B. P. Lamancha, P. R. Mateo, I. R. de Guzmán, M. P. Usaola, and M. P. Velthius, "Automated Model-based Testing Using the UML Testing Profile and QVT," in Proc. of the 6th Int. Workshop on Model-Driven Engineering, Verification and Validation, ser. MoDeVVa '09. New York, NY, USA: ACM, 2009, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/1656485.1656491
[15] M.-F. Wendland, A. Hoffmann, and I. Schieferdecker, "Fokus!MBT: A Multi-paradigmatic Test Modeling Environment," in Proc. of the Workshop on ACadeMics Tooling with Eclipse, ser. ACME '13. New York, NY, USA: ACM, 2013, pp. 1–10. [Online]. Available: http://0-doi.acm.org.cisne.sim.ucm.es/10.1145/2491279.2491282
[16] P. Baker et al., Model-Driven Testing: Using the UML Testing Profile. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.
[17] C. Sanz et al., "Automated model-based testing based on an agnostic-platform modeling language," in Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2015, pp. 239–246.
[18] Eclipse(b), "Eclipse Graphical Modeling Framework (GMF) Tooling," URL: http://www.eclipse.org/gmf-tooling/ [accessed: 2015-12-18].
[19] Hewlet-Packard Company, "HP Application Lifecycle Management," 2011.
[20] International Business Machines Corp., "IBM Rational Team Concert," 2008.
[21] OMG, Business Process Model and Notation (BPMN), Version 2.0, Object Management Group Std., Rev. 2.0, January 2011, URL: http://www.omg.org/spec/BPMN/2.0 [accessed: 2015-12-18].
[22] URL: www.santander.com/ [accessed: 2015-12-18].
[23] S. Avasarala, Selenium WebDriver Practical Guide. Packt Publishing, 2014.
[24] C. Davis et al., Software Test Engineering with IBM Rational Functional Tester: The Definitive Resource, 1st ed. IBM Press, 2009.
[25] T. Rademakers, Activiti in Action: Executable business processes in BPMN 2.0, 1st ed. Shelter Island, NY: Manning Publications, 2012.
[26] URL: http://www.centeropenmiddleware.com/ [accessed: 2015-12-18].

 ISBN: 978-1-61208-458-9

Software Based Test Automation Approach Using Integrated Signal Simulation

Andreas Kurtz
BMW Group
Integration Electric/Electronics, Software
Munich, Germany
andreas.kurtz@bmw.de

Bernhard Bauer
University of Augsburg
Institute for Computer Science
Software methodologies for distributed systems
Augsburg, Germany
bauer@informatik.uni-augsburg.de

Marcel Koeberl
BMW Group
Integration Electric/Electronics, Software
Munich, Germany

***Abstract*—Test automation in distributed systems requires new methods in signal simulation for the stimulation of the distributed system. Increasing complexity of electric electronic (E/E) systems increases the testing-effort. The main challenge is to reduce the time spent on manual stimulation of input signals in favor of automated testing. The systems currently used for test automation have to be adapted to each hardware and software version of the system to be tested. The approach shows a software-based automation solution through the integration of a simulation service in the AUTOSAR architecture. By integrating a generic software-based simulation module with an interaction point at the basic software driver layer, the execution of tests can be automated and improved in terms of adaptively and reproducibility.**

Keywords–Automotive; distributed systems; model based testing; simulation; system model; test model.

I. INTRODUCTION

Mastering complexity and customer orientation are challenges in the development of electric electronic (E/E) and software functions in the automotive industry. In current and future vehicles, the increase of the distribution and the networking of functions demands new ways of automation for testing customer features. Software bugs are the main reason for malfunctions in new developed cars [1]. In the automotive industry, the safety requirements are extremely important because of their implications. Therefore, there is a need to err on the side of caution.

This paper focuses on developing a method for test automation of the automotive system model. The system model stands for the total system deemed to be a distributed system with its networked hardware (HW) and software components (SWC). The long-term goal is a solution for automating system model test, at a total system level, with an integrated distributed software-based solution.

The rest of the paper is structured as followes. Section II presents the State-of-the-art model based testing approach and the current realisation level in the automotive industry. Section III describes the over all approach, followed by Section IV with a detailed description of a concrete implementation. In Section V a related approach is shown to point out the difference to the new approach, concluded in Section IV with a short outlook.

A. Problem Statement

The increasing complexity of developed functions with shorter developing time leads to exceeding use of methods. Figure 1 shows the raising demand of using methods when reducing development time to handle equal development effort. In addition to an increase of system complexity the conventional methods have to change to virtual development.

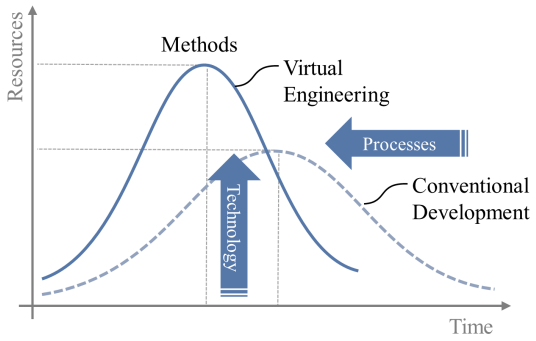


Figure 1. Reducing time demands increase of virtualisation [2]

State-of-the-art software-based automation methods, used in automotive software development, use additional software functions integrated in the software components to be tested. This kind of interaction affects the customer function itself.

These functions, used for virtual input stimulation, interact with the customer functions and require specific solutions for each type of implementation. Using this kind of automation does not reflect the functional behaviour of the system model like in customer usage. Additional signals and interfaces are used to get access to the implemented customer functions with the goal to compare the system model towards specification. Another aspect is the additional software code, needed to realize the interaction. Last, but not least, crosslinked functions are not detected because of the high-level interaction point. Therefore malfunctions in close-by or connected SWCs are not noticed.

A physical simulation of sensor signals at the hardware interface does represent the customer usage but is too expensive to build a specific solution for each HW variation. The main challenge for physical signal simulation is to find

 ISBN: 978-1-61208-458-9

a generic solution for all types of hardware interfaces. The developed method shall use the test model in combination with the AUTOSAR description files, to generate data for stimulation of the system model.

This paper focuses on a method for distributed systems, extracting the needed data out of the test model and the AUTOSAR configuration files. AUTOSAR supports the approach because of the standardised software architecture (Figure 2), giving the chance of developing a generic method.

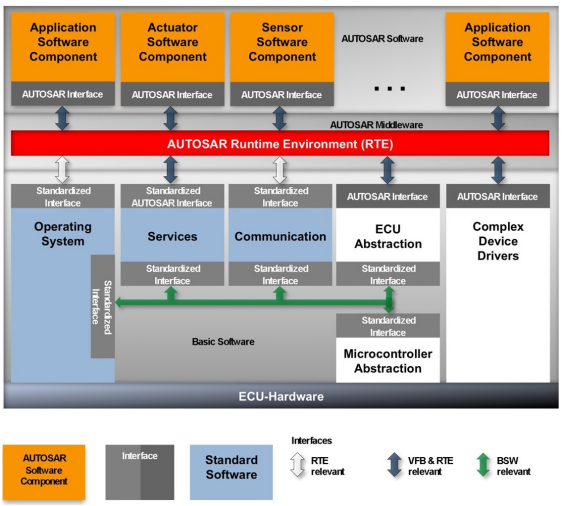


Figure 2. AUTOSAR Architecture, Components and Integration [3]

II. State-of-the-Art

The start scenario to get test cases in model based testing (MBT) is presented in Figure 3(A), with a test designer and its mental test model. The tester, expecting to have the entire test cases, executes the textual test cases. The prospective goal is to reach scenario (C) shown in Figure 3. This enables to compute the test cases for a test automation. By a change, from difficult to understand textual test cases to formal models, we can enable automation methods.

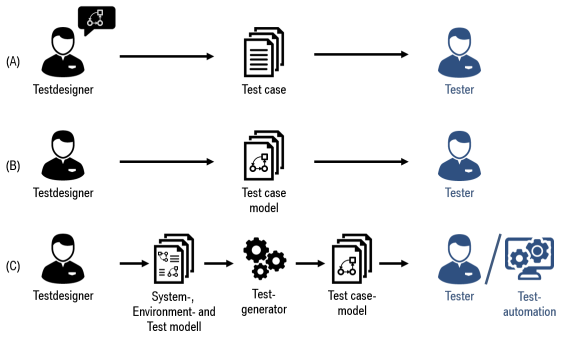


Figure 3. Textual vs. model based specification of tests [4]

With the mentioned focus on the total system as a distributed system, the test case models become complex. Due to the many options of partitioning the variable number of participants (SWCs) to the different hardware options, regarding the standardised architecture framework, the complexity of the models grows.

Model based testing methods are [4]:

- *Component Test* or so called 'unit test', in which the individual program components are tested in isolation.
- *Integration Test* is testing the interaction of several componentes.
- *System Test* is to ensure the operability of the entire system according to the requirements
- *Acceptance Test* is the test under real operating conditions, as well as the interaction of several systems.

At *System Test*, the aim of the approach, the distribution of functions increases automation complexity. In addition, the simulation of input signals is more difficult, due to less accessible interfaces without disturbing the system's behavior.

Figure 4 shows a simplified software architecture with the main layers of AUTOSAR. From bottom up, above the HW there is the basic software (BSW) allocated. The basic software contains modules shown in Figure 2. All communication to the SWCs is distributed via the runtime environment (RTE). Figure 4 shows the same architecture for both cases with two SWCs: SWC1 (and lower layers) representing the human machine interface (HMI), iDrive Controller (ZBE), and SWC2 (and lower layers) representing the Navigation System (Navi) with symbolised tree structure of the menu.

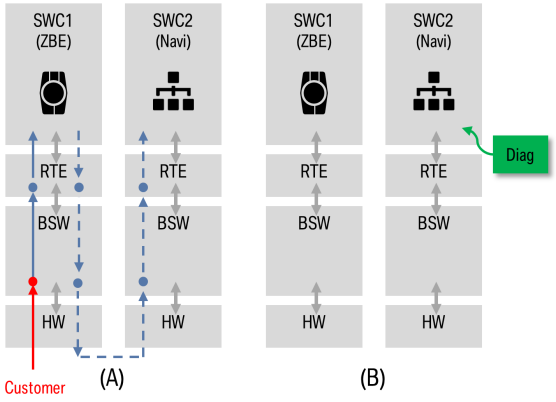


Figure 4. Use Case - SW-Architecture; (A) Customer Use; (B) Simualtion state of the Art; simple draft

With reference to the example, we look closer to the software architecture according to AUTOSAR. Conventional software-based automation methods interfere at application SWC layer, shown in Figure 4(B) named Diag. Usually these functions are integrated for software-based internal error detection and setting data trouble codes (DTC).

Figure 4(A) shows a customer interaction with the total system via the ZBE, e.g., the customer enters navigation destination. Figure 4(B) shows the conventional method with a so-called diagnosis job (Listing: 1) to hit the right menu items. The conventional virtual interaction via diagnosis job interacts in SWC2 and tends to a different behaviour in the total system.

```
1 apiJob(ECU, "steuern\_routine", "ARG;MENU;
    ↳ STR", \%i;\%i;\%i;\%i)
```

Listing 1. Example Diagnosis-Job

 ISBN: 978-1-61208-458-9

This function (Listing: 1) uses a unique identifier (ID) to hit a menu item. A series of these jobs is necessary to reach the same goal (in this case an entered navigation destination), but shows a different total system behaviour. For both cases, we see the same appearance but in Figure 4(B) the communication between SWC1 and SWC2 is missing. The result is that errors in the data communication that occur in the customer case are not detected.

III. Related Work

Model based testing is the basic approach in the automotive product development. Methods for data-flow analysis can help to improve the explained method by computing realistic software-paths and the corresponding data sets. With the help of this information, the test model can be improved to get near 100% test coverage in consideration of the customer use cases. Domain knowledge is the key for computing user realistic software paths. Other paths only have to be checked referring their impact on customer functions.

A existing approach with focus on 'automation, modularization and compatibility of all equipment to do measurement, calibration and diagnosis' [8] is the Can Calibration Protocol (CCP) [8]. The Protocol is used for calibration and data acquisition. Realised as a driver with access to the internal ECU memory this part of the protocol causes additional CPU load. During a session using CCP a 'continuous logical connection' [8] is established to transfer data from the ECU to the master device (off board test automation). This approach interacts at the driver layer. CCP has the main goal of data acquisition in contrast to data simulation.

IV. The Approach

The goal is to compare system- and test model with an integrated distributed software solution to get the system's behavior closer to the system's behavior in customer's use. Figure 5 shows the approach of comparing system- and test model.

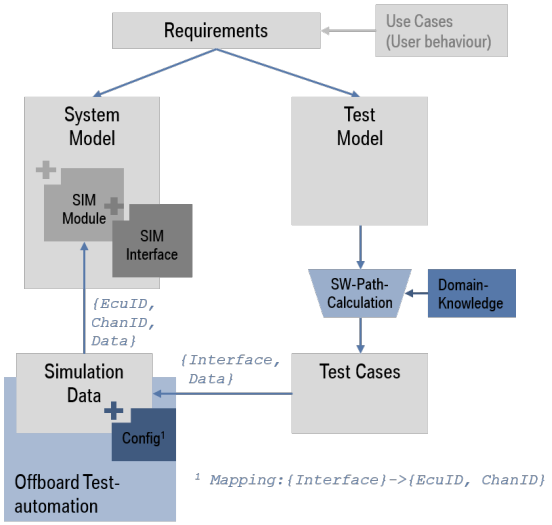


Figure 5. Approach of comparing system- and test model based on a standard model based approach

The methodological approach is to integrate a generic software-based simulation module (SIM Module) and a simulation interface (SIM Interface) in the system model. The novelty of the approach is the layer of intervention (AUTOSAR driver layer) with the associated reduced data complexity. The reduced data complexity is due to the focus on system input signals. In summary, it is a distributed simulation of input signals in a distributed system with an engagement in the basic software driver layer.

– How does this work? – These distributed SIM modules can receive test cases from the off board test automation system and execute the test cases, individually or jointly, by order of the off board system. The data for the test cases is computed out of the test model and transferred in abstract test data and a mapping table (Config). The separation has the advantage of using the test case for different hardware configurations.

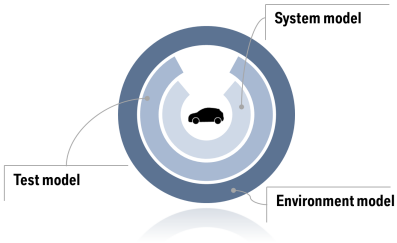


Figure 6. System-, test and environment model [4]

Figure 6 is intended to show that the test model is the combination of the specified system model functional behaviour and the environment model. The environment model includes all external factors e.g., temperature, light or physical characteristics to the system model as well as the customer.

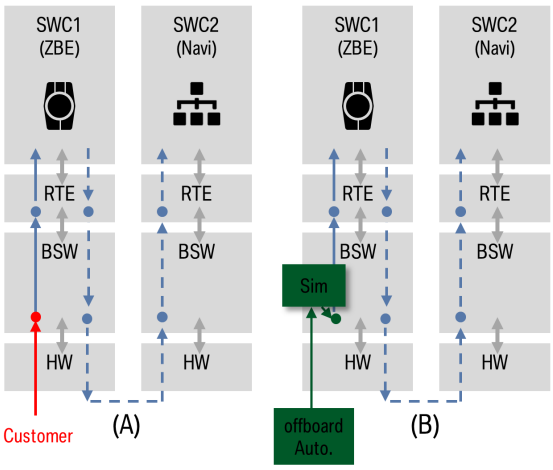


Figure 7. Automation Approach AUTOSAR view. (A) Customer use; (B) simple draw of SIM Interface integration in AUTOSAR Basic Software

The interaction of the simulation, formerly manipulation, at the AUTOSAR basic software interface (Figure 7(B)) allows the SWCs to operate closer to the customer use.

On the one hand, the customer function operates closer to the behaviour in the normal customer usage which leads to detection of errors in the functional implementation or errors between layers. On the other hand, the simulation gets easier

 ISBN: 978-1-61208-458-9

by getting closer to the hardware layer, because of reduced types of signals. There are only discrete signals, because all data processing of customer input ends up in analog digital converter. A positive effect is that in case of simulation, cross-linked functions are also triggered and show their behaviour as well as the misbehaviour.

V. The Details

The methodological approach detailed in Figure 5 shows schematically how the test case data is loaded to the system model. Based on the assumption that a test model is available test cases are derived therefrom. To get only the relevant test cases, we compute theoretical software paths of the test model with the background knowledge of the specific domain (Figure 5).

The separation of the total system in cluster e.g., power-train, power-network, infotainment and other is called domain. We need the domain knowledge to reduce the number of computed paths. It is not meaningful to test all theoretical possible software paths of a single software component at total system level testing. Looking at the distributed system with all of its participants, the software components in cooperation reduce the possible paths. The more components participate the more complex the system gets, but also the functions limit each other e.g., timing, min- and max values. For example when focusing on power-network functions the domain knowledge does have information and boundaries of the power train like the engine speed range. The speed range is decisive for the operating range of the generator and thus for energy availability.

The bigger part of software functions is developed with a model based approach and realised with state machines. The main reasons for the increase of complexity are that on one hand the conditions for triggering the transitions are built in various state machines in different software levels, and on the other hand, almost all of the conditions do have timing constrains. This expands the number of use cases by testing boundary values e.g., lower limit, upper limit, lower limit follower, upper limit follower and last but not least the time steps on the valid timing interval.

Computing the paths of the menu structure has been performed with a data-flow based model analysis [5] [6]. All paths had to start and end in the *main menu* with the requirement that every transition can be passed only once per path to avoid loops. The result is a set of paths with the information of states passed, transitions and trigger for transitions.

After computing all relevant paths of the test model, we can generate test cases, including information as named before. With this information, the idea is to compute a set of all sequences for simulating all user input hardware signals. The computed data is separated in the interface information and the data sequence. This dataset describes the overall customer function input and will be mapped to the ECU and HW-Channel allocation specified in the system model. This configuration database is stored in the off board test automation (Figure 5). With this approach the general customer input information can be mapped to each specific AUTOSAR implementation.

