SOFTENG 2020

The Sixth International Conference on Advances and Trends in Software Engineering

ISBN: 978-1-61208-776-4

February 23 - 27, 2020

Lisbon, Portugal

SOFTENG 2020 Editors

Bernard Stepien, University of Ottawa, Canada

SOFTENG 2020

Forward

The Sixth International Conference on Advances and Trends in Software Engineering (SOFTENG 2020), held between February 23-27, 2020 in Lisbon, Portugal, continued a series of events focusing on the 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, e-health 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.

We welcomed academic, research and industry contributions. The conference had the following tracks:

- Challenges for dedicated software, platforms, and tools
- Software testing and validation
- Software requirements
- Maintenance and life-cycle management

We take here the opportunity to warmly thank all the members of the SOFTENG 2020 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 who dedicated much of their time and effort to contribute to SOFTENG 2020. We truly believe that, thanks to all these efforts, the final conference program consisted of top quality contributions.

We also thank the members of the SOFTENG 2020 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that SOFTENG 2020 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 historic charm of the city.

SOFTENG 2020 Chairs

SOFTENG Steering Committee
Mira Kajko-Mattsson, Royal Institute of Technology, Sweden
Yoshihisa Udagawa, Tokyo Polytechnic University, Japan
Ulrike Hammerschall, University of Applied Sciences Munich, Germany

SOFTENG Industry/Research Advisory Committee
Philipp Helle, Airbus Group Innovations - Hamburg, Germany
Tomas Schweigert, SQS Software Quality Systems AG, Germany
Michael Perscheid, Innovation Center Network, SAP, Germany
Janne Järvinen, F-Secure Corporation, Finland
Paolo Maresca, VERISIGN, Switzerland
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea

SOFTENG 2019 Special Tracks Chair
Raquel Lacuesta, University of Zaragoza, Spain

SOFTENG 2020

Committee

SOFTENG Steering Committee

Mira Kajko-Mattsson, Royal Institute of Technology, Sweden
Yoshihisa Udagawa, Tokyo Polytechnic University, Japan
Ulrike Hammerschall, University of Applied Sciences Munich, Germany

SOFTENG Industry/Research Advisory Committee

Philipp Helle, Airbus Group Innovations - Hamburg, Germany
Tomas Schweigert, SQS Software Quality Systems AG, Germany
Michael Perscheid, Innovation Center Network, SAP, Germany
Janne Järvinen, F-Secure Corporation, Finland
Paolo Maresca, VERISIGN, Switzerland
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea

SOFTENG 2020 Technical Program Committee

Khelil Abdelmajid, Landshut University of Applied Sciences, Germany
Issam Al-Azzoni, Al Ain University of Science and Technology, UAE
Washington H. C. Almeida, University of Brasilia - UNB, Brazil
Vu Nguyen Huynh Anh, Université Catholique de Louvain, Belgium
Darlan Arruda, University of Western Ontario, Canada
Jocelyn Aubert, Luxembourg Institute of Science and Technology (LIST), Luxembourg
Lerina Aversano, University of Sannio, Italy
Ali Babar, University of Adelaide, Australia
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea
Musard Balliu, KTH Royal Institute of Technology, Sweden
Imen Ben Mansour, University of Manouba, Tunisia
Marcello M. Bersani, Politecnico di Milano, Italy
Anna Bobkowska, Gdansk University of Technology, Poland
Luigi Buglione, Engineering Ingegneria Informatica SpA, Italy
Azahara Camacho, University of Cádiz, Spain
Pablo Cerro Cañizares, Universidad Complutense de Madrid, Spain
Sang Kil Cha, KAIST, Korea
Luciano de Aguiar Monteiro, Institute of Higher Education iCEV – Teresina-Piauí, Brazil
Amleto Di Salle, University of L'Aquila, Italy
Fernando Escobar, PMI-DF Brasilia, Brazil
Faten Fakhfakh, National School of Engineering of Sfax, Tunisia
Rob Fuller, University of British Columbia, Vancouver, Canada
Barbara Gallina, Mälardalen University, Sweden
Atef Gharbi, National Institute of Applied. Sciences and Technology, Tunisia
Jiaping Gui, NEC Laboratories America Inc., USA

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

Mapping on the Use of Games for Programming Teaching with an Emphasis on Software Reuse

Diego Castro, Cláudia Werner

COPPE/Computer Systems Engineering Program
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
{diegocbcastro, werner}@cos.ufrj.br

***Abstract*—Many works have already approached the use of games as a teaching method due to several advantages that this strategy can bring to the current teaching method. Therefore, a study was previously performed to identify games created for teaching software reuse fundamentals. However, no work addressing this problem was identified. Software reuse is an essential area of software engineering and is commonly associated with programming. Based on this, this article sought to identify works that had already been done on the use of games for programming teaching but could be used to teach reuse fundamentals.**

Keywords–game; game-based-learning; software reuse; programming; systematic mapping.

I. INTRODUCTION

Software Reuse (SR) is the discipline responsible for creating software systems from pre-existing software [1]. This concept is not just limited to code reuse; software in this context can refer to other products, such as modeling, specifications, test plans, and any other product in the life cycle of a project. With the correct use of this discipline, it can provide several positive impacts in a variety of contexts, such as quality, cost, productivity, code-making performance, rapid prototyping, reduced code writing, reliability, and interoperability of software [2].

Despite the advantages mentioned, SR is still far from being used on a large scale; many people reuse software, but not in a systematic manner. One of the main factors for reuse not being implemented is the difficulty of education in the area [3]. Based on this, a study was previously performed to identify games created for teaching software reuse fundamentals. However, it was not possible to identify a game specifically designed for this purpose.

In this initial research, it was observed that software reuse might be contained in different areas, such as programming and Software Engineering. Thus, this initial research was divided into two parts: the first to search for games to teach software engineering and the other to search for games for programming teaching with an emphasis on software reuse. Based on the information provided, this study aims to identify games that have the purpose of teaching programming with emphasis/potential for reuse, that is, to find games that were developed for teaching programming, but could be used to teach some of the fundamentals of software reuse, such as logical reasoning development, function development, object orientation, among others.

The remainder of this paper is presented as follows: Section II describes the research method used in the systematic mapping, Section III shows some results that were found, Section IV demonstrates an example of how one of the games found could be used to teach SR, Section **??** shows the threats to validity, and Section V concludes with the final remarks.

II. RESEARCH METHOD

Systematic mapping is a secondary study method based on a structured and repeatable process or protocol that explores studies and provides a result in the form of an overview of a particular subject [4]. The mapping presented follows the protocol proposed by Kitchenham [5].

The research process presented in this study covers articles published by the end of 2018 and aims to conduct a systematic mapping to identify work that has already been done on games for programming teaching but could be used to teach software reuse fundamentals, such as logical reasoning development, function development, object orientation, among others.

A. Research Questions

- **Q1:** What is the main advantage / motivation of the use of games to teaching programming language?
- **Q2:** What is the disadvantage of the use of games to teaching programming language?
- **Q3:** What is the main characteristic of the game used?
- **Q4:** What was the evaluation method used?

The mapping presented followed well-defined steps so that it was possible to reach a set of articles that were of interest to the search [5]. The search string was executed in Scopus as recommended by other studies [6] [7], and then the inclusion and exclusion criteria were applied to the set of articles that were found based on the title, abstract, and full text.

The **inclusion criteria** chosen were: **(1)** The article must be in the context of using games for teaching a programming language; **(2)** The article must provide data to answer at least one of the research questions; **(3)** The article should be written in English. The **exclusion criteria** chosen were: **(1)** Book chapters, conference call; **(2)** Studies that can not be fully accessed.

B. Search string and Analysis

The definition of the search string was based on the **P**opulation, **I**ntervention, **C**omparison, **O**utcome (PICO) structure [8], using three of the four levels. The search string was defined by grouping the keywords of the same domain with

 ISBN: 978-1-61208-776-4

the logical operator “OR” and grouping the two fields with the logical operator “AND”. However, we chose to use a date filter, searching only for articles that were published within five years, aiming to find more recent works in the area [9]. Table I demonstrates the PICO structure used in conjunction with the search string.