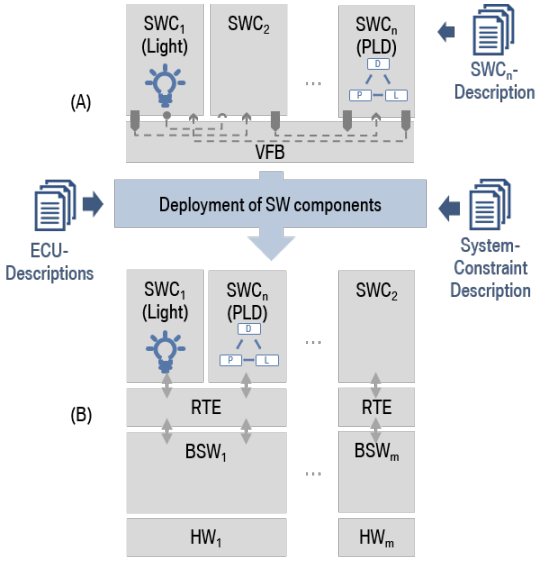


Figure 8. Schematic Diagram of AUTOSAR Configuration files [3]

The configuration *Config* (Figure 5) is an information depending on the specific implementation, computed out of the AUTOSAR description files. These files are generated in the software development process (Figure 8). These files give the specific input of which SWC is mapped to each ECU as well as channel and communication path information. In Figure 8, the shown files contain the following information:

- *Software Component Description*: Describes the functional dependencies
- *System Description*: Describes the partitioning of SWCs to ECUs
- *ECU Configuration*: Describes the signal routing to the HW-Abstraction
- *Basic Software Description*: Describes the mapping of the HW-Abstraction to the HW-Channels

These descriptions are specific for each AUTOSAR software architecture. Therefore, the information depends on the software of the system model. Errors in the configuration files are handed to the test automation.

Figure 9 shows the software architecture solution for the new methodological approach. What is new is that this SIM module is integrated in each ECU, which reads sensor signals. The SIM Module represents the merger of the following three components:

- *SimAgent* is the logical component, including a state management and is responsible for the execution of the sequences, data storage and safety requirements,
- *SimGW* does only route signal data to the Microcontroller Abstraction Layer (MCAL),
- *DioSim* is the interface to the existing driver (DIO) and its read services with the goal to replace the physical signals with the simulated in case of an active simulation.

 ISBN: 978-1-61208-458-9

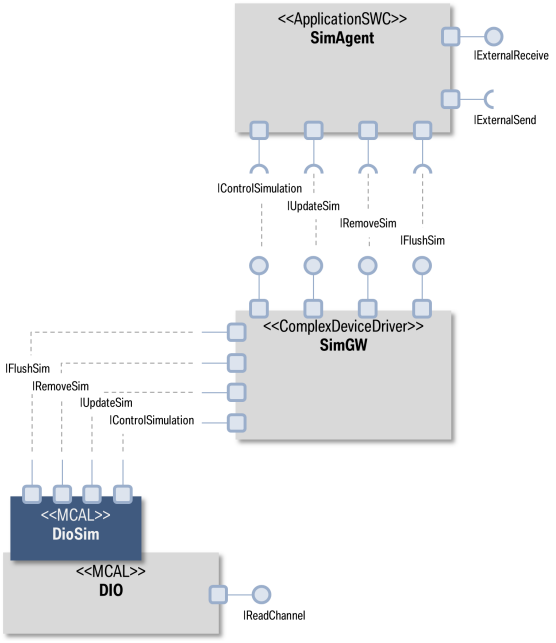


Figure 9. Software Architecture of Simulation module

The SimGW module is used to route the signals from the SimAgent to the DioSim. This is a temporary solution that we do not have to edit the I/O-Abstraction of the existing AUTOSAR basic software. Above the RTE there will be the SimAgent as a part of the *BMW System Function Software Components* next to the normal application software components (Figure 10). *BMW System Functions* are standardised software components that are integrated into each ECU, e.g Diagnosis- or DTC-Functions.

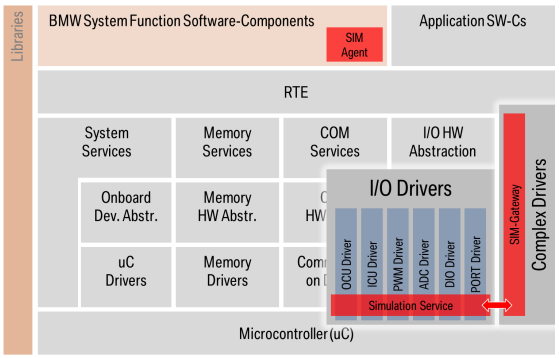


Figure 10. Integration of Simulation Interface in AUTOSAR Driver Layer

– The simulation process. – Enabling of the simulation will follow the sequence shown in Figure 11.

- *Init()*, activating the SimAgent via diagnosis job (CAN-Message),
- *FlushSim()*, erasing the existing data in the memory,
- *UpdateSim()*, setting initial values and parameters, like start value and start time,
- *EnableSimulation()*, activating the simulation service.

After enabling the simulation service, the *UpdateSim()*-Function is used to feed the DioSim-module with the data during the simulation.

The novelty here is that a test case is temporarily stored in the ECU memory and is executed by the SIM Agent. Each SIM module has to keep only the simulation data necessary for the ECU specific simulation to low memory requirements.

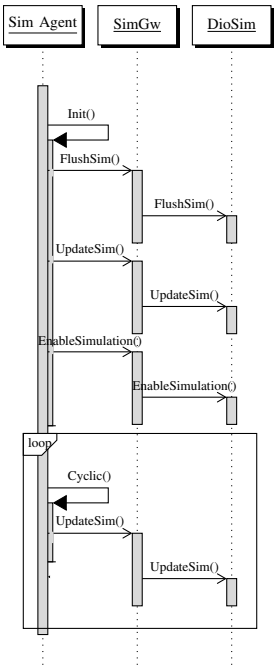


Figure 11. Process Start of Simulation Service [7]

Figure 12 shows the chain of reaction through the AUTOSAR basic software when a SWC requests data from the RTE. If there is a request of a RTE variable, the normal chain of reaction will be triggered. If the request reaches the Dio-module the Dio-module will check if there is a simulation active for the requested channel (*GetSimState()*) and will switch to the simulated data if required. In all other cases the physical state of the hardware I/O will be read.

The additional function request *GetSimState()* will be integrated in the Dio-module and will have an insignificant influence on the time response of the request. There will be no difference between the time response in normal customer use or in simulation usage, because in both cases the new additional function request will be triggered.

The worst case execution time analysis (WCET) calculates a percentage increase of the processor about 3,3%. This is a calculation for 255 simultaneously simulated channels, on an 80MHz CPU with a task cycle of 1ms and in case of the non-existence of simulation data. In this case the software reads, after failure of reading a simulated value, the physical value of the hardware I/O.

For the research an 80MHz CPU with a 12.5ns Assembler instruction execution time is used. This CPU is used to calculate execution times for 1ms tasks. The first one is with no simulation active and the second one is with simulation active without data:

 ISBN: 978-1-61208-458-9

- *DioSim_GetSimState=0* - [normal case]
  & no active Simulation
  WCET = 12,5ns according to 1 function call
- *DioSim_GetSimState=1* - [worst case]
  & no Simulation Data
  & Reading real data
  WCET = 32950ns

An impact could be a time delay being critical for a SWC. In normal case the SWC runs in tasks about 10ms and the delay will be about less than 0,01%.

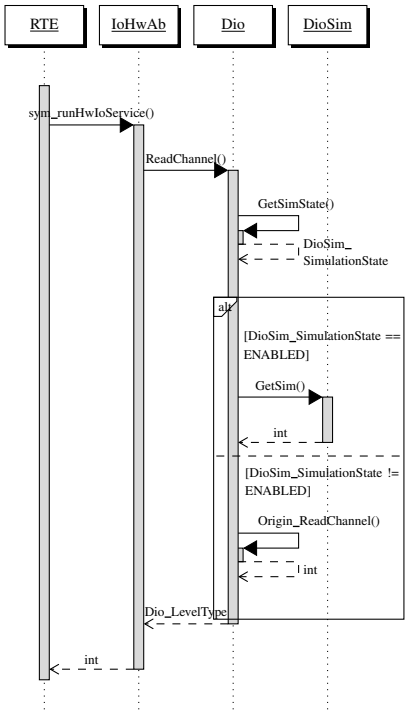


Figure 12. Simulation Service process [7]

Savety functions such as stopping and fall-back mechanisms have been integrated in the SimAgent. This allows to abort the simulation at any time in two ways: delayed or immediately. 'Delayed' for a smooth end of the simualtion. 'Immediately' in case of a system malfunction.

VI. Conclusion

The method of virtual hardware signal simulation, with an integrated software approach, allows to automate the user input analogous to the customer use cases and thereby to compare system- and test model in an innovative way. The approach shows a different solution with no need of special hardware equipment because of the integration of the simulation in the distributed system as a software-based distributed system. The methodology is realised as a standard module for easy integration in the AUTOSAR BSW to get an interface for the automation. The key aspect of this approach is the point of interaction located in the BSW driver layer. The methodology uses an abstraction to specific hardware input signals via mapping to reduce data and to keep the simulation module as generic as possible.

A simple generic simulation module controls the simulation process. Because of its simplicity, the simulation has a barely measurable effect on the CPU workload. The system reaction, respectively the system interaction with the customer and environment will be evaluated with proven and tested methods already in use. Therefore, there is no need in building up new evaluation methods and systems for the system analysis.

The new approach has a substantial similarity to the CCP approach in the connection layer: both interact at the driver layer. The enormous difference between both is the cutting of the data communication to the SIM Agent (slave), during simulation. This reduces CPU workload and the new approach is enables a simulation, much closer to the customer case.

Next steps for the implementation are to check the data size of the simulation sequences, especially for long-term simulations, beacuase memory space in automotive ECUs is scarce. The memory space in the automotive ECUs is associated with high cost hardware, and therefore the main constraint on the test case steps and the number of parallel simulated channels.

References

[1] Basycon Unternehmensberatung GmbH. (2006) Softwarequalität durch verbesserte Entwicklungsprozesse. [Online]. Available: http://www.basycon.de/de/web/basycon/publ_typ_poster [Jan. 5, 2016]

[2] M. Eigner and R. Stelzer, *Product-Lifecycle-Management: Ein Leitfaden für Product-Development und Life-Cycle-Management*, 2nd ed., ser. VDI. Berlin and Heidelberg: Springer, 2013.

[3] AUTOSAR Partnership. (2014) AUTOSAR Components. [Online]. Available: http://www.autosar.org/fileadmin/images/media_pictures/AUTOSAR-components-and-inte.jpg [Dec. 15, 2015]

[4] T. Roßner, *Basiswissen modellbasierter Test*, 1st ed. Heidelberg: dpunkt.verl., 2010.

[5] C. Saad and B. Bauer, Eds., *Model-Driven Engineering Languages and Systems: Data-Flow Based Model Analysis and Its Applications*. Springer, 2013.

[6] C. Saad and B. Bauer, Eds., *Industry Track of Software Language Engineering (ITSLE), 4th International Conference on Software Language Engineering (SLE 2011)(May 2011): The Model Analysis Framework An IDE for Static Model Analysis*, 2011.

[7] M. Köberl, "Integration softwarebasierter Automatisierungsmethoden in eine Test-ECU," Master's thesis, University of Augsburg, Augsburg, 2015.

[8] H. Kleinknecht, A. Krüger, H.-G. Kunz, R. Maier, H. Schröter, and R. Zaiser. (1999) Can Calibration Protocol - Version 2.1.

[9] AUTOSAR Partnership. (2014) AUTomotive Open System ARchitecture: Enabling Innovation. [Online]. Available: http://www.autosar.org/ [Dec. 15, 2015]

[10] B. Beizer, *Software testing techniques*, 2nd ed. New York: International Thomson Computer Press, op. 1990.

[11] D. W. Hoffmann, *Software-Qualität*, ser. EXamen.press. Berlin and Heidelberg: Springer, 2008.

[12] M. Pezzè and M. Young, *Software testing and analysis: Process, principles, and techniques*. [Hoboken and N.J.]: Wiley, ©2008.

[13] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*, 3rd ed. Hoboken and N.J: John Wiley & Sons, ©2012.

[14] R. Seidl, M. Baumgartner, and T. Bucsics, *Praxiswissen Testautomatisierung*, 1st ed. Heidelberg and Neckar: dpunkt, 2011.

[15] S. Byhlin, A. Ermedahl Jan Gustafsson, and B. Lisper, "Applying Static WCET Analysis to Automotive Communication Software."

[16] Vector Informatik GmbH, "AUTOSAR Configuration Process - How to handle 1000s of parameters: Webinar 2013-04-19," 2013.

[17] H. Balzert, *Lehrbuch der Softwaretechnik/2: Software-Management*, 2nd ed., ser. Lehrbücher der Informatik. Heidelberg: Spektrum Akad. Verl, 2008.

 ISBN: 978-1-61208-458-9

Medical Mobile Apps Data Security Overview

Ceara Treacy, Fergal McCaffery
Regulated Software Research Centre & Lero
Dundalk Institute of Technology,
Dundalk, Ireland
e-mail: {ceara.treacy, fergal.mccaffery }@dkit.ie

Abstract— **In the growing industry of mHealth, mobile medical apps are becoming a popular mechanism for healthcare delivery. Characteristically, these apps are designed to both process and transmit data that is sensitive medical data. Such data is required to be kept private and secure through regulations and legislation. The detections of increased app hacking by security companies and researchers are especially significant amidst today's rapid growth in healthcare mobile apps. Consequently, security and integrity of the data associated with these apps is a growing concern for the app industry, particularly in the highly regulated medical domain. Until recently, data integrity and security in transmission has not been given serious consideration in the development of mobile medical apps. There are currently no procedures or standard practices for developers of mobile medical apps to assure data integrity and security in transmission. This paper is an overview of existing mobile medical apps data security issues and security practices. We discuss current regulations, standards and best practices concerning data security in mobile medical apps. The paper introduces the concept of a process model and testing suite to assist mobile medical app developers to implement data security requirements to assure the Confidentiality, Integrity and Availability of data in transmission.**

Keywords-Mobile Medical Apps; data security; regulations; data security testing.

I. INTRODUCTION

In mHealth, mobile apps are generally classified into mobile health/wellbeing apps (MHAs) and mobile medical apps (MMAs). A MMA is an app that qualifies as a medical device and is therefore required to follow the applicable medical device regulatory requirements. Medical professionals and the general public use mobile apps to perform many tasks, such as: health and fitness tracking, sharing medical videos, photos and x-rays; blogs to post medical cases and images; share personal health information; and keep track of alerts on specific medical conditions and interests [1]. MMAs are evolving quickly with the processing capabilities of mobile devices. The use of mobile apps enables dynamic access to personal identifiable information and the collection of greater amounts of sensitive data relating to personal health information (PHI). The use of mobile apps implicates changes in the way health data will be managed, as the data moves away from central systems located in the services of healthcare providers, to apps on mobile devices [2]. Increasing reliance on mobile apps raises questions about compromised patient privacy [3] and security of the data accompanying the apps [2]. The PwC's Health Research Institute's survey claims 78% of surveyed consumers were worried about medical data security, while 68% were concerned about the security of their data in mobile apps [4].

The impact of data breaches in the medical industry is far-reaching in terms of costs, losses in reputation [5] and potential risk to patient safety. Reasons for obtaining access to PHI can be for monetary aim, harmful and personal intention [6]. An example of the importance of cybersecurity can be seen with the health insurer Anthem in the US. A reported breach involved hackers obtaining personal identifiable information and PHI for about 80 million of its customers and employees [7]. The information stolen falls under the Health Insurance Portability and Accountability Act (HIPAA), which is the federal law governing the security of medical data and could result in fines up to $1.5million. A data breach that maliciously makes changes to a medical diagnosis or prescribed medication has serious consequences in terms of physical harm and patient safety. With PHI breaches, either through physician diagnosis or a treatment plan, the possibility of personal harm or loss is pronounced.

The Food and Drug Administration (FDA) regulates medical devices in the U.S and are alert to the cybersecurity of medical devices. In July 2015, the FDA issued a cybersecurity alert to users of a Hospira Symbiq Infusion System pump, where it strongly recommended discontinued use, as it could be hacked and dosage changed [8]. In September 2015, the FBI issued a cybersecurity alert, outlining how Internet of Things (IoT) devices may be a target for cybercrimes and may put users at risk [9]. If a cyber-thief changes patient medical information or a physician diagnosis, serious medical harm or even death can result. An article that references the DarkNet, describes how it is now possible to purchase a medical identity that mirrors individual ailments, size, age and gender, to seek "free" medical services that would not be suspicious to a clinician. It states this type of crime is estimated to cost the healthcare industry in the US between $35 billion and $80 billion each year [10].

It is largely assumed MMAs are not typically deployed in "hacker rich" mobile environments [11]. However, Arxan research shows that many sensitive medical and healthcare apps have been hacked with 22% of these being FDA approved apps [11]. MMA developers do not have extensive experience with the types of threats other consumer app industries (e.g., banking) are familiar with. Consequently, security and privacy has not been given serious consideration until recently, while the importance of security is getting recognized little is yet being done [12]. Development of MMAs is picking up momentum as many companies are lured into the domain by the explosion of the market and the

 ISBN: 978-1-61208-458-9

potential financial gains. However, issues arise such as: many of these developers are not coming from the highly regulated medical device domain and are not aware of the data protection and privacy requirements of PHI. Developers coming from the medical device domain are discovering the technical complications of entering the mobile domain. The European Commission's 'Green Paper on mHealth', findings are that this market is dominated by individuals or small companies, with 30% being individuals and 34.3% are small companies (defined as having 2-9 employees) [13]. This would advocate a lack of experience, knowledge and financial means to address the issues outlined above. The research aims to assist developers address privacy and security of data for MMAs, drawing from the standards and best practice perspectives.

This paper is organized as follows: Section II covers background on MMAs and data transmission. This section also discusses MMA security matters. Section III, outlines the privacy and security laws for health data. In Section IV, we introduce our research of a process model to assure the Confidentiality, Integrity and Availability (CIA) of data in transmission for developers of MMAs. Finally, we conclude the paper and present the future work in Section V.

II. BACKGOUND

A. MMAs and Data Transmission

In July 2011, the FDA issued draft guidance for MMAs and defined a "mobile medical app" as a software application run on a mobile platform (mobile phones, tablets, notebooks and other mobile devices) that is either used as an accessory to a regulated medical device or transforms a mobile platform into a regulated medical device and can be used in the diagnosis, treatment, or prevention of disease [14].

Mobile devices now provide many of the capabilities of traditional PCs with the additional benefit of a large selection of connectivity options [15]. Mobile devices typically connect to wireless sensor networks, which are being used in a wide range of medical and healthcare apps [16]. Wireless Body Area Networks (WBAN) emerged in order to address the growing field of sensor technologies. A WBAN is a purpose sensor network that operates independently to connect to various medical sensors and appliances, located inside and outside of a human body [17]. The information is transmitted via independent nodes that collect sensitive (life-critical) information [18]. A Task Group IEEE 802.15.61, was established for the standardization of WBAN. The current IEEE 802.15.6 standard purpose was to define new Physical (PHY) and Medium Access Control (MAC) layers for WBAN and defines three PHY layers; Narrowband (NB), Ultra wideband (UWB), and Human Body Communications (HBC) layers. The selection of each PHY depends on the application requirements.