Initially, the search string returned a total of 507 papers. When analyzed according to the inclusion and exclusion filters, this number dropped to 17 papers. To minimize the lack of other search bases, considering that the study was only performed on Scopus, it was opted to use the snowballing procedure to minimize article loss, according to Motta et al. [6] and Matalonga et al. [7]. The approach was applied, searching for new papers through the references and through the papers that referenced these works. Thus, 9 more papers were included, totaling 26 analyzed papers. Table II shows how these 26 articles were obtained, and Table III shows each of these articles.

TABLE I. SEARCH STRING

PICO	SYNONYMS
Population	Programming language, algorithm experience, algorithm skills, algorithm alternative, algorithm method, coding experience, coding skills, coding method, coding alternative
Intervention	Tutoring, teach*,instruction, discipline, schooling, education*, mentoring, course, learn*,train*, syllabus
Comparison	Not applicable
Outcome	Game*, gami*, play*, “serious games”, edutainment, “game based learning”, simulation
SEARCH STRING	
TITLE-ABS-KEY ((”programming language” OR ”algorithm experience” OR ”algorithm skills” OR ”algorithm alternative” OR ”algorithm method” OR ”coding experience” OR ”coding skills” OR ”coding method” OR ”coding alternative”) AND (tutoring OR teach* OR instruction OR discipline OR schooling OR educat* OR mentoring OR course OR learn* OR train* OR syllabus) AND (game* OR play* OR ”serious games” OR gami* OR edutainment) AND (LIMIT-TO (PUBYEAR , 2018) OR LIMIT-TO (PUBYEAR , 2017) OR LIMIT-TO (PUBYEAR , 2016) OR LIMIT-TO (PUBYEAR , 2015) OR LIMIT-TO (PUBYEAR , 2014)))	

III. RESULTS

Section A demonstrates a discussion of the main results found in this work, and Section B presents the threats to the validity of this information exposed.

A. Discussion

The articles found in this study sought to demonstrate games that could be used in teaching some concepts related to programming. However, the analysis of the documents was performed in search of works that could be used to explain some of the concepts of reuse. From this, works that were not developed with this context but could be used for this purpose were also found. Figure 1 groups the articles by location and year of publication. It is possible to see an increase in the number of publications over the years and that many countries are looking for improvements in this area.

The bottom of the image also shows the number of articles found grouped by game type. However, some papers used more than one approach. From Figure 1, it is possible to observe that the most used way to teach programming is through the use of ”blocks of code”. By abstracting this idea it is possible to consider the concept of software components that is an important research area of SR and aims to build software from pre-produced components [10].

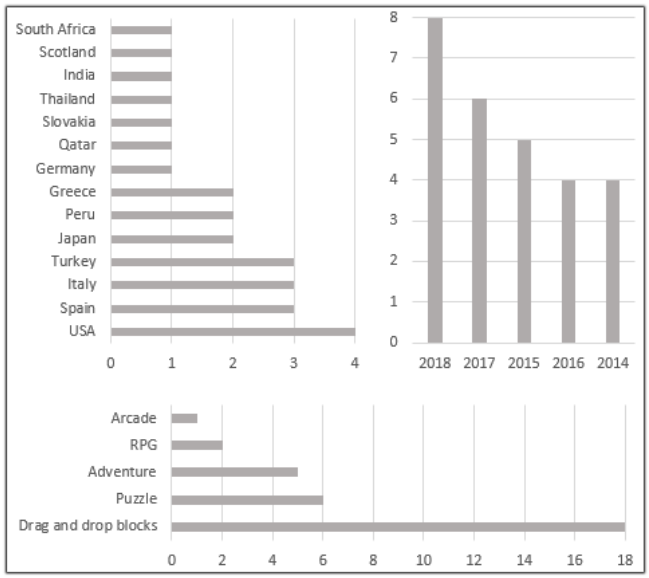

Figure 1. General analysis of the articles found.

Q1: What is the main advantage / motivation of the use of games to teaching programming language?

Using games as a reinforcement tool to teach skills can be a very beneficial strategy for students. They have proven to be a useful tool to complement conventional learning methods. Games allow visualizing concepts that may be too abstract. They also help you get acquainted with the knowledge and methods that may be tedious to study, offering a cycle of challenges and rewards that drives the learning experience [11].

Many authors claim that games have several characteristics that can benefit teaching [12] [13]. They have already been used as successful educational tools in many different fields and topics, such as engineering, learning languages, theater and even health [14]. The advantages include: increased student motivation and engagement, enhancement of pre-existing knowledge, increased performance in practical activities, immediate feedback, fun and satisfaction, among others [11] [15–19].

Q2: What is the disadvantage of the use of games to teaching programming language?

Despite the advantages offered by games as a teaching method, there are also some issues involving this approach. The first problem found was the comparison of the level of learning provided by a game as a teaching method and a class with textual programming. Despite the advantages offered by games, textual programming can still convey better content [20].

Another problem identified was the complexity of the game created. If the teaching tool used is too complicated, students can reduce the time spent solving problems to focus more on the tool. This is an unwanted distraction, and any game used should be easy to use, allowing the student to focus on solving the problem rather than how to use the game [21].

Finally, the last problem identified was that although games provide several advantages, they are not seen as self-sufficient.

 ISBN: 978-1-61208-776-4

TABLE II. ANALYSIS OF THE PAPERS

	Main Study		Snowballing backward		Snowballing Forward	
Activity	Result	Number of paper	Result	Number of paper	Result	Number of paper
First Execution	507 added	507	389 added	389	123 added	123
Repeated Papers	501 withdraw	501	294 withdraw	95	16 withdraw	107
Papers in another language	0 withdraw	501	14 withdraw	81	13 withdraw	94
Remove conference / workshops	16 withdraw	485	0 withdraw	81	0 withdraw	94
Remove books	0 withdraw	485	0 withdraw	81	0 withdraw	94
Remove by title	368 withdraw	117	46 withdraw	35	58 withdraw	36
Remove by abstract	83 withdraw	34	17 withdraw	18	18 withdraw	18
Papers not found	0 withdraw	34	0 withdraw	18	0 withdraw	18
Remove by full paper	17 withdraw	17	12 withdraw	6	13 withdraw	3
Total papers included	17 papers		6 papers		3 papers	
Extracted Papers	26 papers					

TABLE III. TRACEABILITY MATRIX.

Title	Year	Q1	Q2	Q3	Q4
Perceptions of Scratch programming among secondary school students in KwaZulu-Natal, South Africa	2018	X		X	X
Robo3: A Puzzle Game to Learn Coding	2018	X	X	X	X
Improving programming skills in engineering education through problem-based game projects with Scratch	2018	X		X	X
Introducing novice programmers to functions and recursion using computer games	2018	X		X	X
Introducing programming using “scratch” and “greenfoot”	2018	X		X	X
Developing Educational 3D Games With StarLogo: The Role of Backwards Fading in the Transfer of Programming Experience	2018	X	X	X	X
Learning to think and practice computationally via a 3D simulation game	2018	X		X	X
Design and implementation of Robo3 : an applied game for teaching introductory programming	2017	X		X	X
A cross-cultural review of lightbot for introducing functions and code reuse	2017	X		X	
Using Digital Game as Compiler to Motivate C Programming Language Learning in Higher Education	2017	X		X	X
Cubely: Virtual reality block-based programming environment	2017	X		X	X
Analysis of the learning effects between text-based and visual-based beginner programming environments	2017	X		X	X
Visual programming language for model checkers based on google blockly	2017	X		X	X
Educational resource based on games for the reinforcement of engineering learning programming in mobile devices	2016	X		X	X
Teaching abstraction, function and reuse in the first class of CS1 - A lightbot experience	2016	X		X	X
From Alice to Python Introducing text-based programming in middle schools	2016	X		X	X
Visual programming languages integrated across the curriculum in elementary school: A two year case study using Scratch” in five schools	2016	X		X	X
Building a Scalable Game Engine to Teach Computer Science Languages	2015	X		X	X
A mobile-device based serious gaming approach for teaching and learning Java programming	2015	X		X	
Coding with Scratch: The design of an educational setting for Elementary pre-service teachers	2015	X		X	X
Droplet, a Blocks-based Editor for Text Code	2015	X		X	X
Integrating Droplet into Applab – Improving the usability of a blocks-based text edit	2015	X		X	X
The development of a virtual learning platform for teaching concurrent programming languages in secondary education: The use of open Sim and Scratch4OS	2014	X	X	X	X
Effects of using Alice and Scratch in an introductory programming course for corrective instruction	2014	X		X	X
A structured approach to teaching recursion using cargo-bot	2014	X		X	X
The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners, Perspective, Informatics in Education	2014	X		X	X

Professional follow-up and feedback on the course are required to solve any problem that may arise throughout the learning process [21].