Currently, technologies used for data transmission include Bluetooth/ Bluetooth Low Energy, ZigBee, UWB, Wireless Medical Telemetry Service (WMTS), communication networks such as WiFi (WLAN) and mobile data networks 3G & 4G. Data is transmitted to and from the MMA or to sensors on a personal health device or a medical device. Other transmission of data may occur between the MMA and for example: remote Health/Service Centers; Medical Professionals; or Health Record Networks. In some cases, the information sent to the MMA is processed on the app and retransmitted to the specified device or center. Through MMAs the collection of significant medical, physiological, lifestyle and daily activity data [13] is greatly amplified and transmitted via varied and numerous networks.

B. Mobile Medical Application Security

Security and privacy related to patient data are two essential components for MMAs. The fundamental concepts when considering data security are confidentiality, integrity and availability. Confidentiality is protection of the information from disclosure to unauthorized parties. Integrity refers to protecting information from being modified by unauthorized parties. Availability is ensuring that authorized parties are able to access the information when needed. When considering data security risks for MMAs it is necessary to specify what types of security threats they should be protected against. Deployment of MMAs involves security threats from multiple threat sources which include: attacks; the user; other mobile apps; network carriers; operating systems and mobile platforms. These security risks are further extended when consideration is given to the unauthorized access to the functionality of supporting devices and unauthorized access to the data stored on supporting devices [19]. The 2015 Ponemon report on mobile app security, emphasized that not enough is spent on mobile app security [20].

1) Attacks: Attacks are the techniques that attackers use to exploit the vulnerabilities in applications. There are numerous tools available for hacking into MMAs and wireless networks. Hackers target mobile apps to gain entry into servers or databases in the form of malware attacks. A recent list of these tools can be found in the Appendix of the Araxan Report [11]. This report examined 20 sensitive medical and healthcare apps and discovered 90% of Android apps and no iOS apps have been subject to hacking [11]. When data travels across a network, they are susceptible to being read, altered, or "hijacked". Potential for breaches of confidentiality of data occurs during collection and transmission of data. Data in transmission to and from the MMAs must be protected from hacking. Some of the most common issues (but not inclusive) are Easvesdropping, Malware, Node Compromise, Packet Injection, Secure Localization, Secure Management, Sniffing Attacks, Denial of Service (DoS), SQL injection attacks, Code Injection and Man-in-the Middle attacks. The consideration of WBANs for MMAs must satisfy rigorous security and privacy requirements [18]. Wireless channels are open to everyone. Monitoring and participation in the communication in a wireless channel can be done with a radio interface configured at the same frequency band [21]. This may cause severe damage to the patient since the cybercriminal can use the attained data [18] for many of the illegal purposes mentioned above. The ISO/IEEE 11073 standard deals only with mutual communication protocols and frameworks exchanged between and has never

 ISBN: 978-1-61208-458-9

considered security elements until recently, irrespective of all sorts of security breaches [22]. Security issues must be resolved while designing medical and healthcare apps for sensor networks to avoid data security issues [16].

2) Users: Many of the mobile devices will be personal and bypass the majority of inbound filters normally associated with corporate devices which leaves them vulnerable to malware. It is important that the user has good knowledge of the security safeguards, what measures to follow and what precautions to take [23]. A key challenge with MMA data is the lack of security software installed on mobile devices [24]. Many mobile device users do not avail of or are unacquainted with basic technical security measures, such as firewalls, antivirus and security software measures. Mobile device operating systems are very complex and therefore demand additional security controls for the prevention and detection of attacks against them [25]. The accessibility of social media and email make it easy to post or share information in violation of HIPAA regulations. An example being, a New York nurse was fired because she posted a photo to Instagram of a trauma room after treating a patient [26]. Mixed with the availability to mobile phone cameras and social media apps, the risk of employees divulging PHI and violating HIPAA requirements has increased [27]. One of the greatest threats to MMA data security lies with the fact that most are on mobile devices which are portable, making them much more likely to be lost or stolen [28]. Potentially any data on the device is accessible to the thief, including access to any data and hospital networks. Due to the regulatory protection of PHI, it is important that even when the app is on a stolen device the security of the data remains protected and is regularly backed-up [25]. Measures should be available to remotely lock the MMA, disable service, completely wipe out the data [25] and restrict access to supporting devices.

Not all users password-protect their devices. Even when passwords are used because of the lack of physical keyboards with mobile devices, users tends not use complex passwords to secure their information. The use of more than one type of authentication technique suggested by Alqahtani, would afford better data security for MMAs [25]. The difficulty is requesting lengthened authentication requirements from a busy medical professional. Inputting numerous passwords, or waiting for an authentication code in a pressurized situation is not desirable.

3) Other mobile apps: Unfortunately, many users download mobile apps often without considering the security implications. Unintentionally, a user can download malware in the form of another application, an update or by downloading from an unauthorised source. The difficulty in detecting the attack was due to the fact that there currently is no mobile device management application programming interface (API) to obtain the certificate information for each app [29]. An attacker can use Masque Attacks to bypass the normal app sandbox and get root privileges by attacking known iOS vulnerabilities [29]. Cloned apps are a concern, over 50% of cloned apps are malicious and therefore pose serious risks. A recently discovered iOS banking app malware, Masque Attacks, replace an authentic app with malware that has an identical UI. The Masque Attacks access the original app's local data, which wasn't removed when the original app was replaced and steal the sensitive data [29]. The mobile device management interface did not distinguish the malware from the original as it had used the same bundle identifier.

4) Operating systems & devlopment: Consideration with handling data on mobile devices includes unintended data leakage. It is essential that the MMA is not susceptible to analytic providers that will sell the data to marketing companies. The app stores are attempting to address this, e.g., Apple is banning app developers from selling HealthKit data or storing it on iCloud. Google insists that the user is in control of health data as apps cannot be accessed without the user providing permission. Developers could include analytics that report how often a section of the MMA was viewed, similar to the analytics credit card provider's use to flag unwanted access to data. It is equally important to consider the intentional or unintentional sharing of personal information. Leakage of personal data from the device to the MMA and the leakage of MMA data onto personal devices are key considerations. The bypass of outbound filters elevate the risk of non-compliance with data privacy laws and requirements, e.g., the use of personal Dropbox.

A basic requirement such as encryption is not used in many apps. Data is encrypted so that it is not disclosed whilst in transit. Data encryption service provides confidentiality against attacks. The requirement of encryption is stressed, not only for the data, but for the code in development to assure data security [16][25]. Data encryption of passwords and usernames if they are to be storage on the MMA is essential, many apps store this information in unencrypted text. This means that anyone with access to the mobile device the MMA resides on can see passwords and usernames by connecting the device to a PC. If the MMA is hacked, the information encrypted will be useless to the cybercriminals. Many apps send data over an HTTPS connection without checking for revoked certificates [30]. MMA developers should ensure that back-end APIs within mobile platforms are strengthened against attacks using state of the art encryption. As discussed above a MMA could expose healthcare systems that had not previously been accessible from outside their own networks. In MMA data security consideration developers should always use modern encryption algorithms that are accepted as strong by the security community.

Hackers are aware that just because a patch was released does not means it was applied, which, in turn make the app vulnerable for attacks [31]. Some recommend the installation of "Prevention and Detection" software for defending and protecting against malware as essential [25]. Consequently, software that tracks detection and anticipates attacks would require consideration in MMA development.

It is essential that developers research the mobile platforms they are developing for. Each mobile OS offers different security-related features, uses different APIs and handles data permissions its own way. Developers should adapt the code accordingly for each platform the MMA will

 ISBN: 978-1-61208-458-9

be run on. There are no standards that straddle development or security testing across the different platforms. Developers design security for each individual OS.

III. PRIVACY AND SECURITY LAWS FOR HEALTH DATA

In the rush to market the aspects of privacy and security are not properly considered [32]. Increasingly, MMA developers must deal with a range of international regulations if they want to perform business in more than one country. The absence of privacy laws in some countries, in addition to inconsistency or even conflicting laws, means PHI is often misused and treated superficially. Some MMA providers find they are in breach of regulation only when they are warned or fined, blindsided by regulatory issues, due to the complexity [33]. Privacy and security policy issues relating to data with MMAs are now of primary importance since the surge in the value of PHI on the black-market partly due to the lack of security controls within healthcare and the increase in the security of credit card data [34]. A global landscape analysis of current privacy legislation and regulation was undertaken by Thomas Reuters Foundation and mHealth Alliance on the privacy and security policies to protect health data [33]. The report states, that most jurisdictions agree, data security is essential and suggests the world of privacy law is divided into three major groups: Omnibus data protection regulation in the style of the European laws that regulate all personal information equally; U.S.-style sectorial privacy laws that address specific privacy issues arising in certain industries and business sectors, so that only certain types of personal information are regulated; The constitutional approach, whereby certain types of personal information are considered private and compelled from a basic human rights perspective but no specific privacy regulation is in place otherwise [33].

A. European Union

If you have MMAs within the EU, the EU Data Protection Directive (Directive 95/46/EC) [35] is the key piece of regulation that will affect how you manage and store data. This is the one law in the EU regarding security and privacy in health data. This Directive is implemented in laws of Member States and requires establishment of supervisory authorities to monitor its application. However, at the beginning of 2012, the EU approved the draft of the European Data Protection Regulation [36]. This means the law will apply generally over all states in the EU, so it will not require individual Member States implementation. With this progression in regulation all Member States will be at the same stage of security and data protection [32]. Directive 2002/58/EC of the European Parliament and of the Council of 12 July 2002 [37], known as the ePrivacy Directive, is concerned with the processing of personal data and the protection of privacy in the digital age. It is now law in all EU countries and covers all non-essential cookies, and tracking devices. This Directive principally concerns the processing of personal data relating to the delivery of communications services. It provides rules on how providers of electronic communication services, should manage their subscribers' data. It also guarantees rights for subscribers when they use these services. The key parts that MMA developers are concerned with in the directive are: processing security; confidentiality of communication; processing traffic and location data; cookies and controls.

B. United States

According to the Thomas Reuters Foundation and mHealth Alliance report, the US is one of the legislative leaders in this area [33]. The main law that applies to health data issues is HIPAA as stated previously. HIPAA was updated in the HIPAA Omnibus Rule required by The Health Information Technology for Economic and Clinical Health Act of 2010, (HITECH Act). The HITECH Act established new information security breach notification requirements that apply to businesses that handle personal health information and other health data [38]. The FDA released guidance "Content of Premarket Submissions for Management of Cybersecurity in Medical Devices" and this provides a list of recognized consensus standards dealing with Information Technology and medical device security [39]. The fact that MMAs may transmit information wirelessly places them in the domain of Federal Communications Commission (FCC) regulation to ensure consumer and public safety [40]. Recognizing the need for regulatory clarity, the FCC, FDA, Office of the National Coordinator (ONC) and the Department of Health and Human Services (HHS) came together in a grouping called the Food and Drug Administration Safety and Innovation Act (FDASIA) Working Group. The group released a report that contains a proposed strategy and recommendations on an appropriate, risk-based regulatory framework pertaining to health information technology including MMAs [41].

IV. PROPROSED CURRENT RESEARCH

A. Research Background

As the MMA domain grows and becomes a standard established mechanism for health delivery, data security and privacy of health data will be essential. MMAs are being developed persistently without proper security application, principally due to the lack of understanding of current standards, regulation requirements and best practice pertaining to data security in healthcare. There are currently no process models or testing suites for developers to assure data security in transmission for MMAs.

The proposed research is developed using the only Medical Device (MD) security standard, IEC/TR 80001-2-2:2012. This standard presents 19 high-level security-related capabilities in understanding the type of security controls to be considered and the risks that lead to the controls [42]. IEC/DTR 80001-2-8 (currently at a committee draft stage) is a catalogue of security controls developed relating to the security capabilities defined in IEC/TR 80001-2-2. The report presents mapping of security controls for developing security cases to establish confidence in each of the security capabilities [43]. Accordingly, the security controls support the maintenance of confidentiality and protection from malicious intrusion. The report provides guidance to healthcare organizations and MD manufacturers for the

 ISBN: 978-1-61208-458-9

selection of security controls to protect the CIA and accountability of data and systems during development, operation and disposal [43].

This research leverages on the established security controls in IEC/WD TR 80001-2-8 relating to the two transmission security capabilities from IEC/TR 80001-2-2. We will also apply additional security controls pertinent to MMAs, accomplished with comparative expert validation, by means of analysis of applicable standards and best practices. Further, we will research to adopt testing methods and applicable tests to form a testing suite, in collaboration with a data security expert to assure that the required security controls for data CIA in data transmission are in place.

B. Approach

The research will be completed in three parts. The research method will consist of Literature Review (LR) and Action Design Research (ADR) for each part.

1) Approach part one: The two transmission security capabilities selected from IEC/WD 80001-2-2 will provide the starting point. The catalogue of security controls in IEC/WD 80001-2-8 for the two capabilities will provide the basis for the security controls. The LR for this part will establish the additional standards and best practices pertaining to mobile data transmission and security of data. To develop the process to establish the security controls applicable to MMAs. Some of the standards and best practices currently being research include (but not inclusive) are: ISO/IEC 11073; NIST SP 800-53, OWASP mobile security; NIST FIPS 140-2; ISO 27799. The LR will additionally review other domains that have experience in data security in transmisssion, e.g., financial, to establish practices. The ADR fragment will develop and validate the process with comparitive expert review.

2) Approach part two: A LR will establish the current cyber attacks on mobile apps, MDs and MMAs and establish a database. The LR will additionally research testing methods associated with the attacks and applicable tests. This part will be completed in collaboration with identified data security experts and a testing organisationation. To assure that the required security controls for data CIA in data transmission are in place. The Testing Suite will be complied through ADR via the data experts, testing organisation and MMA developers.

3) Approach part three; The completion of the research will be through ADR with two identified MMA development companies. The development of the Process Model and the Testing suite will be validated through ADR with industrial partners. Completion of the research aim is the demonstration of confidence of data security during transmission MMAs. Therefore demonstrating confidence/trust in the data transmission and storage.

V. Conclusion and Future Work

This paper examined existing data security issues and practices in relation to MMAs. A summary of regulations relating to data privacy and security MMA providers are mandated by law to adhere to, were outlined. Compliance and improved understanding of data security regulations and best practices will assist developers to meet the security requirements for data in transmission. The security gaps in MMAs are exploited due to lack of knowledge, understanding or amalgamated regulation for data security with MMAs.

The mobile app industry claim innovation is stifled, due to the lack of clarity in regulations and security concerns. Developers will need to find the optimal balance between data security and privacy as MMAs expand and PHI enters into new aspects. The lack of consistent data security to assure privacy, to allow interoperability, and to maximize the full capabilities [44] of presents a significant barrier to the industry. The primary focus of our future research in this domain will be in the development and implementation of both the process model and testing suite. Validation of the research will be completed in collaboration with two MMA development companies. The MMAs being developed will have different transmission requirements and capabilities to assure diversity.

Acknowledgment

This research is supported by the Science Foundation Ireland through Lero - the Irish Software Engineering Research Centre (http://www.lero.ie) grant 10/CE/I1855 and grant 13/RC/20194.

References

[1] B. M. Silva, J. P. C. Rodrigues, F. Canelo, I. C. Lopes, and L. Zhou, “A data encryption solution for mobile health apps in cooperation environments.,” J. Med. Internet Res., vol. 15, no. 4, p. e66, Jan. 2013, doi:10.2196/jmir.2498.

[2] D. He, M. Naveed, C. A. Gunter, and K. Nahrstedt, “Security Concerns in Android mHealth Apps,” In AMIA Annual Symposium Proceedings. Nov. 2014, pp. 645–654.

[3] Y. Yang and R. . Sliverman, “‘Mobile health applications: the patchwork of legal and liability issues suggests strategies to improve oversight.,’” Health Aff., vol. 33, no. 2, pp. 222–7, 2014, doi: 10.1377/hlthaff.2013.0958.

[4] Price Waterhouse Cooper - Health Research Institute, “Top Health Industry Issues of 2015 - A new health economy takes shape,” Nov. 2014. pp. 1-18.

[5] “Data breach results in $4.8 million HIPAA settlements,” U.S. Department of Health and Human Services, 2014. [Online]. Available from: http://www.hhs.gov/about/news/2014/05/07/data-breach-results-48-million-hipaa-settlements.html 2016.01.10

[6] N. H. Ab Rahman, “Privacy disclosure risk: smartphone user guide,” Int. J. Mob. Netw. Des. Innov., vol. 5, no. 1, pp. 2–8, 2013, doi: 10.1504/IJMNDI.2013.057147.

[7] G. S. McNeal, “Health Insurer Anthem Struck By Massive Data Breach - Forbes,” Forbes, 2015. [Online]. Available from: http://www.forbes.com/sites/gregorymcneal/2015/02/04/massive-data-breach-at-health-insurer-anthem-reveals-social-security-numbers-and-more/ 2016.01.10

[8] U.S. FDA “Safety Communications - Cybersecurity for Medical Devices and Hospital Networks: FDA Safety Communication.” FDA Website, 2013 [Online]. Available from:http://www.fda.gov/MedicalDevices/Safety/AlertsandNotices/ucm356423.htm 2016.01.10

[9] FBI, “Internet of Things Poses Opportunities for Cyber Crime,” FBI Website, 2015. [Online]. Available from: https://www.ic3.gov/media/2015/150910.aspx 2016.01.10

 ISBN: 978-1-61208-458-9

[10] J. Williams, “Don’t Mug Me For My Password! - InformationWeek,” Information Week, 2014. [Online]. Available:http://www.informationweek.com/healthcare/security-and-privacy/dont-mug-me-for-my-password!/a/d-id/1318316 2016.01.10

[11] Araxan, “State of Mobile App Security,” Volume 3, Nov 2014.

[12] J. Kabachinski, “Mobile medical apps changing healthcare technology.,” Biomed. Instrum. Technol., vol. 45, no. 6, pp. 482–6, Nov/Dec. 2011, doi: 10.2345/0899-8205-45.6.482

[13] European Commission, “Green Paper on mobile Health (‘mHealth’),” Brussels, 2014.

[14] U.S. FDA, “Mobile Medical Applications Guidance for Industry and Food and Drug Administration Staff,” 2013.

[15] M. La Polla, F. Martinelli, and D. Sgandurra, “A Survey on Security for Mobile Devices,” Commun. Surv. Tutorials, IEEE vol. 15, no. 1, 2013, pp. 446–471.

[16] M. Al Ameen, J. Liu, and K. Kwak, “Security and privacy issues in wireless sensor networks for healthcare applications.,” J. Med. Syst., vol. 36, no. 1, pp. 93–101, Feb. 2012, doi: 10.1007/s10916-010-9449-4.

[17] J. Y. Khan and M. R. Yuce, “Wireless Body Area Network (WBAN) for Medical Applications,” in New Development in Biomedical Engeneering, D. Campolo, Ed. InTech, pp. 591–623, 2010.

[18] S. Saleem, S. Ullah, and K. S. Kwak, “A study of IEEE 802.15.4 security framework for wireless body area networks.,” Sensors (Basel)., vol. 11, no. 2, pp. 1383–95, Jan. 2011, doi: 10.3390/s110201383.

[19] J. L. Hall and D. McGraw, “For Telehealth to Succeed, Privacy and Security Risks Must be Identified and Addressed,” Health Aff., vol. 33, no. 2, pp. 216–221, 2014, doi: 10.1377/hithaff.2013.0997

[20] Ponemon Institute LLC, “The State of Mobile Application Insecurity,” IBM, 2015.

[21] V. Mainanwal, M. Gupta, and S. Kumar Upadhayay, “A Survey on Wireless Body Area Network : Security Technology and its Design Methodology issue,” in 2nd International Conference on Innovations in Information, Embedded and Communication systems (ICIIECS 2015), IEEE, March 2015, no. I, pp. 1–5, ISBN: 9781479968183

[22] S. S. Kim, Y. H. Lee, J. M. Kim, D. S. Seo, G. H. Kim, and Y. S. Shin, “Privacy Protection for Personal Health Device Communication and Healthcare Building Applications,” J. Appl. Math., vol. 2014, pp. 1–5, June 2014, doi: 10.1155/2014/462453

[23] M. Souppaya and K. Scarfone, NIST Special Publication 800-124 Guidelines for Managing the Security of Mobile Devices in the Enterprise. Gaithersburg, USA: National Institute of Standards and Technology, 2013, pp. 1–29.

[24] D. Nyambo, Z. O. Yonah, and C. Tarimo, “Review of Security Frameworks in the Converged Web and Mobile Applications,” Int. J. Comput. Inf. Technol., vol. 3, no. 4, pp. 724–730, Jul. 2014.

[25] A. S. Alqahtani, “Security of Mobile Phones and their Usage in Business,” Int. J. Adv. Comput. Sci. Appl., vol. 4, no. 11, pp. 17–32, 2013.

[26] C. Wiltz, “Mobile App Developers to Congress: HIPPA is Stifling Innovation | MDDI Medical Device and Diagnostic Industry News Products and Suppliers,” Mobile Health, 2014. [Online]. Available from: http://www.mddionline.com/article/mobile-app-developers-congress-hippa-stifling-innovation-140918 2016.01.10

[27] FierceHealthIT, “Mobile & HIPAA Securing personal health data in an increasingly portable workplace.” FierceHealthIT, 2014. [Online]. Available from: http://servicecenter.fiercemarkets.com/files/leadgen/mobile_and_hipaa_final.pdf 2016.01.10

[28] P. Ruggiero and J. Foote, “Cyber Threats to Mobile Phones,” United States Computer Emergency Readiness Team, 2011.

[29] H. Xue, T. Wei, and Y. Zhang, “Masque Attcak: All Your iOS Apps Belong to US,” FireEye, 2014. [Online]. Available from: https://www.fireeye.com/blog/threat-research/2014/11/masque-attack-all-your-ios-apps-belong-to-us.html 2016.01.10

[30] M. B. Barcena, C. Wueest, and H. Lau, “How safe is your quantified self” Symantech: Mountain View, CA, USA 2014.

[31] Y. S. Baker, R. Agrawal, and S. Bhattacharya, “Analyzing Security Threats as Reported by the United States Computer Emergency Readiness Team,” International Conference on Intelligence and Security Informatics (ISI 2013) IEEE, June 2013, pp. 10–12, ISBN:978-1-4673-6214-6

[32] B. Martinez-Perez, I. Torre-Diez de la, and M. Lopez-Coronado, “Privacy and Security in Mobile Health Apps: A Review and Recommendations,” J. Med. Syst., vol. 39, no. 1, p. 1-8, Jan. 2015, doi: 10.1007/s10916-014-0181-3

[33] Thomas Reuters Foundation and mHealth Alliance, “Patient Privacy in a Mobile World a Framework to Address Privacy LawIssues in Mobile Health,” Thomas Reuters Foundation, London, 2013.

[34] J. Williams, “Don’t Mug Me For My Password! - InformationWeek,” Information Week Healthcare, 2014. [Online]. Available from: http://www.informationweek.com/healthcare/security-and-privacy/dont-mug-me-for-my-password!/a/d-id/1318316 2016.01.10

[35] Directive, E.U. "95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data." Official Journal of EC 23.6, 1995.

[36] EU Commission. "Proposal for a Regulation of The European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). (Vol. 11) 2012.

[37] Directive, E.U. "2002/58/EC of the European Parliament and of the Council of 12 July 2002 concerning the processing of personal data and the protection of privacy in the electronic communications sector, Off." (Directive on privacy and electronic communications). JL 201, 31.7. 2002.

[38] L. J. Sotto, B. C. Treacy, and M. L. Mclellan, “Privacy and Data Security Risks in Cloud Computing,” Electron. Commer. Law Rep., vol. 186, pp. 1–6, Feb. 2010.

[39] U.S. Food and Drug Administration, “Content of Premarket Submissions for Management of Cybersecurity in Medical Devices - Guidance for Industry and Food and Drug Administration Staff,” 2014.

[40] A. A. Atienza and K. Patrick, “Mobile Health: The Killer App for Cyberinfrastructure and Consumer Health,” Am. J. Prev. Med., vol. 40, pp. 151–153, May 2011.

[41] Food and Drug Administration and Safety and Innovation Act (FDASIA), “FDASIA Health IT Report Proposed Strategy and Recommendations for a Risk-Based Framework,” 2014.

[42] IEC/TR 80002-2-2:2012, Application of risk management for IT-networks incorporating medical devices Part 2-2 : Guidance for the disclosure and communication of medical device security needs, risks and controls. 2012.

[43] A. Finnegan and F. McCaffery, “A security Argument for Medical Device Assurance Cases,” Softw. Reliab. Eng. Work. (ISSREW), IEEE Int. Symp., Nov. 2014, pp. 220–225.

[44] European Commission, “Medical Devices: Guidance Document MEDDEV 2.1/1.” 2012.

 ISBN: 978-1-61208-458-9

Reachability Games Revisited

Imran Khaliq
Media Design School
Auckland, New Zealand
email:imran.khaliq@mediadesignschool.com

Gulshad Imran
Dept. of Mathematics, University of Auckland
Auckland, New Zealand
email: ggul005@aucklanduni.ac.nz

***Abstract*— In this paper, we provide a refined analysis of the classical algorithm for solving reachability games. We provide a new algorithm that remembers information about fewer nodes than the classical algorithm does by computing the number of *efforts* made by the player to win the game.**

Keywords – reachability games; effort based strategies; memoryless determinacy

I. INTRODUCTION

In system Verification and Testing, the *reachability* question asking if a system can attain some specified state from a given state is well studied and well motivated. The reachability question was studied in the context of games by many [1] – [7]. Reachability games are played between two players Players 0 and Player 1, over a finite directed graph. The nodes of the graph are the states of the system it models and edges of the graph represent transitions of the system. An infinite sequence of states of the system can now be viewed as an infinite path through the graph. The question of reachability in verification can now be solved in terms of constructing winning strategies for the corresponding reachability game. In reachability games, Player 0 wins a play if the play visits some specified set, called a target set, at least once. A reachability game is solved in linear time on the size of the underlying graph [2].

The problem of solving a reachability game is mainly about constructing the *attractor* set for the winner. The concept of attractor set is also useful for the solution of infinite games with Safety [6], Buchi [8], McNaughton [9], and, Parity [10] winning conditions. The classical algorithm (see for instance [2]) for solving reachability games suggests a winning strategy that remembers ranks of all the nodes in the winning region of Player 0. Roughly, a rank of a node v is i if Player 0 can reach the target set (starting from v) within i moves made by the players. In this paper, we carefully analyse the attractor set of the target set. We call the attractor set (for Player 0) of the target set the winning region for Player 0. We observed that the winning region may contain Player 0 nodes such that all its outgoing edges lead to the winning region of Player 0. We call a set containing such nodes the *effortless region* of Player 0. The ranks of the nodes in effortless region need not to be remembered. Therefore, our winning strategy for Player 0 takes into account such nodes and hence improves upon memory efficiency. We call such strategies, *effort-based* strategies. We call a strategy that is dependent only on the current node of the play, *memoryless* strategy. Thus memoryless strategies do not depend on the *history*, where history is a finite prefix of a play. We will prove that effort-based strategies are memoryless strategies. Such strategies recall fewer ranks than the classical approach.

The summary of the paper is as follows. In Section II, we will provide basic definitions about games in general. In Section III, we will describe reachability games. In Section IV, we will provide our own procedure for solving reachability games. Finally, the conclusion is presented in Section V.

II. BASIC DEFINITIONS

Our games are played between two players. We call them Player 0 and Player 1. The underlying graph of a game is called arena. Our definition of arena is the following:

Definition 2.1 (Arena): An *arena* is a tuple $(V_0 \cup V_1, E)$, where V_0 and V_1 are pairwise disjoint sets of nodes and $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$ is the set of edges. We set $V = V_0 \cup V_1$. We also postulate that for every $u \in V$ there always exists $v \in V$ such that $(u, v) \in E$. We always assume that the set V of nodes is finite. The set of *successors* of $u \in V$ is defined by $uE = \{v \in V \mid (u, v) \in E\}$.

Thus, arenas are just finite bipartite graphs. The nodes of the set V_0 will be called Player 0's nodes and nodes of V_1 will be called Player 1's nodes.

Let $G = (V_0 \cup V_1, E)$ be an arena. A play between Player 0 and Player 1 is described as follows. The play begins at any node $v_0 \in V$. Say the node is in V_0. In this case, Player 0 selects an edge $e = (v_0, v_1)$, moves along the edge, and passes control to the opponent. Then, Player 1 selects an edge $e = (v_1, v_2)$, moves along the edge and passes control to the opponent. This goes on turn by turn. The definition of arena implies that at any given moment of the play, the players are able to make moves and continue the play. Formally, we define a play as follows.

Definition 2.2 (Play): A *play* in the arena $G = (V_0 \cup V_1, E)$ is an infinite sequence $v_0,\ v_1,\ v_2,\ v_3\ \ldots$ such that (v_0, v_1), (v_1, v_2), (v_2, v_3), $\ldots$ are all edges of the graph.

Unless the arena is trivial, there are infinitely many infinite plays. Clearly, every play is an element of V^ω, where V^ω is the set containing all sequences over V.

Let $G = (V_0 \cup V_1, E)$ be an arena. Let $\rho = v_0, v_1, v_2, \ldots,$ be an infinite play played between Player 0 and Player 1. We would like to define what it means that Player 0 wins the play ρ. The winner is determined through the following definition.

 ISBN: 978-1-61208-458-9

Definition 2.3: A *winning set* for Player 0 is a subset $W \subseteq V^\omega$. We say that Player 0 *wins* the play ρ if $\rho \in W$. Otherwise, Player 1 wins the play.

Now, we are ready to formally define a game.

Definition 2.4 (Game): A game between Player 0 and Player 1 is a tuple $(V_0 \cup V_1, E, W)$, where $G = (V_0 \cup V_1, E)$ is an arena and W is a winning set for Player 0. We typically denote games by Γ.

Note that, arenas are finite objects while winning sets W are not necessarily objects that are defined by finite means.

A. Strategies

In this section, given a game, we define the winner of the game and explain what it means to solve the game. The notation $\sigma \in \{0,\ 1\}$ will represent one of the two players Player 0 and Player 1, his opponent will be represented by $1 - \sigma$.

Let $\Gamma = (V_0 \cup V_1, E, W)$ be a game. We define histories of the game as follows.

Definition 2.5: A *history* is a finite prefix of a play. We define set of histories for Player σ as follows: $H(\sigma) = \{h \mid h$ is a history and the last letter of h is in $V_\sigma\}$.

Clearly, $H(\sigma) \cap H(1-\sigma) = \emptyset$ and every history is either in $H(\sigma)$ or in $H(1-\sigma)$. Informally, a strategy for Player σ, is a rule that tells the player which edge to select given a history of a play in $H(\sigma)$. Formally, we define a strategy for a player as follows.

Definition 2.6: A *strategy* for Player σ is a function $f_\sigma : H(\sigma) \to V$ such that for every $h = v_0, v_1, \ldots, v_n \in H(\sigma)$ we have $(v_n, f_\sigma(h)) \in E$.

Now, given a node $v \in V$ and strategy f_σ for Player σ, one can consider all the plays starting at v and consistent with the strategy f_σ. Here, we say that a play $\rho = v_0, v_1, v_2, \ldots$ is *consistent with* f_σ if $v_0 = v$ and for all histories $h = v_0, v_1, \ldots v_i \in H(\sigma)$ of this play we have $(v_i,\ f_\sigma(h)) \in E$ and $v_{i+1} = f_\sigma(h)$.

Definition 2.7: Let $\Gamma = (V_0 \cup V_1, E, W)$ be a game. Let $v \in V$ be a node.

- We say *Player σ wins from a node v* if Player σ has a winning strategy f_σ such that all the plays that begin from v and consistent with f_σ are winning for the player.
- We say *Player σ wins from a set* or has a *winning strategy from a set $A \subseteq V$* if Player σ has a winning strategy from each node in A.
- We say that v is a *winning node for* Player σ if the player wins the game from the node v.

From this definition, it follows that, if v is a winning node for a player, then v can not be a winning node for the opponent. One of the fundamental concepts in game theory is the following definition.

Definition 2.8: A game Γ is *determined* if every node of the game is winning for either Player 0 or Player 1.

There are examples of games that are not determined, see [11]. Determinacy is one of the important topics in descriptive set theory. One of the important theorems is the following theorem of Martin [12].

Theorem 2.9 (Martin's determinacy theorem): Every Borel game, that is the game at which W is a Borel set, is determined.

Reachability games are Borel and hence determined. The definition below is meant when we say a game is solved.

Definition 2.10: Let Γ be a game. We say Γ is solved if there exists an algorithm that given the Γ, outputs the sets W_0 and W_1, where W_0 is the set of all nodes in Γ from which Player 0 wins the game and W_1 is the set of all nodes in Γ from which Player 1 wins the game. The set W_σ, $\sigma \in \{0, 1\}$ is called winning region for Player σ.

III. Reachability Games

In this section, we discuss reachability games in detail. The algorithm and definitions discussed here are borrowed from [2]. In reachability games, Player 0 wins a play if the play visits a specified set of nodes at least once. Formally, we define reachability games as follows.

*Definition 3.1 (*Reachability Games): A *reachability game* Γ consists of:

1) The arena $G = (V_0 \cup V_1, E)$.
2) The target set T of nodes $T \subseteq V_0 \cup V_1$.

We say that *Player 0 wins a play* $v_0,\ v_1,\ v_2,\ v_3\ \ldots$ if there exists an i such that $v_i \in T$. Otherwise, Player 1 wins the play.

From the definition, it is clear that Player 0 wins a play ρ from a node u if

- ρ begins from the node u;
- there is a finite prefix η of ρ such that the last node in η belongs to the target set.

Definition 3.2: A *memoryless strategy* for Player σ is a function $f_\sigma : V_\sigma \to V$ such that $(u, f_\sigma(u)) \in E$. A game enjoys *memoryless determinacy* if for every node one of the players wins the game with memoryless strategy.

It turns out that winners in reachability games have memoryless winning strategies. We prove this in the next theorem. Before we proceed, we define some notations. Let Γ be a reachability game. Assume $X \subseteq V$. Define,

$$\mathsf{reach}_\sigma(X) = \{u \in V_\sigma \mid \exists\ v \in uE \cap X\} \cup \{u \in V_{1-\sigma} \mid uE \subseteq X\}.$$

When the player is clear, then sometimes we denote the above set by $\mathsf{reach}(X)$.

Theorem 3.3 (Memoryless Determinacy [2] pp. 34): Reachability games enjoy memoryless determinacy.

Proof: The winning region for Player 0 is defined inductively. We set, $X_0 = T$, and for $i \in \omega$, $X_{i+1} = \mathsf{reach}_0(X_i) \cup X_i$. Since the set of nodes V is finite there is an s such that $X_s = X_{s+1}$, where s is the smallest such number.

Claim 3.4: The set X_s is the winning region for Player 0, that is $W_0 = X_s$.

For the proof, we use the concept of rank. We say a node u has *rank* r, $r \geq 0$, if $u \in X_r \setminus X_{r-1}$. A node u has infinite rank if $u \notin X_r$ for all r.
To define a memoryless strategy f_0 for Player 0 we linearly order $<$ the set V_1 that is, $v_1 < v_2 < v_3 < \cdots < v_l$ where $l = |V_1|$. We define f_0 as follows:

> Let $v \in X_s \cap V_0$. Let r be the rank of v. Then $f_0(v)$ is minimal with respect to $<$ such that $(v, f_0(v)) \in E$ and rank of $f_0(v)$ is $r-1$.
> For $v \notin X_s$, we set $f_0(v)$ be the minimal with respect to the order $<$ such that $(v, f_0(v)) \in E$.

We show that f_0 is winning strategy from the set X_s. Let v be a node in X_s. From the above definition of $\mathsf{reach}(Y)$ it implies that v has some finite rank r say. If v is Player 0's node then by the strategy f_0 Player 0 chooses $f_0(v)$ of rank $r-1$. If v is Player 1's node then Player 0 waits for Player 1's move. Since $vE \subseteq X_{r-1}$, any choice of Player 1 selects a node in vE of rank strictly less than r. Each player's move select a node of lesser rank every time. Since r is finite, every play that begins from v ultimately ends at a target node. Hence $X_s \subseteq W_0$.

Now, we show that Player 0 cannot win from a node in $V \setminus X_s$. Let $M = V \setminus X_s$. To define a memoryless strategy f_1 for Player 1 we linearly order the set V_0 that is $v'_1 < v'_2 < v'_3 < \cdots < v'_m$ where $m = |V_0|$.

> If $v \in M \cap V_1$ then $f_1(v)$ is minimal with respect to the order such that $f_1(v) \in M$ and $(v, f_1(v)) \in E$.
> If $v \in X_s$ then $f_1(v)$ is the minimal with respect to the order such that $(v, f_1(v)) \in E$.

Let v belong to M. If v is Player 0's node then Player 1 does nothing but just waits for the Player 0's move. Any choice of Player 0 selects a node in M. This is because $vE \subseteq M$ as otherwise v would be in W_0.

If v is Player 1's node then there exists a node $v' \in M$ such that $(v, v') \in E$ otherwise v would belong to W_0. By strategy f_1 Player 1 chooses minimal $f_1(v)$ with respect to the order $<$ such that $f_1(v) \in M$ and $(v, f_1(v)) \in E$. Player 0 cannot win any play which begins from a node belongs to M if Player 1 follows the strategy f_1. This is because $M \cap T = \emptyset$. Hence, $W_0 = X_s$ and $W_1 = M$. ■

Corollary 3.5: There exists an algorithm that solves reachability games in $O(|V| + |E|)$ time, where V is the set of nodes and E is the set of edges in the arena.