Q3: What is the main characteristic of the game used?

This study identified several games that sought to teach programming through increased motivation and engagement through fun. Most of these games were designed to be used by users with minimal or no knowledge of programming language [22].

The first game found was LightBot, which is a game to teach programming logic and has features such as multi-level, difficulty progression, feedback, challenges, use of similar tasks, concepts of functions, abstraction, flow control, recursion and code reuse [15] [16] [19]. Another game very similar to the one described above is Cargo-Bot, which has the same characteristics, but with other gameplay that revolves around a crane that moves and stacks a set of colored boxes. Players write small programs to move boxes from one initial setup to another [23]. Another game called Robo3 was found that had characteristics very similar to those described [11].

Another game very similar to the ones listed above was a game designed to teach Java programming that, to advance the levels, the player needs to overcome different levels. As the player surpasses these levels, he or she can progress through the story, unlocking new elements and gaining experience

points to unlock new content [22].

Another game found was Lost in Space, which includes, among other components, a game rules system, a physics engine, and a rendering engine. The game screen is divided into two parts. The left side containing the code interpreter text area and a help window and on the other side, the game phase. Through this game, some features were highlighted, such as obstacles, code interpreter (pseudocode of the game), collisions, movement, enemy and attack system [14].

In this research, we also identified some visual programming languages that are not considered as games directly, but that uses "block" approach to building programs. The first two to be identified were Alice [24] and Scratch [17], which are block-based visual programming languages designed to promote media manipulation for new programmers. From these languages, it is possible to upload media projects and scripts, animated stories, games, book reports, greeting cards, music videos, tutorials, simulations, and art and music projects. Two other languages very similar to those described are StarLogo TNG [20] and Droplet [25] [26], which are also drag-and-drop visual languages.

Greenfoot is an integrated tool that aims to teach object-oriented programming. Also, the tool allows teachers to introduce the most essential and fundamental concepts of object orientation in an easily understandable way [27]. Finally, the last visual language found is called Google Blockly [28], which is a library for building visual programming editors.

Finally, another feature that was used to create these games was the use of virtual and augmented reality. The Cubely game made use of these technologies to develop an idea that blended block programming concepts and the Minecraft game [29].

Q4: What was the evaluation method used?

Several evaluation methods were identified in this research. However, in general, all evaluations have a questionnaire applied to a specific population after using the tool to validate it [20] [24] [28].

Another possible means of the evaluation was the use of control groups where one group used the tool, and the other did not, and the same questionnaire was applied to both groups [14]. Through this assessment, it is possible to find out if there was a gain of experience through the tool use since it is possible to compare the results of the two groups.

The last evaluation method found was about the use of the tool as part of the discipline — the tool as a complement to the teaching of programming [30].

To conclude, games can be a new method to complement the current teaching method due to its main advantages, such as increased practice and engagement through challenges, rewards, fun, and feedback. However, it is still something new that needs attention due to problems such as the complexity of the game that can affect learning, and the level of learning provided by games is still lower than current teaching methods.

B. Threats to Validity

Through a critical analysis of the mapping, it is possible to perceive some threats that may have affected the final result of the work. The first to be highlighted is about the period in which the mapping was performed, collecting information from just five years. The second threat is the problem of interpreting the information found, which is up to the author to understand the game found and think of a way that could be applied in the teaching of SR.

IV. Reusing games to teach SR

This mapping found several games; however, none of them was produced to teach SR. Nevertheless, these games, with only a few or no modifications, could be used to explain certain concepts of software reuse, such as the importance of reusing, software components or code reuse.

Thinking about this idea of teaching SR, the platforms of Scratch, Alice, Droplet, and Google Blockly could, for example, be used to teach code reuse. All code that is generated with these platform is made from pre-produced blocks that resemble the idea of pre-produced components.

Robo3 and Lightbot are of puzzle type and are very similar, the general idea of these games is to create sequences of activities (which are described as functions) that perform a task, such as taking the avatar from point A to point B. Thinking about this type of games, these functions can be used in the game several times, teaching the student the concept of code reuse. Cubely is a Minecraft-based game; however, its mechanics are very similar to the two games described earlier and could also be used to teach code reuse.

V. Conclusion and Future Work

For many people who are not directly linked to the software reuse area, they refer to it as just code. Due to this fact, this mapping sought to find programming teaching games that could be used to teach reuse concepts that are often abstract to many students. From this, it was possible to identify six games and six block-based programming languages. The game, and the visual programming language that were identified in more articles were LightBot [15] and Scratch [17], respectively. The main characteristics found were the use of rules, phases, difficult progression, feedback, challenges, and the use of similar tasks in sequence.

As mentioned before, software reuse is inserted in several contexts, and the most common are propagation and engineering. This work sought to identify games that were created to teach programming but could be used to explain some of the fundamentals of software reuse, thus looking at works from the first context. To better understand how these games are used as teaching methods, it is intended to perform another mapping to identify games that aim to teach software engineering, since as software reuse is inserted in the engineering and possibly similar features can be used to the teaching of the two subjects.

Although this work has found some games that could be used to teach some reuse fundamentals such as components, functions, and object orientation, none of these games were specificaly designed to teach software reuse. Therefore, based on the characteristics that were found (multi-level, difficult progression, feedback, challenges, among others), it is intended to create a game with the specific purpose of software reuse teaching.

 ISBN: 978-1-61208-776-4

REFERENCES

[1] C. W. Krueger, "Software reuse," ACM Computing Surveys (CSUR), vol. 24, no. 2, 1992, pp. 131–183.

[2] J. Sametinger, Software engineering with reusable components. Springer Science & Business Media, 1997.

[3] N. Niu, D. Reese, K. Xie, and C. Smith, "Reuse a" software reuse" course," in American Society for Engineering Education. American Society for Engineering Education, 2011.

[4] I. Steinmacher, A. P. Chaves, and M. A. Gerosa, "Awareness support in distributed software development: A systematic review and mapping of the literature," Computer Supported Cooperative Work (CSCW), vol. 22, no. 2-3, 2013, pp. 113–158.

[5] B. Kitchenham, "Procedures for performing systematic reviews," Keele, UK, Keele University, vol. 33, no. 2004, 2004, pp. 1–26.

[6] R. C. Motta, K. M. de Oliveira, and G. H. Travassos, "Characterizing interoperability in context-aware software systems," in 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE, 2016, pp. 203–208.

[7] S. Matalonga, F. Rodrigues, and G. H. Travassos, "Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review," Journal of Systems and Software, vol. 131, 2017, pp. 1–21.

[8] M. Petticrew and H. Roberts, Systematic reviews in the social sciences: A practical guide. John Wiley & Sons, 2008.

[9] S. Jiang, H. Zhang, C. Gao, D. Shao, and G. Rong, "Process simulation for software engineering education," in Proceedings of the 2015 International Conference on Software and System Process. ACM, 2015, pp. 147–156.

[10] G. Sindre, E.-A. Karlsson, and T. Stålhane, "Software reuse in an educational perspective," in SEI Conference on Software Engineering Education. Springer, 1992, pp. 99–114.