Proof: We construct the winning region for Player 0 inductively. Initially, we set $X_0 = T$. Suppose X_i is constructed. To construct X_{i+1}, first we copy elements of X_i to X_{i+1}. Second, we add a node u in X_{i+1} if:

- $u \in V_0$ and if there exists a node $v \in X_i$ such that $(u, v) \in E$, then we add to X_{i+1}.
- $u \in V_1$ and if $uE \subseteq X_i$, then we add to X_{i+1}.

To implement the procedure for constructing the winning region in $O(|V| + |E|)$ time, we assign a counter $c(u)$ to a node $u \notin T$. Initially, we set $c(u) = 1$ if $u \in V_0$ and $c(u) = |uE|$ if $u \in V_1$.

Whenever, we add a node v to X_{i+1}, where $v \notin X_i$, we subtract 1 from the counter of each node u such that $(u, v) \in E$; From this point on the edge (u, v) will never be used again. When a counter becomes zero then the node is also added to X_{i+1}. This shows that the running time of the algorithm is in $O(|V| + |E|)$. ■

Corollary 3.6: Let Γ be a reachability game. The function $\phi_\sigma : 2^V \to 2^V$ defined by $\phi_\sigma(A) = A \cup \mathsf{reach}_\sigma(A)$, is monotone function with respect to set inclusion, where $A \subseteq V$ and 2^V is the set containing all subsets of V. That is, $\phi_\sigma(A) \subseteq \phi_\sigma(B)$ whenever $A \subseteq B$.

Definition 3.7: We denote the winning region of Player 0 in a reachability game by $\mathsf{Attr}_0(T)$ and call it the *0-attractor* of the set T. A memoryless winning strategy f_0 as described in the proof of Theorem 3.3 is called *T-attractor strategy* for Player 0. When the target set T is clear, then we simply say attractor strategy for Player 0.

Note that, in reachability games, we can change the roles of the players. In this case, Player 1 tries to reach a given set T while the opponent tries to avoid it. As shown above, one can build the 1-attractor of the set T. Hence, we can talk about σ-attractor sets for the players when a target set is specified.

Definition 3.8: A *σ-trap* (or trap for σ) is a subset $X \subseteq V$ such that $vE \subseteq X$ for every $v \in X \cap V_\sigma$ and $vE \cap X \neq \emptyset$ for every $v \in X \cap V_{1-\sigma}$. A memoryless strategy which assigns for $v \in X \cap V_{1-\sigma}$ a node $f(v) \in vE \cap X$ is called a *trapping strategy* for Player $1 - \sigma$.

Corollary 3.9: The complement of σ-attractor of a target set T is σ-trap.

IV. Reachability games and effort moves

In this section, We give a refined analysis of the set $\mathsf{Attr}_0(T)$ for a given reachability game Γ. The idea here is to compute the number of efforts made by Player 0 to win the game from $\mathsf{Attr}_0(T)$. Let $X \subseteq V$ be a set. Define $\Im(X) = \{v \in V \mid vE \subseteq X\}$. We define the sequence $\Im_0, \Im_1, \Im_2, \Im_3, \ldots$ as follows:

$$\Im_0 = X, \quad \Im_{i+1} = \Im(\Im_i) \cup \Im_i.$$

Since the arena is finite, there exists a minimal k such that $\Im_{k+1} = \Im_k$. We call this $\Im_k$, the *effortless region for* X and denote it by $\mathsf{eff}(X)$.

Lemma 4.1: Player 0 has a winning strategy from $\mathsf{eff}(T)$ to visit T.

Proof: Let $u \in \mathsf{eff}(T)$. This implies that $u \in \Im_i$ for some i. Since $uE_i \subseteq \Im_{i-1}$, any play starting at u will eventually visit $\Im_0 = T$. Thus, a winning strategy for Player 0 is simply to choose any node v such that $(u, v) \in E$. ■

In order to construct the winning region for Player 0, we define the sequence $\mathsf{eff}_0, \mathsf{eff}_1, \mathsf{eff}_2, \mathsf{eff}_3, \ldots$ as follows:

$$\mathsf{eff}_0 = \mathsf{eff}(T)$$

$$\mathsf{eff}_{i+1} = \mathsf{eff}(\mathsf{eff}_i \cup \mathsf{reach}(\mathsf{eff}_i)).$$

Let us recall $\mathsf{reach}(Y) = \{u \in V_0 \mid \exists v \in uE \cap Y\} \cup \{u \in V_1 \mid uE \subseteq Y\}$, where $Y \subseteq V$. Here, the second part is empty. Since, the arena is finite, it implies that there exists a minimal t such that $\mathsf{eff}_{t+1} = \mathsf{eff}_t$.

Definition 4.2: The Player 0's move from a node u to a node v in a reachability game is called an *effort move* if $u \in \mathsf{reach}_0(\mathsf{eff}_i) \setminus \mathsf{eff}_i$ and $v \in \mathsf{eff}_i$ for some i.

Lemma 4.3: Player 0 has a strategy from $\mathsf{eff}_{i+1} \setminus \mathsf{eff}_i$ to visit T after $i+1$ effort moves have been made.

Proof: We prove the lemma by induction on i. For $i = 0$, let $x \in \mathsf{eff}_1 \setminus \mathsf{eff}_0$. This implies that any play from x will eventually visit $\mathsf{reach}_0(\mathsf{eff}(T))$. For every $u \in \mathsf{reach}_0(\mathsf{eff}(T))$, there exists a $v \in \mathsf{eff}(T)$ such that $(u, v) \in E$ otherwise $u \notin \mathsf{reach}_0(\mathsf{eff}(T))$. We set $f(u) = v$. Player 0 now moves to this v.

Let the lemma be true for $i = k$. Let $x \in \mathsf{eff}_{k+1} \setminus \mathsf{eff}_k$. Any play from x will eventually visit $\mathsf{reach}(\mathsf{eff}_k)$. For every $u \in \mathsf{reach}_0(\mathsf{eff}_k)$ there exists a $v \in \mathsf{eff}_k$ such that $(u, v) \in E$, otherwise $u \notin \mathsf{reach}_0(\mathsf{eff}_k)$. We set $f(u) = v$. If Player 0 follows this strategy f, then Player 0 can visit eff_k after one effort has been made if a game begins from the node x. By induction hypothesis Player 0 can visit T after $k+1$ effort moves. ■

Theorem 4.4: Let Γ be a reachability game. Consider the sequence defined as follows:

$$\mathsf{eff}_0 = \mathsf{eff}(T)$$

$$\mathsf{eff}_{i+1} = \mathsf{eff}(\mathsf{eff}_i \cup \mathsf{reach}_0(\mathsf{eff}_i)).$$

If t is the minimal number such that $\mathsf{eff}_{t+1} = \mathsf{eff}_t$ then eff_t is the winning region for Player 0 and $V \setminus \mathsf{eff}_t$ is the winning region for Player 1.

Proof: Let $u \in \mathsf{eff}_t$. By the above two lemmas Player 0 has a strategy to visit T after t effort moves. Hence $\mathsf{eff}_t \subseteq W_0$. This strategy can be written explicitly as follows. For any given $u \in V_0$ set:

> If $u \in \mathsf{reach}_0(\mathsf{eff}_i) \setminus \mathsf{eff}_i$ for some i then select v such that $(u, v) \in E$ and $v \in \mathsf{eff}_i$. Otherwise, select any w such that $(u, w) \in E$.

To prove $W_0 = \mathsf{eff}_t$, we show that Player 1 has a winning strategy from $V \setminus \mathsf{eff}_t$. We set $W' = V \setminus \mathsf{eff}_t$. Let a play begins from a node u in W'. If u is Player 0's node then Player 1 does nothing but just waits for the Player 0's move at u. Any choice of Player 0 selects a node in W'. This is because $uE \subseteq W'$ as otherwise u would be in $\mathsf{reach}_0(\mathsf{eff}_t)$. If $u \in V_1 \cap W'$, then there exists a node $v \in W'$ such that $(u, v) \in E$ otherwise u would belong to eff_t. We define $g(u) = v$. Any play which begins from a node in W' and consistent with this strategy g always stays inside W'. Hence g is a winning strategy for Player 1 from W' because $W' \cap T = \emptyset$. Thus, $W_0 = \mathsf{eff}_t$ and $V \setminus \mathsf{eff}_t = W_1$. ■

Note that the winning strategy f, for Player 0, extracted from the classical algorithm that solves a reachability game, remembers ranks of all the nodes in $\mathsf{Attr}_0(T)$. That is, for all $u \in \mathsf{Attr}_0(T)$, $f(u) = v$, where $v \in uE$ and rank of v is strictly less than u. For all $z \in V_0 \setminus \mathsf{Attr}_0(T)$, $f(z)$ is such that $(z, f(z)) \in E$. Our new procedure for solving reachability games suggests a winning strategy g that remembers only nodes in

$$T \cup \mathsf{reach}_0(\mathsf{eff}_0) \cup \mathsf{reach}_0(\mathsf{eff}_1) \cup \mathsf{reach}_0(\mathsf{eff}_2) \cup \ldots.$$

We define g as follows. For all $u \in X$, $g(u) = f(u)$, where $X = T \cup \mathsf{reach}_0(\mathsf{eff}_0) \cup \mathsf{reach}_0(\mathsf{eff}_1) \cup \mathsf{reach}_0(\mathsf{eff}_2) \cup \ldots$. For all $z \in V_0 \setminus X$, $f(z)$ is such that $(z, f(z)) \in E$. We call this strategy, *effort-based* strategy . Thus, we obtain the following theorem.

Theorem 4.5: Given a reachability game, there exists a linear time algorithm that extracts an effort-based memoryless winning strategy for the winner.

V. Conclusion

To win a reachability game, the classical algorithm remembers ranks of all nodes of the arena. The winning region for a player may contain nodes such that all their outgoing edges lead to the winning region of the player and hence no effort is involved by the player at such nodes. The region that contains such nodes we called it effortless region. Our algorithm takes effortless region into account and hence improved memory efficiency. Moreover, the algorithm is memoryless and it takes linear time on the number of nodes and edges of the arena.

References

[1] W. Thomas, "Infinite games and verification," Proc. Computer Aided Verification, LNCS 2404, Springer 2002, pp. 58–64.

[2] E. Grädel, W. Thomas, and T. Wilke, "Automata, logics, and infinite games: A guide to current research", LNCS 2500, Springer, Heidelberg, 2002.

[3] L. Roditty, and U. Zwick, "A fully dynamic reachability algorithm for directed graphs with an almost linear update time", Proc. 36th ACM Symposium on Theory of Computing, 2004, pp. 184-191.

[4] B. Khoussainov, J. Liu, and I. Khaliq, "A dynamic algorithm for reachability games played on trees", Proc. 24th International Symposium of Mathematical Foundations of Computer Science, LNCS, vol. 5734, 2009, pp. 518-529.

[5] P. Hunter, "Reachability in succinct one-counter games", Proc. 9th International Workshop on Reachability Problems, LNCS 9328, 2015, pp. 37- 49.

[6] K, Chatterjee, L. Alfaro, and T. A. Henzinger, "Strategy improvement for concurrent reachability and turn-based stochastic safety games", Jouranl of Computer and System Sciences, vol. 79, issue 5, 2013, pp. 640 - 657.

[7] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz, "How much memory is needed to win infinite games", Proc. 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw Poland, 1997 pp. 99 - 110.

[8] K. Chatterjee, T. Henzinger, and N. Piterman, "Algorithms for buchi games", Proc. 3rd Workshop of Games in Design and Verification, 2006.

[9] R. McNaughton, "Infinite games played on finite graphs", Annals of Pure and Applied Logic, vol. 65, pp. 149184, 1993.

[10] M. Huth, J. H. Kuo, and N. Piterman, "Fatal attractors in Parity games", Proc. FoSSaCS, pp. 34 - 49, 2013.

[11] D. Gale, and F. Stewart, "Infinite games with perfect information. Annals of Mathematical Studies (Contributions to the Theory of Games II)". vol. 28, 1953, pp. 245-266.

[12] D. Martin, "Borel determinacy. Annals of Mathematics", vol. 102, 1975, pp. 363-375.

 ISBN: 978-1-61208-458-9

Dynamic Symbolic Execution with Interpolation Based Path Merging

Andreas Ibing

Chair for IT Security
TU München
Boltzmannstrasse 3, 85748 Garching, Germany
Email: andreas.ibing@tum.de

***Abstract*—This paper presents a dynamic symbolic execution engine for automated bug detection in C code. It uses path merging based on interpolation with unsatisfiable cores to mitigate the exponential path explosion problem. Code coverage can be scaled by varying the interpolation. An algorithm for error and branch coverage is described. The implementation extends Eclipse CDT. It is evaluated on buffer overflow test cases from the Juliet test suite in terms of speed-up through merging, reduction of the number of analyzed program paths and proportion of merged paths.**

Keywords–Symbolic execution, interpolation, branch coverage, error coverage.

I. INTRODUCTION

Symbolic execution [1] is a program analysis technique, that can be used for automated bug detection. In order to find bugs with arbitrary program input, the program input is treated as symbolic variables. Operations on these variables then yield logic equations. Satisfiability of program paths and satisfiability of bug conditions are decided by an automated theorem prover (constraint solver). The current state of automated theorem provers are Satisfiability Modulo Theories (SMT) provers [2].

Symbolic execution can be applied both as static analysis (without executing the progran under test) and as dynamic analysis (using binary instrumentation) [3]. Dynamic symbolic execution follows a program path with a complete concrete program state, and additionally a partial symbolic program state. The partial symbolic program state comprises the constraints on symbolic variables which have been collected on the path (path constraint). The concrete program state satisfies the constraints on the symbolic variables. Dynamic symbolic execution is also known as concolic execution (concrete/symbolic [4]). Dynamic symbolic execution has several advantages compared to the static only approach. Complicated program constructs can be concretized, i.e., executed only concretely by dropping the relevant symbolic variables [3]. Concretization is sound with respect to bug detection, i.e., while it does lead to false negative detections, it does not lead to false positive bug detections. Concretization also provides more flexibility in handling library function calls. Function call parameters can be concretized and the function executed concretely. Dynamic symbolic execution with configurable concretization is also called selective symbolic execution [5]. Another argument for dynamic symbolic execution is that execution of concrete code is much faster than symbolic interpretation.

The number of satisfiable program paths in general grows exponentially with the number of branch decisions, for which more than one branch is satisfiable. This bad scaling behaviour is known as path explosion problem. In order to alleviate the path explosion problem, it is shown in [6] that a live variable analysis can be applied so that program paths, that only differ in dead variables, can be merged. A more comprehensive sound path merging approach is described in [7]. It is based on logic interpolation (Craig interpolation [8]), i.e., on automated generalization of constraint formulas. The interpolation uses unsatisfiable cores (unsat-cores) and approximates weakest precondition computation. Given an unsatisfiable conjunction of formulas, an unsat-core is a subset of the formulas whose conjunction is still unsatisfiable. This approach leads to better scaling behaviour by finding more possibilities to merge program paths.

The accuracy of bug detection tools is typically evaluated as percentage of false positive and false negative bug detections in a sufficiently large bug test suite. Currently the most comprehensive test suite for C/C++ is the Juliet suite [9]. In order to systematically test a tool's accuracy, it combines 'baseline' bugs with different data and control flow variants. The maximum context depth spanned by a flow variant is five functions in five different source files. Each test case is a program that contains 'good' (bug-free) as well as 'bad' functions (which contain a bug), so that both false positives and false negatives can be measured.

This paper presents a dynamic symbolic execution engine which uses unsat-core based interpolation of unsatisfiable program paths and unsatisfiable bug conditions in order to achieve scalability through merging as many program paths as early as possible. The engine is applied to the problem of automated bug detection (testing). This includes that with each bug detection, the constraints for merging program paths are automatically adapted.

The remainder of this paper is organized as follows: Section II describes the motivation and details of the algorithm. Section III describes scaling of code coverage by varying interpolation, and puts the described algorithm for error and branch coverage into context. The implementation as plug-in extension to the Eclipse C/C++ development tools (CDT) is depicted in section IV. Section V evaluates the tool in terms of speed-up through path merging and of the number of completely and partly analyzed program paths on buffer overflow test cases from the Juliet test suite. Related work is described in section VI. Evaluation results are discussed in section VII.

 ISBN: 978-1-61208-458-9

II. ALGORITHM

This section describes the motivation (in subsection II-A) and details of the algorithm and gives an analysis example in subsection II-E.

A. Motivation

Motivation for the algorithm are the following points:

- It is sufficient to detect each bug on one program path only. It is not necessary to detect each bug on all paths where this bug might be triggered. A program path can therefore be pruned if it is impossible to detect any new bugs on any extension of the path. This information can be gained from backtracking program paths, that were analyzed till program end. This implies a depth-first traversal of the program execution tree (the tree of satisfiable program paths).
- Since interpretation is much slower than execution of code, as much code as possible should not be interpreted. Execution should be interrupted only at locations, that need to be interpreted symbolically. Constraint-based analysis is needed only for the detection of input dependent bugs. In this paper, a debugger is used for adaptive binary instrumentation, i.e., breakpoints are path-dependent for efficiency. Variables can become symbolic (through assignment of a symbolic value) and concrete (through assignment of a concrete value). Breakpoints are only set for locations where symbolic variables are used.

B. Algorithm overview

The algorithm has two analysis steps. The first step is a path-insensitive static analysis as preparation, in order to determine locations that must be symbolically interpreted (initial debugger breakpoints). Program input is treated as symbolic. This includes the return values of certain standard library functions (these functions can be configured). From these starting points, the static analysis uses inference over the data flow to find out which locations can be reached with these variables. This is often called a taint analysis. Details are described in subsection II-C.

The second step is dynamic symbolic execution, which is a path-sensitive and context-sensitive analysis and therefore needs a constraint solver as logic backend. For an efficient dynamic analysis where variables path-sensitively can become symbolic or concrete, the program locations that need to be symbolically interpreted are also path-sensitive. Conceptually, read/write breakpoints are needed for all symbolic variables. This is implemented by setting breakpoints on all locations where a symbolic variable is used. Breakpoints are adaptively set and removed during analysis. A path can be merged (pruned) during symbolic analysis when the path constraint implies a merge formula for the same location. Merge formulas are generated by backtracking unsat-cores for unsatisfiable error conditions and unsatisfiable program paths. Locations where path merging possibilities are checked (merge locations) are branch nodes in control flow graphs (CFG). Details are described in subsection II-D.

The algorithm overview is also listed as pseude-code in Algorithm 1. The static pre-analysis corresponds to line 1. The depth-first dynamic symbolic execution corresponds to lines 4-37.

```
1  Set{Location} symlocs = findInitialBreakLocations();
2  debugger.setBreaks(symlocs);
3  direction = forward;
4  while (! (direction == exhausted) ) do
5      if (direction == forward) then
6          Location loc = debugger.continue();
7          if (isProgramEnd(loc)) then
8              direction = backtrack;
9              continue;
10         if (mergeLocs.contains(loc)) then
11             if (cansubsume(loc)) then
12                 direction = backtrack;
13                 continue;
14         cfgnode = getNode(loc);
15         interprete(cfgnode);
16     else if (direction == backtrack) then
17         foundNewInputVec = false;
18         while (!foundNewInputVec) do
19             backtrackErrorGuards(cfgnode);
20             if (cfgnode instanceof BranchNode) then
21                 setNewMergeLocation(cfgnode);
22             cfgnode = backtrackLastNode(path);
23             if (isProgramStart(cfgnode)) then
24                 direction = exhausted;
25                 break;
26             if (cfgnode instanceof DecisionNode) then
27                 if (hasOpenBranch(cfgnode)) then
28                     boolean isSat = checkSat(path +
                        openBranch);
29                     if (isSat) then
30                         InputVector newInput =
                            getModel(path + openBranch);
31                         foundNewInputVec = true;
32                     else
33                         uc = getUnsatCore(path +
                            openBranch);
34                         setGuard(openBranch, uc);
35         if (foundNewInputVec) then
36             direction = forward;
37             debugger.restart();
```

Algorithm 1: Dynamic symbolic execution with interpolation based path merging

C. Preparation: path-insensitive extended taint analysis

The static pre-analysis determines for all program locations:

- which variable definitions may reach the location (reaching definitions [10])
- whether a *symbolic* variable might be used (read) at the location; in the following this location property is called 'maybe symbolic'
- whether the location is a potential bug location; This property depends on the bug types that are to be found. For the example of buffer overflow detection, a location is considered a potential bug location when it contains an array subscript expression or a pointer dereference.
- whether the location is a control flow decision node

From this information, it is then determined, for which locations breakpoints must be set for *all* program paths. That is:

- input dependent bug locations: a potential bug location where a symbolic variable might be used. These locations must be symbolically interpreted in order to detect the bugs with a solver satisfiability check or to compute an unsat-core.
- input dependent control flow decisions: input dependent branches must be symbolically interpreted for correct merge formula generation during backtracking and to avoid incorrect path merging. This is described in more detail is subsection II-D.

The analysis is implemented as a monotoneous propagation of changes and uses the worklist algorithm [10]. Source files are parsed into abstract syntax trees (AST), and control flow graphs are computed for all function definitions in the ASTs. When the properties of a control flow node change, the change is propagated to its children (which are then added to the worklist). Since the propagation is monotoneous, the reaching of a fixed-point (empty worklist) is guaranteed. The analysis is not path-sensitive and does not need a constraint solver. It therefore has a better scaling behaviour than symbolic execution.

D. Selective symbolic execution with unsat-core based interpolation

The symbolic execution is essentially a depth-first traversal of the execution tree. As such, it has a forward and a backtracking mode. The backtracking mode generates program input for the next path. The current path is backtracked to the last input-dependent control flow decision. If possible (satisfiable), the last decision is switched to obtain a new path.

1) Forward symbolic execution: The forward symbolic execution mode corresponds to lines 5-15 in Algorithm 1. The debugger is run until it stops at a breakpoint. For this program location, the corresponding CFG node is resolved (line 14). This CFG node is then interpreted and translated into an SMT logic equation. Values of concrete variables are queried from the debugger when needed. More details regarding the translation are provided in the implementation section (Section IV). Functions from the standard library are wrapped, so that input can be traced and forced as desired using debugger commands.

a) Unsat-core interpolation for unsatisfiable bug conditions: Another path can be merged if no new bug detection is possible along any of its extensions, i.e., when potential bug locations remain unsatisfiable for the new path. This is the case when the new path's path constraint implies the unsat-cores of the potential bug locations.

b) Updating for bug detections: When a bug is detected, any new detections of same bug (same type and location) become irrelevant. Therefore, any unsat-cores that were computed for this potential bug location before, can be deleted and the constraints removed from merge formulas. Unsat-cores are computed using the idea of serial constraint deletion from [7]. A path constraint is a conjunction of a set of formulas. For each of these formulas it is checked in turn with the solver, whether the conjunction remains unsatisfiable if the formula is removed. The function is only kept if the conjunction would become satisfiable otherwise. In the following, a computed unsat-core is also called an error guard.

```
1  void CWE121_fgets_12_bad() {
2    int data = -1;
3    if(global_returns_t_or_f()) {
4      char input_buf[ARR_SIZE] = ""; path ⇒ (data=-1) ?
5      if (fgets(input_buf, ARR_SIZE, stdin) !=
           NULL) {
6        data = atoi(input_buf);      path ⇒ ⊤ ?
7      } else {
8        printLine("fgets() failed."); path ⇒ (data=-1) ?
9      }
10   } else {                        path ⇒ ⊤ ?
11     data = 7;
12   }
13   if(global_returns_t_or_f()) {
14     int i;
15     int buffer[10] = { 0 };       path ⇒ ⊤ ?
16     if (data >= 0) {
17       buffer[data] = 1;
18       for(i = 0; i < 10; i++) {
19         printIntLine(buffer[i]);
20       }
21     } else {
22       printLine("ERROR: Array index
             negative.");
23     }
24   } else {
25     int i;
26     int buffer[10] = { 0 };       path ⇒ ⊤ ?
27     if (data >= 0 && data < (10)) {
28       buffer[data] = 1;
29       for(i = 0; i < 10; i++) {
30         printIntLine(buffer[i]);
31       }
32     } else {
33       printLine("ERROR: index
             out-of-bounds");
34     }
35   }
36 }

37 int global_returns_t_or_f() {
38         return (rand() % 2)
39 }
```

Figure 1. Example function, from the Juliet test suite [9]

c) Path merging: Breakpoints are set during backtracking for branch locations, for which at least one merge formula has been computed. When the current path constraint implies the merge formula, the path is pruned. The implication check uses the solver and corresponds to line 11 in Algorithm 1. The implication is valid if its negation is not satisfiable.

2) Backtracking: The backtracking mode corresponds to lines 16-37 in Algorithm 1. It generates program input for the next path and backtracks error guards to generate merge formulas. Backtracking is only concerned with the partial symbolic program state along the current path, i.e., only with locations which were symbolically interpreted.

a) Input generation: The symbolic program state is backtracked to the last decision node. For child branches that were not yet covered in the context of the current (backtracked) path, it is checked with the solver whether or not this path extension is satisfiable. If it is satisfiable, the solver's model generation functionality is used to generate corresponding program input values for the next path. If not, an unsat-core is computed and the symbolic program state is further backtracked.

b) Unsat-core interpolation for unsatisfiable paths: Because unsatisfiable paths are not further explored, any potential error locations after the unsatisfiable branch are not

 ISBN: 978-1-61208-458-9

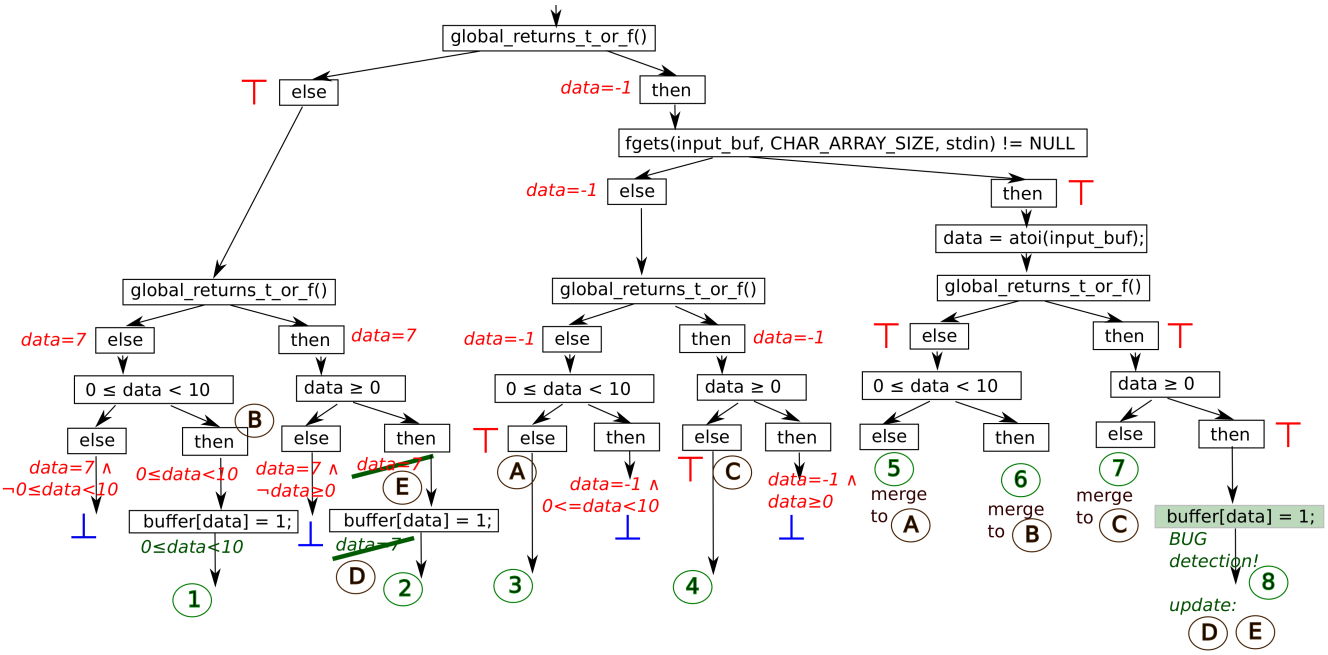


Figure 2. Algorithm progress for the example function from Figure 1

evaluated in this context. Another path can therefore only be merged as long as unsatisfiable branches remain unsatisfiable. This means, that an unsat-core for an unsatisfiable path is also treated as an error guard.

c) Backtracking error guards: Error guards are generated as unsat-cores in forward symbolic execution at the locations of unsatisfiable bugs, or during backtracking at unsatisfiable branch nodes. Backtracking also backtracks these formulas. The conjunction of backtracked error guards for one path (one execution tree node) is a merge formula. When a node's child is backtracked, then any formulas which were generated in this child (as symbolic interpretation) are removed from the node's error guards. Because constraints are removed, backtracking means a generalization of merge formulas. Decision nodes are the only control flow node type that has more than one child node, i.e., several branch nodes. The childrens' contribution to a decision node's error guard is determined during backtracking as the conjunction of the childrens' error guards. The reason is that path merging requires, that no new bug detection becomes possible on *any* extension of the current path. When backtracking reaches a branch node, a breakpoint is set and associated with the merge formula.

E. Example

To illustrate the algorithm, it is applied to the example function shown in Figure 1. The example is a 'bad' function from the Juliet suite [9], which contains a path-sensitive buffer overflow in line 17. The standard library functions `fgets()`, `atoi()` and `rand()` are treated as giving arbitrary (unconstrained) symbolic input. These functions are called in lines 5, 6 and 38. Static pre-analysis additionally yields breakpoints for lines 3, 13, 16 and 27 as input-dependent control flow decisions, and for lines 17 and 28 as input-dependent potential error locations. Together, these lines are indicated as shaded in the figure. Breakpoints are set on these lines, so that the debugger stops there and they are symbolically interpreted. The remaining not shaded locations are always just executed concretely, the debugger is not stopped for them. For the example, we assume that the function is called directly before program end, i.e., there are no backtracked formulas from other functions called later.

Algorithm progress is illustrated in Figure 2. The explored satisfiable program paths are marked with green numbers 1-8 in exploration order. Unsat-cores for unsatisfiable bugs are shown as green formulas (on path 1 and 2). Unsatisfiable branches are marked with a blue 'false' symbol ($\bot$, four times). Backtracked unsat-cores are shown as red formulas next to the respective control flow nodes. Merge locations are the branch nodes, i.e., 'then' and 'else'. The merge formulas are shown in red next to them. The 'true' symbol ($\top$) indicates an empty backtracked unsat-core (7 times). Path merges occur during the exploration of paths 5, 6, and 7. The respective merge targets are marked 'A' to 'C'. The bug is detected on path 8. This potential bug was unsatisfiable on path 2. The respective computed (backtracked) unsat-core is removed, because any re-detection of this bug on another path would be irrelevant. By removing constraints, more path merging can become possible in general. The updated tree nodes are marked 'D' and 'E', the removed constraints are crossed out in green.

Analysis results are also illustrated in Figure 1: merge locations and the corresponding merge conditions (implications) are indicated in red on the right side of the figure. For any further call of this function in the program under analysis, all paths can be merged at latest in lines 15 or 26 (because the respective implications are always valid).

III. A Coverage and Interpolation Hierarchy

Interpolation is an automated generalization of formulas. Through interpolation, interpolated path constraints may become equal. Here, interpolation by removing constraints

 ISBN: 978-1-61208-458-9

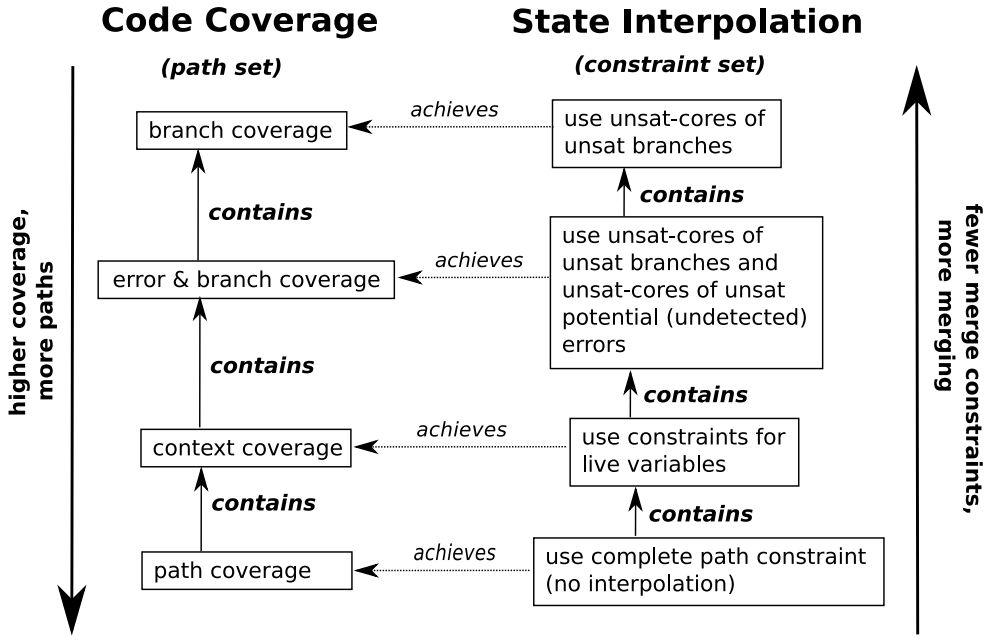


Figure 3. Coverage and interpolation hierarchy

from the path constraint during depth-first path exploration is considered.

Code coverage can be scaled by varying interpolation. Merge formulas are yielded from interpolation. With 'more' interpolation it is meant here to remove more constraints from the path constraint. With fewer constraints in merge formulas, more paths imply the merge formula and are pruned from the execution tree. The achieved coverage is given by the set of remaining paths.

Figure 3 illustrates four interesting algorithms and corresponding coverage. Unsat-cores are not unique. This corresponds to different path sets that can achieve the same coverage criterion. The annotation 'contains' for interpolation constraint sets on the right side of the figure assumes that unsat-cores are computed in the same way with serial constraint deletion.

A. Branch coverage

(Backtracked) unsat-cores of unsatisfiable branches are used as merge formulas. A path is only pruned if it implies the previously computed backtracked unsat-cores. This means that any extension of the (pruned) path can not cover any yet uncovered branch. Therefore, this interpolation achieves branch coverage. Branch coverage means that every branch in the program that can be covered with any program input is actually covered.

B. Error and branch coverage

This is the algorithm described in Section II. It uses unsat-cores for unsatisfiable branches and additionally unsat-cores of potential (and yet undetected) errors. This comprises unsat-cores necessary to achieve branch coverage. The additional constraints require to only prune a path when all previously unsatisfiable error conditions remain unsatisfiable. This means that any extension of the pruned path can not witness any yet undetected error. Error coverage means that every error that is satisfiable with any program input, and for whose potential existence a constraint is generated, is actually witnessed on a remaining (not pruned) path.

C. Context coverage

In backtracking, the sets of dead and live variables are exactly known. Live variables are the ones that are read on at least one extension of the current path. Only live variables can contribute to unsat-cores in a path extension. This interpolation therefore comprises the interpolation needed to achieve error and branch coverage. By removing dead constraints, path constraints can become identical. Context coverage means that every program location is covered in every distinct (live) context.

D. Path coverage

Complete path constraints (including constraints for dead variables) are used, no interpolation is done. Every path constraint is different, so no paths are pruned. Depth-first traversal without path merging achieves path coverage, i.e., every satisfiable program path is actually covered. This includes context coverage.

IV. Implementation

The implementation extends previous work, that is described in [11]. This previous work is a dynamic symbolic execution engine for Eclipse CDT, that does not have any path merging functionality.

A. Dynamic symbolic execution using Eclipse CDT

This subsection shortly review [11]. The engine is a plug-in extension for CDT's code analysis framework (Codan [12]). It uses CDT's C/C++ parser, AST visitor and debugger services framework (DSF [13]). The DSF is an abstraction layer over the debuggers' machine interfaces that are supported by CDT. The plug-in further uses Codan's CFG builder.