[11] F. Agalbato, "Design and implementation of robo3: an applied game for teaching introductory programming," Scuola di Ingegneria Industriale e dell'Informazione, 2017.

[12] T. Jordine, Y. Liang, and E. Ihler, "A mobile-device based serious gaming approach for teaching and learning java programming," in 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE, 2014, pp. 1–5.

[13] R. Atal and A. Sureka, "Anukarna: A software engineering simulation game for teaching practical decision making in peer code review." in QuASoQ/WAWSE/CMCE@ APSEC, 2015, pp. 63–70.

[14] Á. Serrano-Laguna, J. Torrente, B. M. Iglesias, and B. Fernández-Manjón, "Building a scalable game engine to teach computer science languages," IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, vol. 10, no. 4, 2015, pp. 253–261.

[15] E. V. Duarte and J. L. Pearce, "A cross-cultural review of lightbot for introducing functions and code reuse," Journal of Computing Sciences in Colleges, vol. 33, no. 2, 2017, pp. 100–105.

[16] R. Law, "Introducing novice programmers to functions and recursion using computer games," in European Conference on Games Based Learning. Academic Conferences International Limited, 2018, pp. 325–334.

[17] D. Topalli and N. E. Cagiltay, "Improving programming skills in engineering education through problem-based game projects with scratch," Computers & Education, vol. 120, 2018, pp. 64–74.

[18] N. Pellas and S. Vosinakis, "Learning to think and practice computationally via a 3d simulation game," in Interactive Mobile Communication, Technologies and Learning. Springer, 2017, pp. 550–562.

[19] M. Aedo Lopez, E. Vidal Duarte, E. Castro Gutierrez, and A. Paz Valderrama, "Teaching abstraction, function and reuse in the first class of cs1: A lightbot experience," in Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education. ACM, 2016, pp. 256–257.

[20] N. Boldbaatar and E. Şendurur, "Developing educational 3d games with starlogo: The role of backwards fading in the transfer of programming experience¡? aq1?¿," Journal of Educational Computing Research, vol. 57, no. 6, 2019, pp. 1468–1494.

[21] N. Pellas, "The development of a virtual learning platform for teaching concurrent programming languages in the secondary education: The use of open sim and scratch4os," Journal of e-Learning and Knowledge Society, vol. 10, no. 1, 2014, pp. 129–143.

[22] A. Sierra, T. Ariza, F. Fernández-Jiménez, J. Muñoz-Calle, A. Molina, and Á. Martín-Rodríguez, "Educational resource based on games for the reinforcement of engineering learning programming in mobile devices," in 2016 Technologies Applied to Electronics Teaching (TAEE). IEEE, 2016, pp. 1–6.

[23] E. Lee, V. Shan, B. Beth, and C. Lin, "A structured approach to teaching recursion using cargo-bot," in Proceedings of the tenth annual conference on International computing education research. ACM, 2014, pp. 59–66.

[24] C.-K. Chang, "Effects of using alice and scratch in an introductory programming course for corrective instruction," Journal of Educational Computing Research, vol. 51, no. 2, 2014, pp. 185–204.

[25] D. Bau, "Droplet, a blocks-based editor for text code," Journal of Computing Sciences in Colleges, vol. 30, no. 6, 2015, pp. 138–144.

[26] D. A. Bau, "Integrating droplet into applab—improving the usability of a blocks-based text editor," in 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). IEEE, 2015, pp. 55–57.

[27] H. Chandrashekar, A. G. Kiran, B. Uma, and P. Sunita, "Introducing programming using "scratch" and "greenfoot"," Journal of Engineering Education Transformations, 2018.

[28] S. Yamashita, M. Tsunoda, and T. Yokogawa, "Visual programming language for model checkers based on google blockly," in International Conference on Product-Focused Software Process Improvement. Springer, 2017, pp. 597–601.

[29] J. Vincur, M. Konopka, J. Tvarozek, M. Hoang, and P. Navrat, "Cubely: Virtual reality block-based programming environment," in Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology, ser. VRST '17. New York, NY, USA: ACM, 2017, pp. 84:1–84:2. [Online]. Available: http://doi.acm.org/10.1145/3139131.3141785

[30] L. A. Vaca-Cárdenas, F. Bertacchini, A. Tavernise, L. Gabriele, A. Valenti, D. E. Olmedo, P. Pantano, and E. Bilotta, "Coding with scratch: The design of an educational setting for elementary pre-service teachers," in 2015 International Conference on Interactive Collaborative Learning (ICL). IEEE, 2015, pp. 1171–1177.

 ISBN: 978-1-61208-776-4

An Overview of SAP Core Data Services

Add Belati
Corporate Application Department
Saudi Aramco
Dhahran, Saudi Arabia
email: add.belati@aramco.com

Firas Alomari
Corporate Application Department
Saudi Aramco
Dhahran, Saudi Arabia
email: firas.alomari@aramco.com

***Abstract*—Increasing amount of data and the diversity of available data structures enable applications that can make timely decisions based on live data that, in most cases, can be evaluated without traditional application layer processing. SAP introduced the Core Data Service (CDS) framework as a data modeling approach in which Virtual Data Models (VDM) are defined at the database layer. CDS improve applications performance by pushing data processing from the application to the database layer to reduce data movements. In this paper, we present some technical insights into SAP's CDS, including new application patterns supported by CDS and the motivations behind it. We also discuss some practical considerations and challenges that may arise with CDS adoption and implementation.**

***Index Terms*—ERP; Programming; Database; Code Pushdown**

I. Introduction

Enterprise Resource Planning (ERP) and Business Intelligence (BI) systems are a fundamental part of today's enterprise IT applications portfolio. They provide a set of standardized software packages that capture interdisciplinary business processes across the entire value chain of an enterprise in a streamlined fashion [1]. ERP and BI systems integrate business functions with a centralized data repository shared by all business processes in the enterprise. The information provided by these systems drive daily business operations and provides for the development of new business ideas.

ERP systems were designed to capture daily operational and transactional business data. This kind of data processing typically referred to as Online Transactional Processing (OLTP) [2], uses row-based operations to efficiently process transactions, instantly recording business events (e.g., payments) and reflecting changes as they occur. However, they are not efficient at performing set-wide operations on entire tables. BI or Data Warehouse systems, on the other hand, were designed as Online Analytical Processing (OLAP) systems, to provide analytical and trend reporting from the growing data in the ERP systems. They leverage column-oriented tables to speed up operations over a huge volume of data at the expense of efficiency in executing OLTP workload.

OLTP and OLAP systems evolved separately to prevent the long-running and resource-intensive OLAP workload from decreasing the transactional throughput of the OLTP system [3]. Specifically, OLAP solutions were running analytical queries on a copy of the transactional data (i.e., views) from OLTP data stores [4]. This enabled companies to efficiently address a growing number of conflicting business needs, albeit, at the expense of increased complexity to link, orchestrate and synchronize multiple systems in the IT infrastructure. Therefore, unified Analytical Transaction Processing (ATP) systems were proposed to perform fast analytical processing coupled with transactional data management [4], preferably, by merging operational and analytical systems into one single system. ATP systems run the analytical queries directly on top of the transactional data to enable real-time data operations, reduce IT complexity and lower the total cost of ownership.

Systems, Applications and Products (SAP) introduced the High-Performance Analytic Appliance (HANA) [5] to provide a unified ATP system that fulfils the aforementioned requirements of business applications. It provides a data management platform to support efficient processing of both transactional and analytical workloads on the same physical database. It supports a code pushdown approach to execute data-intensive processing in the database close to the raw data (i.e., code-to-data) to reduce expensive data movement and improve applications performance. To take advantage of this code pushdown concept, SAP introduced CDS framework as a data modeling infrastructure to enable data reusability, extensions, and integration in HANA. Later, CDS were also introduced to the SAP ABAP application stack to enable developers to take advantage of the code pushdown with other databases.

In this paper, we introduce CDS and the motivation behind it. We also discuss some of CDS implications on application development strategy. The paper is organized as follows: CDS are described in Section II. In Section III we discuss CDS advantages and present some practical considerations related to code-push-down with CDS. We conclude the paper and present future research directions in Section IV.

II. SAP Core Data Services (CDS)

In this section we present a brief background of code pushdown and introduce the data services layer provided by CDS. Additionally, we describe CDS technical features and enhancements.

A. Background

The availability of multi-model databases that support complex data structures such as spatial, text, graph, and time series data enables built-in advanced analytics capabilities, such as

 ISBN: 978-1-61208-776-4

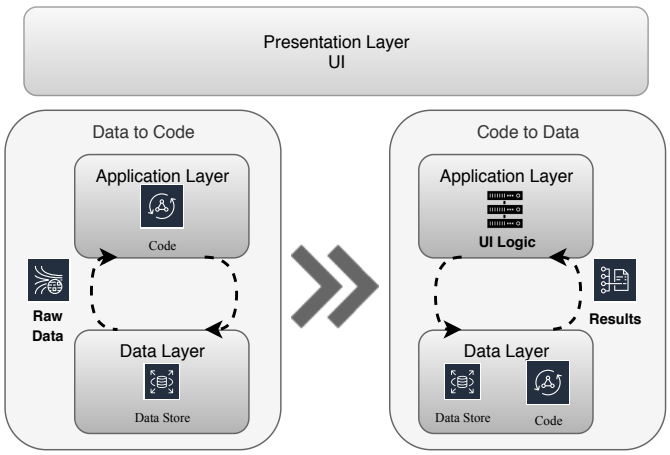

Fig. 1. Code Pushdown Model.

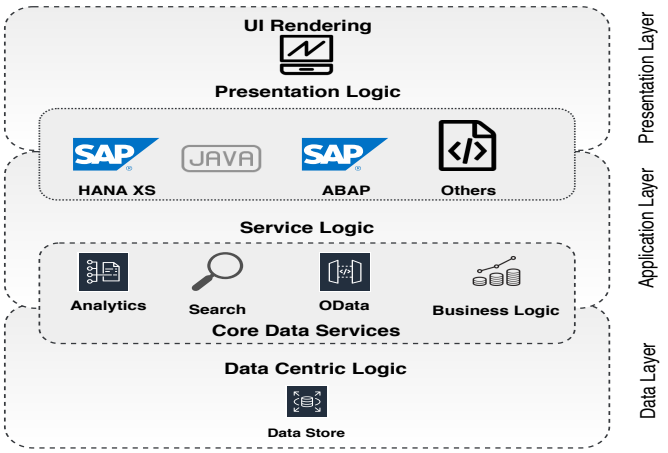

Fig. 2. Core Data Services Model.

text mining, spatial analysis and predictive analytics.These capabilities support diverse application patterns ranging from transactional systems to decision and analytic support systems in real-time and on live data [6]. However, as the size of data grows, moving data becomes the bottleneck and the cost of moving data around becomes prohibitive. Yet, in many cases, the data can be evaluated and processed on the data storage without the need for traditional application layer processing.

Let us consider the example of one million values, representing different currencies, that need to be calculated. Typically, the individual values are transferred to the application server from the database to only be discarded after the conversion and calculation has been done. Therefore, it's advantageous to move the data less by running workloads in the data storage or as close to it as possible [7]. This concept of code pushdown or code-to-data (see Figure 1) improves the applications performance by executing application logic inside the database, thus, decreasing the amount of data transfer and round-trips between the database and application layers.

B. Data Services Layer

Technically, applications communicate with databases through a number of various interfaces such as Standard Query Languages (SQL), Stored Procedures, SQL Scripts, or specific APIs. In the code pushdown concept, data services such as search or predictive analytics use these interfaces to push the computation to the database layer and only move results to application layer. Therefore, a common abstraction layer for integrating database interfaces and data services together is necessary. To this end, SAP introduced a data service layer (i.e., CDS) as a common abstraction layer for integrating these interfaces and data services together. This layer provides data models for defining and formatting data sources consistently across multiple systems, enabling different applications to share the same data, thus reducing development costs and time as well as improving the quality and performance of the applications.

The CDS data modeling infrastructure uses specific domain language to define different data services in a unified data definition and query language [8]. These data models are defined and consumed on the database server rather than on the application layer. They can be further enriched with semantics to allow developers to define entity types (e.g., orders or products) and the semantic relationships between them, which correspond to key relationships in traditional entity relationship (ER) models.

In particular, CDS offer central definitions that can be used in many different application context, such as transactional and analytical applications, to interact with data in the database in a unified way. They provide a cross platform unified data abstraction layer that sits between the database and client applications (see Figure 2). They function as proxy that drives data intensive computation to the database and exposes only relevant results to be consumed by different applications. Similar to Open Data Protocol (OData) for User Interface (UI) abstraction where UI rendering is pushed up to the client, CDS push down data-intensive calculations to the data layer to get the benefits of the underlying databases high-performance capabilities such as fast in-memory column operations, query optimization, and parallel execution.

C. CDS Description

CDS models are expressed in a Data Definition Language (DDL). DLL is based on standard SQL with some enhancements, such as associations, extensions, and annotations. The CDS Query Language (QL) is an extension to SQL used to consume CDS data. The QL includes enhancements such as defining views within the CDS data model. It also introduces the use of associations defined by the DDL. Instance-based authorization to CDS entities are defined using the Data Control Language (DCL). DCL can leverage literal conditions that compare elements of a CDS entity with literal values such as organization code, thus, pushing granular authorization filters and restrictions to the database for better performance.

 ISBN: 978-1-61208-776-4

They can also integrate with traditional authorization concept in SAP to check against existing authorization objects.

There are two variants of CDS: ***ABAP*** and ***HANA***. ***HANA CDS*** are HANA database dependent entities residing on the database itself. They enable the creation of database tables, views and data types using the native HANA DB SQL statements, enriching them with semantical properties, and using HANA native functions to perform data intensive computations. The HANA CDS does not require a specific application stack and therefore can support a variety of technologies and programming languages.

On the other hand, ***ABAP CDS*** provide a framework for defining and consuming semantic data models on the central database of the ABAP application stack. One can use SQL like statements to create and deploy the corresponding CDS entity on the target database automatically. Simillar to HANA CDS the models are based on the DDL and DCL, which are managed by ABAP Dictionary. However, unlike HANA CDS, ABAP CDS are database independent.

Virtually, design principles for both ABAP and HANA CDS are the same but due to differences in the respective environments, some technical differences between these flavors evolved. One clear distinction is that ABAP CDS access control and authorization can support traditional ABAP-based authorizations or defined in the DCL of the CDS entities. Both methods can be used independently or together, however the traditional authorization concept allows for the reuse of existing authorizations in the ABAP system.

Technically, CDS are an enhancement of the standard SQL that provides a data modeling framework for developers to define CDS entities, such as tables, views, and user defined data structures in the database. The CDS entities capture the semantics of the data to join the data needed for the application into one single model. There are two types of CDS entities: ***views*** and ***table functions***.

The CDS ***views*** are defined for existing tables or views in the database. They can be used to rearrange table fields based on the application needs. The views can have additional input parameters to filter the data during selection process at database level itself. So there is no need for a where condition in the application layer code.

Alternatively, CDS ***table function*** views include computations for database tables that are used by other CDS views, such as date and time calculation and conversion functions. Similar to CDS views, table functions can have additional parameters. Both type of views provide a number of major capabilities and enhancements over standard SQL, such as:

- *Joins*: are used to group fields from one or more different tables or views. Joins such as INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, or UNION are supported.
- *Associations*: are joins on demand that get executed when specifically used in a query. They are reusable, and allow a CDS view to be linked with other data sources, such as classical tables and views or another CDS entity, with a varying degree of cardinality. Cardinality and simple path expression in queries are the most important benefits of associations views.
- *Annotations*: define properties and behavior at run time. For example, one can save views or link another column during run time using the "@" character in the CDS view definitions. It further allows parameters to be used or meta data to be added to the CDS view data.
- *Expressions*: are used for calculations such as aggregation and mathematical computation in the data model queries.
- *Extensions*: are views with additional information that is not in the original table. They reduce data movement by adding expressions, associations and additional columns to an existing view.