 ISBN: 978-1-61208-458-9

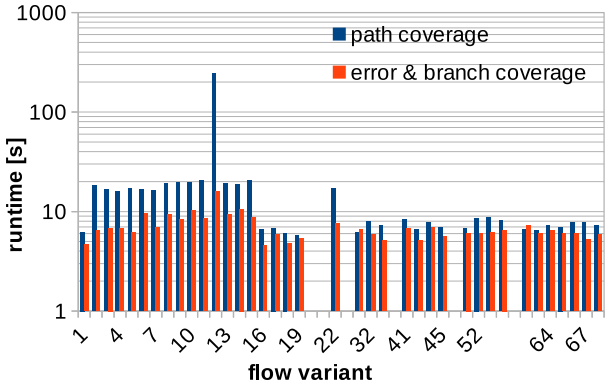


Figure 4. Run-times with and without path merging

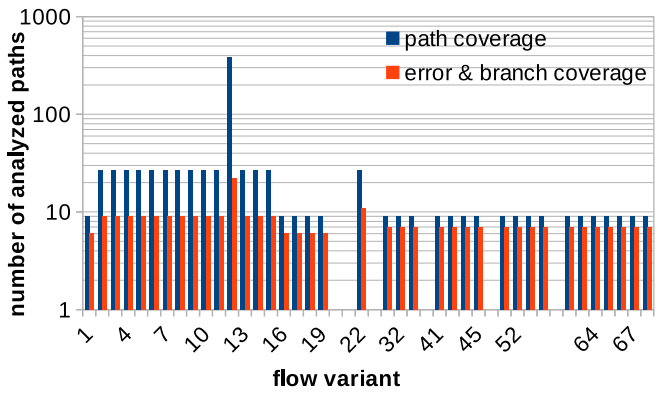


Figure 5. Number of analyzed paths with and without path merging

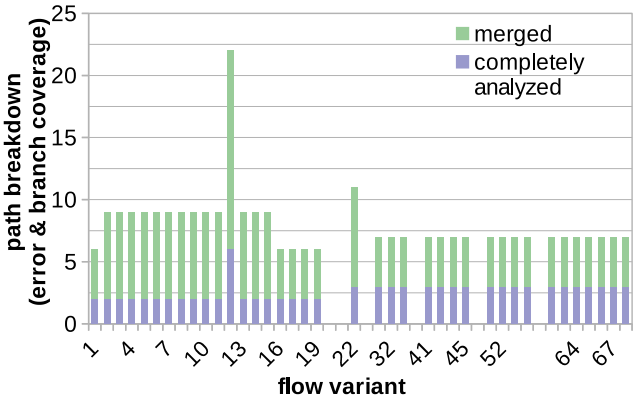


Figure 6. Breakdown of the number of analyzed paths with merging

Initial breakpoints are set on function calls that are configered to return unconstrained symbolic input values. Further breakpoints are set during symbolic execution for locations, where symbolic variables are used. Breakpoints are also set on pointer assignments, because pointer targets might become symbolic through assignment of a symbolic value.

When the debugger stops at a location, the corresponding CFG node is resolved. The AST subtree that corresponds to the CFG node is then interpreted according to the tree based interpreter pattern [14]. The visitor pattern [15] is used to traverse the AST subtree and translate it into an SMT logic equation. SMT queries are formulated in the SMTlib's [16] sublogic of arrays, uninterpreted functions and bit-vectors (AUFBV), and the Z3 SMT solver [17] is used to decide them.

B. *Interpolation and path merging*

Correct merging requires the 'maybe symbolic' static preanalysis described in section II. Interpolation based path merging means that more breakpoints are set than without merging. On a path that can not be merged, more locations are symbolically interpreted than without merging. The implication check for merging further requires variable projections as described in the following. Single assignments are used in the translation to logic to avoid destructive updates, i.e., single assignment names are used for variables in the logic equations. Because a merge formula was computed on a different path and the translation into logic uses single assignment names, a subset of variable names in both formulas (merge formula and path constraint) has to be substituted. These variable names are the last single assignment versions in both formulas of variables whose definitions reach the merge location. These variables are projected (substituted) to the corresponding syntax tree names (names in the source code). Because branch nodes are not necessarily explicit in the source node, the merge locations are not exactly branch nodes, but rather the next following program location where a debugger breakpoint can be set. This is the following expression or declaration with initializer. Merge location examples are given in Figure 1. The computation of unsat-cores with serial constraint deletion is straight-forward.

V. EXPERIMENTS

The implementation is evaluated with 39 buffer overflow test programs from the Juliet suite (buffer overflows with `fgets()`) that cover Juliet's different control and data flow variants for C. The test programs are analysed with the Eclipse plug-in, as JUnit plug-in tests. Eclipse version 4.5 is used on a i7-4650U CPU on 64-bit Linux kernel 3.16.0, with GNU debugger `gdb` version 7.7.1. The presented algorithm for error and branch coverage is compared with straight-forward dynamic symbolic execution without any merging (i.e., path coverage), as described in [11].

The results are shown in Figures 4, 5 and 6. The horizontal axes show the flow variant number. Juliet's flow variants are not numbered consecutively, to allow for later insertions in future test suite verions. Both the presented algorithm and the path coverage algorithm accurately detect the contained errors for all flow variants except for flow variant 21. In this flow variant (control flow depends on a static global variable), CDT's CFG builder falsely classifies a branch node as dead node, which leads to false negative detection.

A. *Speedup with merging*

The measured run-times both with and without path merging are shown in Figure 4. The figure's time scale is logarithmic. Despite of the additional analyses for path merging, there is a clear speed-up for all test cases. The biggest speed-up is achieved for flow variant 12, which also contains the largest number of satisfiable program paths.

B. Reduction in the number of analyzed paths

Figure 5 shows the number of analyzed paths with path coverage on the one hand and with error and branch coverage on the other. There is a clear reduction in the number of analyzed paths for all test programs. Merging prunes a subtree, which in general splits into more than one satisfiable program path. The figure shows a strong correlation with Figure 4, so that the reduction in the number of analyzed paths can be seen as the main reason for the speed-up.

C. Proportion of merged paths

Figure 6 shows a breakdown of the analyzed paths for merging only (error and branch coverage). The analyzed paths are distinguished into paths, that are completely analyzed until program end, and others, that are merged at some point. The figure shows that for all test cases the majority of analyzed paths is merged at some point, which is an additional reason for speed-up and explains another part of it (with the reduction of the analyzed lengths of the paths that are merged).

VI. Related Work

Work on symbolic execution spans over 30 years. An overview is given in [18]. Dynamic symbolic execution is presented in [3]. The concept of selective symbolic execution is described in [5]. The implementation uses breadth-first execution tree traversal without path merging. Path merging based on live variable analysis is presented in [6]. Path merging based on interpolation using unsat-cores is described in [7]. The latter approach is more comprehensive. It comprises elimination of constraints for dead variables, because those are not present in backtracked formulas. The interpolation based merging approach is used in a static symbolic execution tool for verification [19].

The work at hand differs in that it combines dynamic and selective symbolic execution with interpolation based path merging. Further, the efficient application to testing requires that merge conditions are updated in case of a bug detection, whereas a verification tool may terminate at the first error detection (when verification fails). The work at hand builds on the author's previous work described in [20], which performs static symbolic execution with path merging based on a live variables analysis. As already mentioned, it also builds on own previous work described in [11], which performs dynamic symbolic execution without any path merging.

VII. Discussion

Interpolation based path merging with the presented algorithm for error and branch coverage shows a clear speed-up already for the tiny Juliet test programs. Due to the improved scaling behaviour, it is expected to lead to increasing speed-ups for larger programs. Future work might include loop subsumption and the detection of infinite loops. Another point is the addition of checkers for different bug types.

Acknowledgement

This work was funded by the German Ministry for Education and Research (BMBF) under grant 01IS13020.

References

[1] J. King, "Symbolic execution and program testing," Communications of the ACM, vol. 19, no. 7, 1976, pp. 385–394.

[2] L. deMoura and N. Bjorner, "Satisfiability modulo theories: Introduction and applications," Communications of the ACM, vol. 54, no. 9, 2011, pp. 69–77.

[3] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed automated random testing," in Conference on Programming Language Design and Implementation, 2005, pp. 213–223.

[4] K. Sen, D. Marinov, and G. Agha, "CUTE: A concolic unit testing engine for C," in European Software Engineering Conference and International Symposium on Foundations of Software Engineering, 2005, pp. 263–272.

[5] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A platform for in-vivo multi-path analysis of software systems," in Int. Conf. Architectural Support for Programming Languages and Operating Systems, 2011.

[6] P. Boonstoppel, C. Cadar, and D. Engler, "RWset: Attacking path explosion in constraint-based test generation," in Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2008, pp. 351–366.

[7] J. Jaffar, A. Santosa, and R. Voicu, "An interpolation method for CLP traversal," in Int. Conf. Principles and Practice of Constraint Programming (CP), 2009, pp. 454–469.

[8] W. Craig, "Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory," The Journal of Symbolic Logic, vol. 22, no. 3, 1957, pp. 269–285.

[9] T. Boland and P. Black, "Juliet 1.1 C/C++ and Java test suite," IEEE Computer, vol. 45, no. 10, 2012, pp. 88–90.

[10] F. Nielson, H. Nielson, and C. Hankin, Principles of Program Analysis. Springer, 2010.

[11] A. Ibing, "Dynamic symbolic execution using Eclipse CDT," in Int. Conf. Software Engineering Advances, 2015, in press.

[12] E. Laskavaia, "Codan- a code analysis framework for CDT," in EclipseCon, 2015.

[13] P. Piech, T. Williams, F. Chouinard, and R. Rohrbach, "Implementing a debugger using the DSF framework," in EclipseCon, 2008.

[14] T. Parr, Language Implementation Patterns. Pragmatic Bookshelf, 2010.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[16] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB standard version 2.0," in Int. Workshop Satisfiability Modulo Theories, 2010.

[17] L. deMoura and N. Bjorner, "Z3: An efficient SMT solver," in Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2008, pp. 337–340.

[18] C. Cadar and K. Sen, "Symbolic execution for software testing: Three decades later," Communications of the ACM, vol. 56, no. 2, 2013, pp. 82–90.

[19] J. Jaffar, V. Murali, J. Navas, and A. Santosa, "TRACER: A symbolic execution tool for verification," in Int. Conf. Computer Aided Verification (CAV), 2012, pp. 758–766.

[20] A. Ibing, "A backtracking symbolic execution engine with sound path merging," in Int. Conf. Emerging Security Information, Systems and Technologies, 2014, pp. 180–185.

 ISBN: 978-1-61208-458-9

Verification of Architectural Constraints on Interaction Protocols Among Modules

Stuart Siroky, Rodion Podorozhny, Guowei Yang

Computer Science Department
Texas State University
San Marcos, TX 78666, USA
Email: {cs1773, rp31, gyang}@txstate.edu

***Abstract*—The importance of adhering to an adopted architectural style throughout software development and maintenance has been long recognized. This paper introduces an approach to efficiently checking the correspondence of architectural constraints on sequences of method invocations, i.e., interaction protocols involving more than two modules. Our approach combines parameterized slicing and non-deterministic symbolic execution. Slicing produces an executable portion of the bytecode of the system under analysis relevant to the given architectural property, and symbolic execution is applied to the slice to check all paths and all interleavings for any violations of the given architectural constraints. We have implemented our approach in a prototype, where IBM WALA library is used for slicing, Javassist is used to aid in the mocking of the unused code, and Symbolic PathFinder is used for symbolic execution. Two case studies on verification of Model-View-Controller systems have demonstrated the usefulness of our approach. In particular, property guided automatic slicing, in some cases, significantly reduces the size of the input to the symbolic execution, resulting in a reduction of verification time.**

Keywords–verification; architecture; symbolic execution; call graph.

I. INTRODUCTION

Software architecture was defined as a set of constraints on components, form and rationale by Perry and Wolf [1]. The importance of adhering to an adopted architectural style throughout software development and maintenance has been recognized by the software engineering community. It helps avoid architectural erosion and drift, so that the chosen architectural style continues providing its benefits and ensuring its correspondence to requirements.

A number of formal architectural description languages (ADL) have appeared over the years. For instance, Wright [2], is an example of ADL with an emphasis on specification of interaction protocols between modules as part of abstract behavior specification of components and connectors. Further work in the area of software architecture paid attention to automation of checking correspondence between the architectural prescription and implementation. The ArchJava tool [3] can serve as an example in this direction. The tool and associated ADL allows for definition of component ports and connectors and type checking of combinations of ports and connectors. It also allows for checking correspondence of a given implementation against an architectural specification.

In this work, we introduce an approach to checking correspondence of architectural constraints on sequences of method invocations, i.e., interaction protocols involving more than two modules. For example, constraints of this kind are defined in a popular Model-View-Controller (MVC) architectural style [4], [5]. The implementation of the approach for validation uses Java programming language. Thus, the analysis system processes bytecode for a Java Virtual Machine.

First, the approach uses multi-parameter slicing [6] so that to reduce the amount of the code to be analyzed. The prototype uses IBM's WALA [7] library to perform slicing. Next, mocking is performed with the help of Javassist bytecode manipulation library [8] to make the produced bytecode slice executable. Next, the approach uses symbolic execution [9], [10] via Symbolic PathFinder [11] to systematically explore all paths connecting the initial and final methods of interest so that to check if any architectural constraints are violated by the implementation.

The symbolic execution traversal checks if there are feasible paths that will break the constraints on legal method invocation sequences and builds path conditions to allow for test case generation along the legal invocation chains. In our approach, the search is directed only in the sense that the paths that do not connect the initial and final methods are not traversed. The symbolic execution traversal uses results of the slicing and mocking to explore a smaller state space. The symbolic execution is non-deterministic in relation to those variables whose values would not be fixed due to the choice of a source method. Thus, in case of a concurrent system, all interleavings in the slice would be traversed, increasing the assurance level of the analysis results.

We implement our approach in a prototype, where we use WALA [7] for calculating the slice, Javassist [8] to aid in the mocking of the code not contained in the slice and use Symbolic PathFinder (SPF) [11] for symbolic execution. Evaluation based on two case studies demonstrates the usefulness of our approach. In particular, property guided automatic slicing, in some cases, can significantly reduce the size of the input to the symbolic execution, resulting in a reduction of verification time.

The rest of the paper is organized as follows. Section II discusses related work. Section III presents our approach to verification of architectural constraints. Section IV evaluates our approach using two case studies. Section V concludes the paper with a discussion of some future work.

II. RELATED WORK

In this section, related work is described. The idea of slicing paired with symbolic execution is not new. Two closely related research projects that use a combination of slicing and symbolic execution are described below.

 ISBN: 978-1-61208-458-9

First is the work of Jaco Gendenhuys et al. titled "Probabilistic Symbolic Execution" [12]. In this work, the authors explore the adaptation of symbolic execution to perform a more quantitative type of reasoning – the calculation of estimates of the probability of executing portions of a program. They present an extension of the widely used Symbolic PathFinder symbolic execution system that calculates path probabilities. They exploit state-of-the-art computational algebra techniques to count the number of solutions to path conditions, yielding exact results for path probabilities. To mitigate the cost of using these techniques, they present two optimizations, PC slicing and count memorization, that significantly reduce the cost of probabilistic symbolic execution. Here, slicing and symbolic execution are paired together in a way that uses symbolic execution to calculate path probabilities to aid in the slicing. This differs from our focus on reducing the state space with slicing and then using symbolic execution for verification.

Another work that primarily focuses on the line reachability problem is that of Kin-Keung Ma et al. in "Directed Symbolic Execution" [13]. In this work, the authors study the problem of automatically finding program executions that reach a particular target line. This problem arises in many debugging scenarios; for example, a developer may want to confirm that a bug reported by a static analysis tool on a particular line is a true positive. They propose two new directed symbolic execution strategies that aim to solve this problem: shortest-distance symbolic execution (SDSE) uses a distance metric in an inter-procedural control flow graph to guide symbolic execution toward a particular target; and call-chain-backward symbolic execution (CCBSE) iteratively runs forward symbolic execution, starting in the function containing the target line, and then jumping backward up the call chain until it finds a feasible path from the start of the program. They also propose a hybrid strategy, Mix-CCBSE, which alternates CCBSE with another (forward) search strategy. The line reachability problem is very similar to the final point of interest in our problem. The difference is that they are trying to create test cases and the constraint is satisfied with a single path; while in our case, all possible paths between two points must be explored and the constraint must hold for all such paths.

The two projects mentioned above are related because they combine slicing and symbolic execution. Yet they do not focus on verification of architectural constraints in the area of software architecture research.

Next, related work in the software architecture is overviewed. The most prominent initial work on a formal architectural description language (ADL) is that by R. Allen on the Wright ADL [14] done in the early 1990s. In his work, R. Allen introduces a formal language for definition of protocols assigned to connectors in an ADL. The work itself focuses on description of the suggested formal ADL and does not contain applications of verifiers even though the author does suggest doing such verification with a SPIN model checker [15]. Another related work is by Jonathan Aldrich on a system for verifying consistency between a specification in a formal ADL and source code [3]. He developed a tool called ArchJava that allows for verification of topological and component constraints consistency between a prescribed architecture and source code under development. In his work though the protocols for communication among modules are not specified and not verified. The ArchJava stops at defining types of connectors and at checking if topological constraints of a software architectural prescription are adhered to. Finally, the work by S. Uchitel shows an application of a model checker LTSA to verification of protocols among modules defined in the UML sequence diagram [16]. The author creates an extension to LTSA model checker [17] by Jeff Magee and Jeff Kramer that is aimed at verifying for the lack of a deadlock, race conditions and event sequence properties based on protocols defined in sequence diagrams. The approach presented in this work differs in that it verifies the protocols on the produced bytecode via symbolic execution and uses slicing to create property specific slice of that bytecode.

III. Approach

Our approach combines property-guided slicing and symbolic execution. The kinds of properties we focus on are constraints on interaction protocols among modules. Such constraints are often part of architectural constraints on connectors.

An architectural constraint of this kind has a source and sink method invocations. Thus, we would like to determine a slice of the program that contains paths by which the execution threads can move from the source invocation to the sink invocation. Once, given a property, such an executable slice is produced, symbolic execution with constraints on the values of variables that correspond to the property is performed. Thus, our approach starts with a traversal of the analyzed system's call graph that, first, identifies the part of the call graph containing only the paths connecting a source and sink methods and, next, does a traversal of implementations of the methods inside that part of the call graph using symbolic execution [10], [9].

The symbolic execution traversal checks if there are feasible paths that will break the constraints on legal method invocation sequences and builds path conditions to allow for test case generation along the legal invocation chains at a later time. Thus, slicing prunes the method implementation of calls not relevant to the property. Yet, care must be taken to retain those calls on which there is either control or data dependency. Therefore, we are using a slicing implementation that constructs a proper system dependency graph (SDG). Furthermore, our approach makes sure that the obtained slice is executable so that it can be fed directly to symbolic execution. The symbolic execution traversal uses the call graph to avoid invocation of method calls that do not correspond to allowed transitions. Unlike [13], our approach needs to traverse all possible call graph paths between source and sink methods to show there is no violation.

For the initial validation of the approach we constructed a prototype that uses IBMs WALA library [7] for the slicing, Javassist [8] to aid in the bytecode manipulation when mocking the code to create an executable slice and Symbolic Path Finder (SPF) [11] to perform the symbolic execution for property verification. Even though we used WALA library, a slicing algorithm customized for this kind of architectural property that is explicitly constrained both by a source and a sink method would be more efficient. For instance, such an algorithm can be improved by concurrent traversal of the SDG starting simultaneously from the nodes that correspond to source and sink methods of a given property. These tools however could

 ISBN: 978-1-61208-458-9

```
public class SliceMockExample {
        private Obj1 object1;
        private Obj2 object2;

        public void initMethod() {
                object1 = new Obj1();
                object2 = new Obj2();

                object1.setValue(5);
                object2.method1();
                m1();
                m2();
        }

        public void m1() {
                …
                object2.method2();
                …
        }

        public void m2() { … }
} // end class SliceMockExample
```

```
public class Obj1 {
        private int value = 0;
        public void setValue(int val) { value = val; }
        public void m1() { … }
        public int addValue(int val) {
                value += val;
                return value;
        }
} // end class Obj1
```

```
public class Obj2 {
        Obj1 ob1;
        public void method1() { … }
        public void method2() { … }
} // end class Obj2
```

Figure 1. Mocking illustration

not be used for our prototype without modification to deliver an executable slice.