D. Example

We show two examples of using CDS in Figure 3. The CDS are used to build a data model and service definitions on a conceptual level. Specifically, CDS models are translated into native database artifacts (e.g., schema) and interpreted to services to be exposed to applications. Specifically, at line 4 the OData annotation generates OData service automatically. The OData is then consumed by the application to enable the fuzzy search shown in the example. Further, at line 11 the example shows additional annotations that are used to enable other visual elements such as default values and enabled UI controls. In lines 23, the example also shows the currency conversion scenario mentioned in Section II-A using CDS table function. Besides the performance enhancement with this approach, the data abstraction layer makes it easier to define semantically rich data models. These models can be used for easy data access and allow for reuse in different types of application.

III. Discussion

Code pushdown or code-to-data concept is not a new one. In fact, some would argue that database stored procedures offer a similar performance advantage to CDS entities. However, stored procedure languages offer abstractions close to the database layer and they often lack concepts to express the application semantics. Moreover, with stored procedures one would lose the advantages of the CDS unified development environment by introducing code into the system that is more difficult to maintain, challenging to test, often frustrating to debug and needs to be integrated and managed through diverse database interfaces [9]. With CDS, one can make use of SAP's available tools to assist developers during CDS development, testing, and in their deployment to the production systems. Practically, CDS objects are integrated in the repository of SAP applications artifacts for complete life cycle management. They are like any other SAP development objects, subject to the transport system within SAP so that they can be easily deployed or transported from development via test to a production system consistently.

 ISBN: 978-1-61208-776-4

```
@AbapCatalog.sqlViewName: 'ZCDS_UNTCNV'
@AccessControl.authorizationCheck: #CHECK
//Annotation to generate the OData service automatically
@OData.publish: true
  define view zdemo_converesion
   with parameters to_unit:abap.unit(3)
   as select from ZX_TABLE
   {
    id,

    //Enabling the fuzzy search on the UI level
    @Search.defaultSearchElement : true
    @Search.fuzzinessThreshold : 0.8
    @Search.ranking : #HIGH
    ProductDescription.text as Name,

    @Search.defaultSearchElement : true
    @Search.fuzzinessThreshold : 0.7
    @Search.ranking : #LOW
    Product.category as Category,


    dec3 as original_value,
    //Using SAP standard convrsion expression to convert
    //values of operand to dictionary specific data type
    cast( 'MI' as abap.unit(3) ) as original_unit,


    //Utilizing SAP standard Table Functions for conversions between units
    unit_conversion( quantity => dec3,
                     source_unit => cast( 'MI' as abap.unit(3) ),
                     target_unit => :to_unit,
                     error_handling => 'SET_TO_NULL' ) as converted_value,
    :to_unit as converted_unit }
```

Standard * Notebooks HT- Hide Filter Bar

To show filters here, add them to the filter bar in Filters

Standard *

Product Category	Product ID
Notebooks	HT-1000
Notebooks	HT-1001
Notebooks	HT-1002

Standard filter allows to search for product category and product name

Fig. 3. Core Data Services Examples.

Applications based on CDS framework implies a major shift in development practices. Specifically, CDS shift the development paradigm from a process-centric activity to a data-centric activity [4]. For example, applications need to be designed to take advantage of CDS features and define new data hierarchies that can be readily used by applications. Since development is driven by the need to eliminate or reduce data movement from the database into application servers, which, in fact, is the main bottleneck for data-intensive applications in traditional three-tier architecture. Developers have to identify data-intensive parts of their application [10]. These parts can then be redesigned -leveraging CDS views- to push down the computation into the data layer. However, identifying data-intensive parts is not necessarily a trivial exercise. Therefore, Intuitive techniques for describing and modeling data are necessary [3]. One can guide this by best practices, design patterns, code analysis, performance profiling, etc.

Furthermore, a data-centric approach reduces the applications code footprint. Specifically, with consistent data models across systems, different applications can reuse the same models [7], [3]. Models can also be extended or built on top of other models (i.e., stacked) to meet the application needs. Service definitions such as OData and REST APIs can then be designed and exposed using the necessary models. Subsequently, user interfaces can be developed independently of the data and service definitions. In fact, with this approach application development becomes more of a service orchestration activity rather than a programming activity.

Concisely, the development practices should be guided by the following three principles: First, UI Rendering with presentation logic should be pushed up to the Client (i.e., browser, mobile apps). Second, data-intensive computation is done in the database and only the results are moved to the application server. Specifically, application server should handle control-flow and procedural logic only. Third, development artifacts from all layers are managed in a central repository so they can be easily created, tested, integrated and deployed.

This will also introduce a number of challenges that should be considered in our development methodology [7], [11]. For example, requirements should not only capture how the process works but they should also describe the data acquisition process. Similarly, analysis activities need to carefully consider data sources, transformation and context as well as the traditional analysis of business processes. Finally, data-centric development requires additional effort to ensure that effective data quality controls are in place. Therefore, testing should expand from verifying application functionality to ensuring application data quality are validated, and validation rules are carried over to operations and maintenance of the application. These are only some of the challenges that must be considered in the development life cycle.

IV. Concluding Remarks

CDS offer easy-to-understand, reusable tools to help realize code push-down (i.e., Code-to-Data) model. With CDS, it is possible to build applications that integrate application control logic in the database layer to achieve real-time performance. It may require additional effort to move logic down to the

 ISBN: 978-1-61208-776-4

database level. However, it reduces complexity and leads to simplification in data models and applications, redefining application development practices. Developers of business applications must therefore make careful choices about what data operations to include and what to omit. Certainly, it's not feasible or economically viable to model all existing applications data operations in CDS. Alternatively, if too little is included, the models may not support application needs adequately, which pushes more development effort and cost onto the application developers. This shift in application development practice presents new challenges, however, it provides opportunities to support new kinds of interactive applications, which were not possible before.

In future research, we plan to investigate processes that are appropriate for the CDS concept in our ERP system. Furthermore, we plan to evaluate costs and benefits of transforming existing business application to make use of the CDS concept. Specifically, we plan to develop criteria to guide the developers in identifying which applications or parts of an application are more appropriate for CDS. One idea we have is to utilize performance-profiling tools to prioritize applications for further evaluation. Furthermore, we plan to evaluate our development methodology and identify how it needs to be adapted to meet the expectations of this transformation.

REFERENCES

[1] Z. P. Matolcsy, P. Booth, and B. Wieder, "Economic benefits of enterprise resource planning systems: some empirical evidence," *Accounting & Finance*, vol. 45, no. 3, pp. 439–456, 2005.

[2] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd, "Efficient transaction processing in sap hana database: the end of a column store myth," in *Proceedings of the 2012 ACM SIGMOD Int. Conf. on Management of Data*. ACM, 2012, pp. 731–742.

[3] J.-H. Boese, C. Tosun, C. Mathis, and F. Faerber, "Data management with saps in-memory computing engine," in *Proceedings of the 15th Int. Conf. on Extending Database Technology*. ACM, 2012, pp. 542–544.

[4] H. Plattner, "A common database approach for oltp and olap using an in-memory column database," in *Proceedings of the 2009 ACM SIGMOD Int. Conf. on Management of data*. ACM, 2009, pp. 1–2.

[5] N. May, A. Bohm, and W. Lehner, "Sap hana– the evolution of an in-memory dbms from pure olap processing towards mixed workloads," *Datenbanksysteme für Business, Technologie und Web*, 2017.

[6] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, "Sap hana database: data management for modern business applications," *ACM Sigmod Record*, vol. 40, no. 4, pp. 45–51, 2012.

[7] H. Plattner, "The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates," *Proc. of the VLDB Endowment*, vol. 7, no. 13, pp. 1722–1729, 2014.

[8] J. Hrastnik, R. Dentzer, and R. Colle, *Core Data Services for ABAP*. SAP PRESS, 2019.