The summary of the steps of our approach is as follows. The first step is to define the architectural constraints that are of interest. The constraints will provide the source and sink methods based on which a slice will be calculated. The information from the calculated slice is used to mock the original bytecode into an executable slice. The modified bytecode can then be non-deterministically processed by the symbolic execution component which applies the constraints to all feasible paths in the slice. The implementation of this tool is illustrated below.

Once the constraints have been defined it is possible to begin calculating a slice. Several inputs are fed into the slicing component. The original byte code, a scope file, exclusion file, and the name of the class containing the main method. The scope file helps to define the set of code to be sliced. The exclusion file is used to ignore libraries and large bodies of code that can be safely ignored. This would include being able to ignore `java.math` or `java.io` if the code that is being processed did not use these libraries. Without the exclusion file there is too much library information for WALA to be effective. The name of the file containing the main method is needed so that WALA will know where to begin. With these four inputs WALA represents the code in a call graph. The call graph along with the final point of interest, the call and callee methods, a system dependence graph (SDG) are created by WALA. The final node of interest is also determined in the graph using the call and callee methods. Since the final method of interest in our constraint could be called in multiple places the point from which it is called, the callee method, is needed to determine the exact point the slicing should begin from.

The next step is to calculate the actual slice of interest. Using the call graph, SDG, final point of interest and the slicing options, a description of the slice can be calculated. The slicing options allow the user to determine the level of control and data dependence the slicer will use and the direction of the slicing calculation, forward or backward. For all of our experiments backward slicing was used to produce an executable slice of the code. The dependency option used for all experiments was full data and control dependence to help ensuring that the slice calculated would be an executable slice. The output description of the slice contains the methods, branches, statements and variables contained in the slice and their relation. This output description produced by WALA is not bytecode, nor is it easily transformed to bytecode, if at all. WALA also does not guarantee that the slice will be executable. The description of the slice is then parsed, extracting all of the methods that are contained in the slice.

The list of methods contained in the slice and the original bytecode for the program are used to create an executable version of the slice. First, a list of methods to mock needs to be determined. This list starts off by including all the methods in the original bytecode. From here, the list of methods contained in the slice is removed along with other sets of methods deemed to be needed. This list of methods that is not contained in the slice that is excluded from mocking (i.e., the methods whose complete implementation is needed for a slice to be executable, but that were not placed into the slice by WALA) contains the object constructor methods, any abstract methods, and base library methods. Excluding these methods from mocking eases the modification of the bytecode with little impact to the final result. The result is a list of methods that will be mocked in the code. The list of methods to be mocked is used to mock the original bytecode, producing modified bytecode files that can be executed. The mocking step finds the classes and methods in the code to be mocked and uses Javassist to remove the bodies of the methods and fix up the bytecode to keep it executable. If a method returns a value, an appropriate return type will be inserted back into the method body. The removal of the method body effectively removes any traversal further down that path. This approach to mocking effectively stubs out the remaining irrelevant code [18].

This is a simplistic and not completely efficient way of modifying the code. One caveat of doing the code modification in this manner is that if there is code that returns an object and that object then calls some other function. i.e., `getObjA().add(xxx)`, and then if the result of the `getObjA()` method returns a null after mocking the secondary call of `null.add(xxx)` then this invocation sequence will cause a null exception. Such a situation can be handled by adding the first method to a list of methods to exclude from mocking. A more precise approach would be to remove the invocation of the methods in question and the associated bytecode. It is sufficient to show the proof of concept for creating a reduced set of code. The modified class files are then written back and can be run by any test program just like the originally complicated code. In our case, the modified code is fed to the second part of the process, the verification of the architectural constraints using Symbolic PathFinder (SPF).

We implement a Java PathFinder [19] listener to monitor invocations and returns of methods maintaining a call stack. At the point where an invoke instruction is seen in the stack, it is checked against the constraints and a violation or a pass can be reported. Currently, constraint definition is done programmatically, i.e., it is hardcoded into the listener, albeit the use of a formal temporal logic appropriate for expressing the given properties would be preferred. This is left for future work.

Here, a small example for the slicing and mocking is described and illustrated in Figure 1.

 ISBN: 978-1-61208-458-9

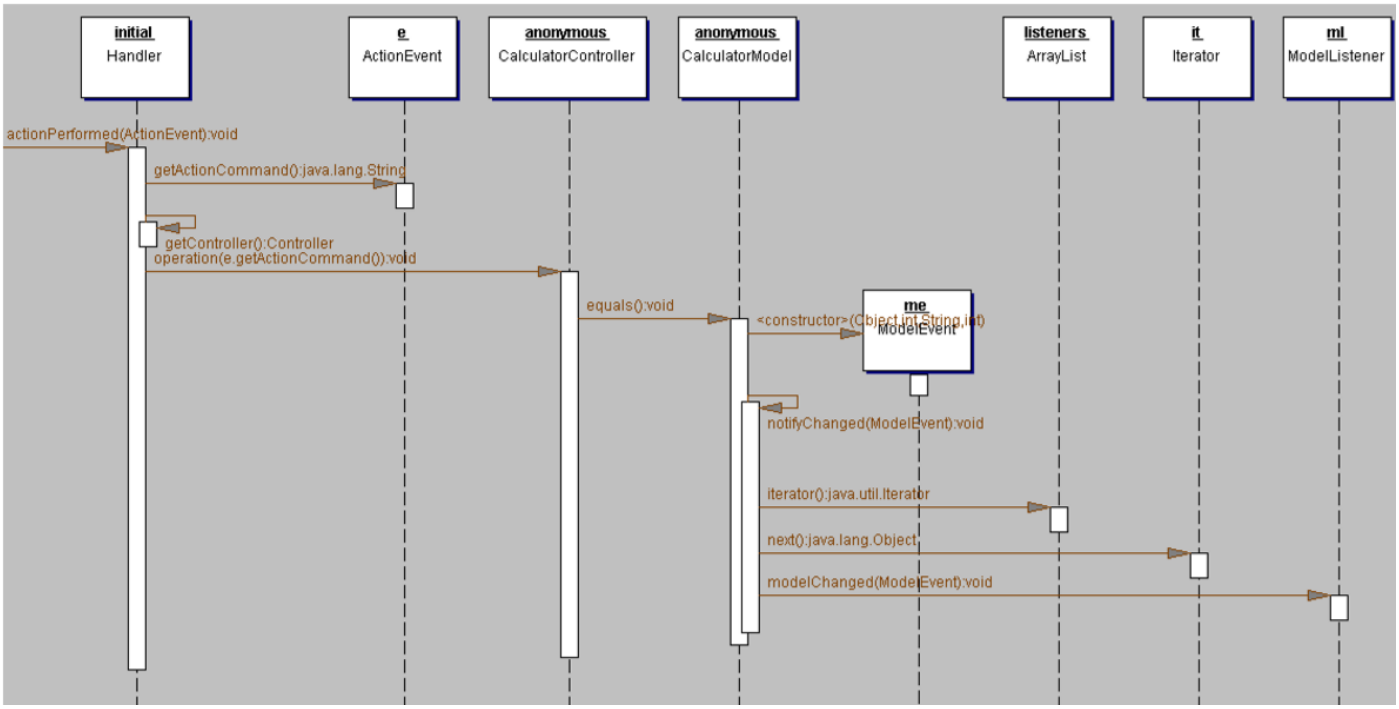


Figure 2. MVC Sequence Diagram (by John Hunt)

TABLE I. CALCULATOR MVC SYMBOLIC EXECUTION RESULTS

Techniques	# Paths	Runtime (SS)	# States	Max Depth	Memory (MB)	# Instructions
Orig; no Slicing	90	3	225	5	78	82,563
W 2 threads; no slicing	90	287	39,947	17	132	133,433,898
W 2 threads; w slicing	3	1	17	5	60	5,626

TABLE II. CALCULATOR MVC SLICING RESULTS

Actions	Time (SS)
Build Call Graph	4.37
Create SDG & Slice	7.78
Modify Bytecode	0.631
Total	12.82

In this example code, a slice is to be calculated for `Obj2.method2()`. The resulting slice should contain bytecode for the following sequence of method invocations: `SliceMockExample.initMethod()` → `SliceMockExample.m1()` → `Obj2.method2()`. The other methods will be mocked, having the code removed or replaced as necessary. The code to be removed and replaced is highlighted in blue in Figure 1. Even though the removed code in this illustration is small, a real world implementation will have more complicated and larger method implementations to be removed and mocked. There may also be many mocked methods. Thus, reduction in the amount of code to be removed from the analyzed system can be noticeable.

IV. EVALUATION

A. Case Study 1: MVC Calculator

For the first case study we chose a simplified implementation of a calculator that uses the Model View Controller (MVC) architectural style from [5]. The MVC architectural style can be considered to be composed out of the following design patterns: Component, Strategy, and Observer. Its implementation was simplified by replacing Swing library calls with stubs so that SPF would not execute the Swing library code. The test driver code was also added so that to mimic a scenario of user interactions with the calculator.

The constraint on an interaction protocol to be checked is due to the Strategy design pattern used by MVC.

It requires that a user gesture represented by an invocation of a method in the View should not directly invoke a method that modifies state of the Model. Instead, a View method should invoke a method in the Controller, which, in turn, should invoke a method in the Model. Thus, a Controller encapsulates strategies that map user gestures to manipulation of the Model. The Model is responsible for notifying the view of any changes. The sequence diagram in Figure 2 illustrates the event-based notification communication according to MVC [5].

Our implementation of the approach uses particular method names specific to this implementation when checking this

 ISBN: 978-1-61208-458-9

TABLE III. MODELCHECKCTL MVC SYMBOLIC EXECUTION RESULTS

Techniques	# Paths	Runtime (SS)	# States	Max Depth	Memory (MB)	# Instructions
Orig. No Slicing	4	2	5	2	76	74,851
W slicing	4	1	5	2	76	16,510

TABLE IV. MODELCHECKCTL MVC SLICING RESULTS

Actions	Time (SS)
Build Call Graph	3.54
Create SDG & Slice	19.8
Modify Bytecode	0.224
Total	24.22

constraint on interaction protocol between components. Specification of the constraint itself is currently implemented programmatically. The following constraints on interaction protocol were verified:

- The view should never update without notification from the model.
- The model should never notify if the controller has not issued an operation.
- The controller should not issue an operation if the handler has not created an action.
- The handler should not create an action if there is no activity in the view.

The pertinent classes of the calculator implementation under analysis include `CalculatorView`, `CalculatorController`, and `CalculatorModel`. The whole codebase under analysis contains many more classes, but these contain the methods used in the architectural constraint. The `CalculatorView` contains a method that mimics pressing a button by invoking a "button pushed" method on an instance of a given button. It also contains an inner class `Handler` with an `actionPerformed` method. It is this method that is invoked in response to a "button pushed" method. The `actionPerformed` is supposed to invoke an operation method from the `CalculatorController` class. The operation method, in turn, is supposed to invoke a relevant method from `CalculatorModel`. The buttons correspond to basic calculator operations: `addition`, `subtraction`, `store`, and `equals` (to perform the previously chosen operation between two operands). The constraint requires that an `actionPerformed` method should not invoke the `CalculatorModel` methods directly.

The verification tool prototype has been applied to several variants of the Calculator application to collect performance data. The tool with both slicing and symbolic execution was run on:

- the Calculator application that does not violate the given constraints
- the Calculator application that violates the given constraints
- the Calculator application that violates the given constraints and has a potentially long running execution paths that are not of interest to the given constraints (internal concurrency via multi-threading was used)

Also, the tool with symbolic execution alone was run on the `Calculator` variants with a violation and without a violation of the constraint as a baseline.

The results of the first case study are shown in Tables I and II. The codebase for the `Calculator` contains 787 total methods in the project, 99 of which are created and not inherited from the external libraries. After the slicing, the mocking removed 77 method bodies, a reduction of 77.78%. The size of the original bytecode was 42.5 KB (43,577 bytes), and was reduced to 29.9 KB (30,654 bytes), 70.3% of the original bytecode size. From the results, little gain can be seen between the original code and the sliced code other than a reduction in the number of paths that SPF traversed. The total time to verify, including the time slice and modify the code, was expensive – almost four times what the original code took to verify. However, when one compares the variant of the analyzed system with internal concurrency to the sliced code, the advantage of reducing the problem is more noticeable due to a large number of interleavings removed from non-deterministic execution by SPF. There are savings in total time of verification and the number of paths explored. In this case, slicing happened to remove a code block with internal concurrency. Analysis and modification of the code can be expensive; however, if it allows the removal of computationally expensive code, the benefit can be substantial. In this experiment, the symbolic execution time was reduced from 4 minutes and 47 seconds (287 seconds) to only one second.

B. Case Study 2: ModelCheckCTL

The second case study uses Computation Tree Logic (CTL) based model checker implemented in the MVC architectural style. The architectural constraints to verify are the same as for the first case study as enforced by MVC. This example is about twice as large compared to the `Calculator` example in terms of the size of the codebase. The Model part of the MVC itself has more components in that it contains classes for representation of a Kripke Structure, a CTL formula and a result representation. The `Calculator` example only contains one class in the Model part that encapsulates the calculator's current computation result and previously entered operand.

For brevity, we do not describe the implementation details for the constraint definition in this case study. The model checker under analysis provides a user with a GUI for definition of a CTL formula in a text field, choice of a file with a Kripke structure to be analyzed via a file chooser, and a text area that displays, textually, the result of model checking (i.e., whether a property holds and if not - a textual representation of a counterexample). The GUI has a number of buttons that are used to run the analysis, to clear the text areas, and to close the application. As the system was built according to MVC

 ISBN: 978-1-61208-458-9

architectural style, the verification performed is very similar to the first case study.

The results of this experiment are shown in Tables III and IV. The codebase for the model checker contains $1,097$ total methods, 258 of which are created and not inherited. After the slicing, the mocking removed 196 method bodies, a reduction of 75.97%. The size of the original bytecode was 96.4 KB ($98,802$ bytes), and was reduced to 64.1 KB ($65,736$ bytes), 66.5% of the original bytecode size. A significant amount of bytecode was removed by slicing. Also, we note that multiple properties can be verified on the slice, so even greater benefit can be achieved by performing slicing before symbolic execution.

V. CONCLUSION

In this paper, we introduced an approach to checking architectural constraints on sequences of method invocations via combination of slicing and symbolic execution. To our knowledge, this is a novel approach to property-guided verification of architectural constraints. Our approach would make verification of such properties more efficient by reducing the application to a potentially much smaller executable slice for a given increment of a software development process, thus significantly cutting down on time taken by regression testing both during development until release and maintenance after the release of the analyzed software system. Using slicing for the reduction of the problem size seems to hold the most promise for code that has low coupling. It is especially effective in cases where the code deemed not needed for property verification is long running and/or contains multi-threading. In such a case, the resultant slice is noticeably smaller than the initial codebase, and furthermore, some potentially long running code not needed for property verification will be removed from the slice.

This work shows proof of concept of the suggested approach, and there are a number of directions for further improvement. First of all, a slicing algorithm that takes multiple criteria (statements for which a slice is calculated) can speed up calculation of a slice. One approach might be to compute one slice using forward traversal starting from the source method of an interaction protocol property and another one using backward traversal from the sink method of the property and then use their intersection as a resultant slice. Any node already marked by the former slicing traversal would end the slicing exploration down that path by the latter slicing traversal and vice versa. Implementation of the slicing algorithm traversal from multiple nodes of an SDG could be done concurrently, further reducing the slicing time. As future work we also intend to add a general specification of the constraints via an appropriate logic and a test case generation capability based on the path conditions created by the symbolic execution of the property-guided slice. We would like to perform a quantitative comparative analysis against similar approaches. In addition, we would like to validate the prototype by applying it to analysis of larger systems.

REFERENCES

[1] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," SIGSOFT Softw. Eng. Notes, vol. 17, no. 4, Oct. 1992, pp. 40–52.

[2] R. Allen, "A formal approach to software architecture," Ph.D. dissertation, Carnegie Mellon, School of Computer Science, January 1997, issued as CMU Technical Report CMU-CS-97-144.

[3] J. Aldrich, C. Chambers, and D. Notkin, "Archjava: Connecting software architecture to implementation," in Proceedings of the 24th International Conference on Software Engineering, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 187–197.

[4] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," J. Object Oriented Program., vol. 1, no. 3, Aug. 1988, pp. 26–49.

[5] J. Hunt, "You've got the model-view-controller," Planet Java.

[6] F. Tip, "A survey of program slicing techniques," Journal of Programming Languages, vol. 3, March 1995, pp. 121–189.

[7] "IBM WALA," http://wala.sourceforge.net/wiki/index.php/Main_Page.

[8] "Javassist," http://jboss-javassist.github.io/javassist/.

[9] L. A. Clarke, "A program testing system," in Proc. of the 1976 annual conference, ser. ACM '76, 1976, pp. 488–491.

[10] J. C. King, "Symbolic execution and program testing," Communications of the ACM, vol. 19, no. 7, 1976, pp. 385–394.

[11] C. S. Păsăreanu and N. Rungta, "Symbolic PathFinder: symbolic execution of Java bytecode," in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '10, 2010, pp. 179–180.

[12] J. Geldenhuys, M. B. Dwyer, and W. Visser, "Probabilistic symbolic execution," in Proceedings of the 2012 International Symposium on Software Testing and Analysis, ser. ISSTA 2012. New York, NY, USA: ACM, 2012, pp. 166–176.

[13] K.-K. Ma, K. Y. Phang, J. S. Foster, and M. Hicks, "Directed symbolic execution," in Proceedings of the 18th International Conference on Static Analysis, ser. SAS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 95–111.

[14] R. Allen, "A Formal Approach to Software Architecture," Carnegie Mellon University, Technical Report CMU–CS–97–144, 1997.

[15] G. J. Holzmann, "The model checker spin," IEEE Trans. Softw. Eng., vol. 23, no. 5, May 1997, pp. 279–295.

[16] S. Uchitel, R. Chatley, J. Kramer, and J. Magee, "LTSA-MSC: Tool support for behaviour model elaboration using implied scenarios," in Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS. Springer Verlag, 2003, pp. 597–601.

[17] J. Magee and J. Kramer, Concurrency: State Models & Java Programs. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[18] "Mocking," http://www.michaelminella.com/testing/the-concept-of-mocking.html.

[19] "Java PathFinder Tool-set," http://babelfish.arc.nasa.gov/trac/jpf.

 ISBN: 978-1-61208-458-9