[9] N. May, A. Böhm, M. Block, and W. Lehner, "Managed query processing within the sap hana database platform," *Datenbank-Spektrum*, vol. 15, no. 2, pp. 141–152, 2015.

[10] F. B. Alomari and D. A. Menascé, "Self-protecting and self-optimizing database systems: Implementation and experimental evaluation," in *Proc. of the 2013 ACM Cloud and Autonomic Comp. Conf.*, 2013, pp. 1–10.

[11] A. Boehm, "In-memory for the masses: enabling cost-efficient deployments of in-memory data management platforms for business applications," *Proc. of the VLDB Endowment*, vol. 12, no. 12, pp. 2273–75, 2019.

 ISBN: 978-1-61208-776-4

A Development Framework to Standardize Software Engineering Practices

Jishu Guin, Michele Macrì and Andrus Kuus

Software Engineering Research Group
Proekspert AS
Tallinn, Estonia

***Abstract*—The success of an engineering organization evidently depends on the growth of its engineers and the advancement of engineering. In order to realize these two broad factors, there must be inherent support for them at the organizational level. A well suited organization structure can, by virtue of its design, guide the engineering process to attain continuous growth of engineers and advancement of the field. This work explores the potential of matrix organizational structure combined with elements of Rational Unified Process (RUP) as a candidate to drive the success factors towards a desirable direction. The functional units of the matrix are mapped to the major phases of conventional software development process from requirement engineering to testing. The units or groups operate in compliance with the principles of RUP. The work at its current stage is a proposal of the framework and does not attempt to build a theory that can be verified empirically. However, empirical research methods have been considered as a way forward for future work. This paper, based on a preliminary analysis, attempts to show that the proposed structure not only can provide a platform for sustainable growth of engineers and advancement of engineering by incorporating standard engineering practices and methods in software development but also build a synergy to support more significant and challenging endeavors in future.**

Keywords–Matrix; RUP; Empirical; Software; Engineering; Practice.

I. INTRODUCTION

Professional education of engineers demands the acquisition of specialized knowledge as one of the key domains in addition to problem-solving skills and good judgment for the service of society. The nature of this knowledge has a broad spectrum, from fundamental to contextual [1]. Engineering profession provides a platform to apply and evolve this body of knowledge. In the decades of software engineering, various methods and practices have made their way into academia as part of the software engineering curriculum. These methods have proven their effectiveness and importance in industry. Application of these methods over the years built a synergy with the discipline of software quality leading to the development of various practices and tools to improve software engineering e.g., Rational DOORS. In addition to quality, the application of methods to various particular problems evolves the method to encompass larger and more complex scenarios. The goal of incorporating engineering practices in the development process is different from merely creating better software in terms of quality. Means to introduce standard methods and practices can level up the software quality by enriching the overall engineering discipline in the organization.

The work stems from a vision of improvement in the current organization of the authors. The software development workflow in the company is mainly directed towards product delivery. A typical project uses agile methods to ensure business demands are met. The focus of the development process is more on the delivery of features than the engineering methods. The assessment of quality delivered is usually based on information such as defects reported and customer feedback. The drawback of this approach is the suboptimal usage of the merits of engineering which comprises of its standard methods and metrics. The incorporation of industry-proven practices not only adds value to software development, but also towards the engineering culture and advancement. The primary aim of the work is to design a strategy to incorporate practices and methods as an integral part of the software development process to ameliorate the spheres of *culture*, *quality* and *advancement* as shown in Figure 1 to attain the following improvement.

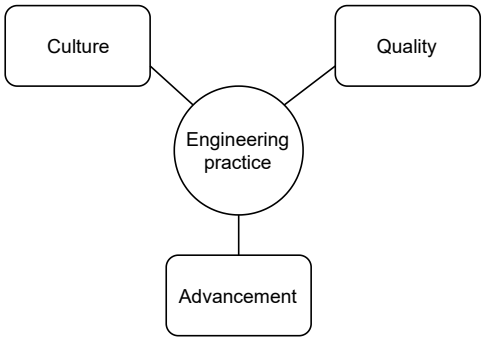

Figure 1. Vision of improvement

Culture - Familiarity of standard engineering concepts among developers and software engineers in order to improve technical communication and cooperation.
Quality - Enhance the quality of development by applying industry-proven methods with measurable outcomes.
Advancement - Assist automation of activities and produce reusable artifacts e.g., binary file, models, source code.

The words *method* and *practice* are both used in the work as means to improve the engineering process. Specifically, *method* refers to the standard technical solution applied in a systematic way to the development process. A solution is standard if it is used in the industry and more likely has a scientific foundation e.g., modeling of requirements. *Practice* refers to certain well recognized activities that can improve software development e.g., use of artifacts as prerequisite for each phase of the process. Further, the words method and

 ISBN: 978-1-61208-776-4

practice will be used interchangeably for brevity on account of their common goal.

The practices that are subject of this work are specifically tied to the software. They cater mainly to the engineering needs of the software and its development. The purpose of the work is not to enforce the use of a set of practices across all projects but to propose a foundational framework that can provide an environment to nurture application of standard practices leading to reuse of practices across projects. The re-usability aspect of the expected result can pave the way for a desired level of standardization.

The initial part of the work draws an overview of software engineering in an organization. This bird's eye view assists to identify certain areas that are associated with software engineering - *Standard Practices* being one of them. The study is significant to understand the co-relation of these areas with the idea of standardizing practices and the way certain factors are impeding the aim of this work. The following part of the work presents a framework to incorporate methods into the software development process. Two solutions are analyzed as part of this work and, based on the results, an attempt is made to reach a candidate solution. The work concludes by providing a guideline for selection of standard practices. The next section discusses the related work in the area of software engineering practices followed by an account of the software engineering domain from the organization's perspective. Section IV derives a strategy to incorporate standard practices and tabulates the guidelines for selection of practices. Finally, Section V presents the conclusion and directions for future work.

II. Related work

The specific nature of the work that embarks on an attempt to create a framework to incorporate engineering practices has limited the number of available related works. The idea of incorporating best engineering practices, however has been proposed by several Software Process Improvement (SPI) models. In Software Process Capability Maturity Model (CMM), the concept of benchmarking is used to accentuate the importance of methods and practices in software process [2]. The SPICE model of SPI distinguishes engineering activities as one of the key process categories [3]. The adoption of these standards remain a challenge for small organizations [4]. Generally, small companies are extremely responsive and flexible, because that is their advertised competitive advantage. Small companies don't have enough staff to develop functional specialties that would enable them to perform complex tasks secondary to their products. The business demand and lack of resource leads to the perception that SPI methods are expensive and time consuming [5]. This gives way to impediments in the optimal usage of standard engineering methods and practices. A number of works discussed in [4][5] propose to improve the software process in small companies. These models of improvement aim to attain certain level of maturity through a process of assessment. Despite their suitability to small companies the methods assume additional tasks that are secondary to the product. These tasks are not specifically focused to incorporate standard practices in the development process. The Technical Debt (TD) literature [6] identifies the lack of best practices as one of the ways to incur debt. Usage of good technical practices is a recommended way to prevent TD [7]. In reuse-oriented software engineering [8][9], the emphasis is on storing reusable knowledge in a repository to improve new developments in future. The model is focused on the storage and accessibility of the knowledge in the form of standard artifacts produced by development activities. Although both, TD and reuse oriented methods, rely on practices, they do not provide sustainable means to incorporate standard methods into development process. Application of standard practices is valuable for attaining aforementioned vision of improvement that this work aims to accomplish. The related works, despite their inclination towards standard practices, fall short in providing a directed strategy to incorporate practices. A targeted strategy to introduce practices is essential to retain the importance of standard methods in the face of business demands. The framework proposed in this work specifically aims to provide an ecosystem to apply and nurture practices. Instead of enforcing a secondary process, the framework makes standard practices an integral part of software development, thus, attaining a balance between business demands and engineering needs.

III. Software engineering domain

The model as shown in Figure 2 identifies *Standard practices* as an area associated with Software Engineering. The work aims to strengthen this area. Introduction of standard practices in the software development process necessarily causes a change in the current engineering domain in the organization. A study of the domain in the company provides insight into this correlation from two perspectives - The factors that impede the use of industry proven methods and the impact of standard practices on the domain. Certain key associations are identified to draw a picture of the engineering domain. The associated areas are elicited by considering the most fundamental connections in a software undertaking based on the conventional knowledge of the field. These associations embody complex relationship with each other. The study doesn't aim to provide a comprehensive exploration of these relations but attempts to draw certain key points that assists in stipulating the guidelines to model a solution. The areas depicted in the model, despite the complex relationship among each other, yields some useful information to model and assess the framework as discussed in the remaining part of the section.

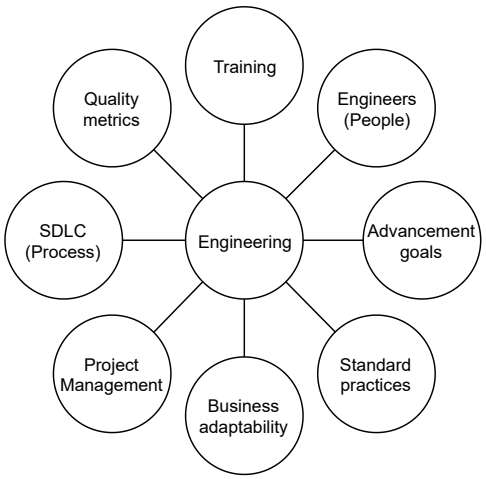

Figure 2. Software engineering domain model

A. Challenges from business

The current state of the development process uses agile methods, e.g., Scrum, that emphasizes on the value it produces for the product and customer. It is imperative to consider this value addition to business for successful operation of this industry. The areas of *Project Management* and *Business adaptability* ensure that the engineering process conforms to those needs. The former enriches business by producing deliverable that generate more value for resources invested and the later ensures that the process adapts to the expected standard of quality and constraints imposed by business. Some industries may not require the high standards of quality as demanded by safety-critical systems. Safety-critical projects bear the cost to ensure the quality demanded by the domain [10]. An implementation of industry-proven practices and methods contribute to the engineering needs of the project and it comes at the cost of time, expertise and budget for tools. The balance between customer and engineering needs of a project decides the balance between business and quality. A shift of focus on the business needs may lead to a counter-intuitive result of degradation in quality. The lack of a mechanism to secure this balance impedes the introduction of standard engineering practices in development.

B. Challenges from engineering

Software development life cycle lies at the core of software engineering process. Software Development Life Cycle (*SDLC*) incorporates main phases of development, from requirement analysis to testing. Standard practices aim to accomplish specific activities of these phases. In order to motivate the use of standard practice, the SDLC must encourage granularity of the activities that standard practices aim to accomplish. The blurred boundary between different engineering tasks compromises the granularity, leading to failure in accommodating standard practices. In contrast, an artifact based development as shown in Figure 3 can lead to a clear prerequisite and output for tasks. Despite the practical challenges of achieving clear boundaries between activities, an effort in that direction can lead to increased usage of standard practices.

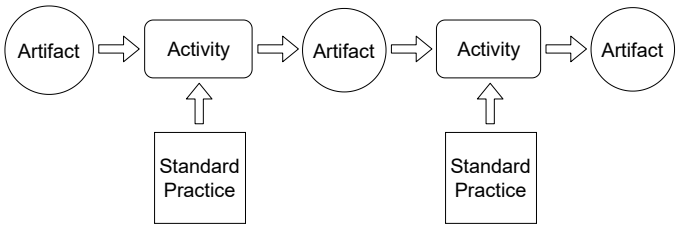

Figure 3. Artifact based workflow

C. Correlation with other areas

The areas of *Quality metrics*, *Training*, *Engineers* and *Advancement* do not directly resist the goal of this work but are influential to *Standard practice*. *Quality metrics* measures the attributes of process and product. The discipline of software quality being a driver for continuous improvement is an area where the organization has major scope for improvement. Standard engineering practices provide measurements that assist in acquiring data to measure quality. As more metrics are instilled in the process, it motivates increased usage of standard practices. The *Training* area in its current state in the company is mostly guided by choices of employees and not according to the demand posed by the development process. A set of standard practices can guide training and thus help to standardize the practices by imparting required knowledge to employees. The area of *Engineers* comprises the people aspect of engineering from hiring specialists to their growth in the company. Criteria for recruiting engineers greatly benefits from a set of practices required for the position. Training of these industry-proven methods expands the skill set of engineers with respect to overall software engineering discipline thus contributing to their growth. *Advancement goals* is a crucial dimension that drives engineering to embark on more significant endeavors e.g., Domain Engineering, Safety-critical systems. A successful realization of this vision necessitates a foundation that comprises standard practices as a key constituent. The next section explores two solutions and attempts to evaluate them based on this canonical domain model.

IV. Incorporation of standard practices

The challenge against standardizing practices is the resistance from aforementioned areas primarily from business and engineering. The aforesaid discussion on the challenges connotes the following two key points that a strategy to incorporate methods must take into account.

Motivation - Structure of the engineering unit must motivate focus on the engineering needs of the project to attain a balance with business aspects.
Sustainability - The process needs to provide a sustainable platform that demands standard industry practices e.g., use of artifacts as integral part of the development process.

The rest of this section describes a primary and alternate solution to the challenge of introducing standard practices followed by an analysis of the benefits and challenges of each solution. Based on the analysis a candidate strategy is drawn that reasonably attempts to address the challenges while retaining the benefits. The section concludes with a set of guidelines to assist the selection of practices.

A. Matrix structure

The balance between business and engineering is a key factor and demands to be maintained. Each of these two dimensions is necessary and significant part of the software engineering model. Empowerment of only one poses the risk of subverting the other. Standard practices contribute to the engineering dimension. This work proposes a framework inspired

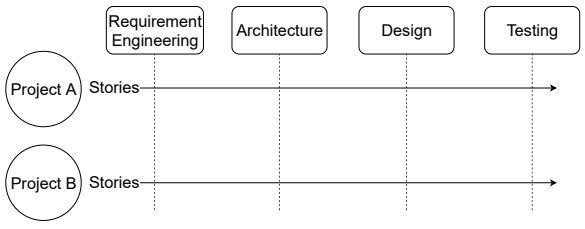

Figure 4. Framework based on matrix structure

by the Matrix organization structure [11] as a solution to attain a balance. The vertical dimension of this matrix represents engineering and the horizontal business. As shown in Figure 4 the functional units of the matrix are based on the phases of software engineering. The task of the design unit subsumes

the coding activities. There are standard design artifacts that can be used by tools to partially generate source code. Further improvement in design practices can maximize the code generated by tools. In this structure, instead of the project, functional units serve as the home base of engineers in it. The units focus on the application of engineering solutions. The member of a unit may work on tasks from multiple projects in context of the unit's engineering function. Implementation of stories is realized by the contribution of units in their relevant area.

The two dimensional structure brings forth the significance of engineering aspect by giving it the position of a body in the development process. This empowerment, although necessary to motivate the culture of engineering practices, is not sufficient to provide a steadfast platform for standard practices to operate and mature. This firm foundation can be established by using elements of Rational Unified Process (RUP) where artifacts are essential outcomes of activities. In RUP, artifacts are the tangible products of the project, the things the project produces or uses while working towards the final product [12]. In the matrix structure, the artifact produced by a standard method is a prerequisite for another as shown in Figure 3. This in effect makes artifacts form the operational interface between the functional units. The inclination towards an artifact based approach creates a sustainable demand for the use of standard practices. The possibility to incorporate standard practices into the development process, withstanding the opposing factors, by virtue of the design of the framework is the main contribution of this work.

The framework, by design, aims to meliorate the area of *Standard practices* but due to the presence of focused functional units the practices are also cultured and mature over time. In a matrix structure, when teams of functional specialists work together, a synergistic effect occurs, resulting in increased innovation and productive output, even though individually they may be working on different projects [11]. In addition to incorporating methods there are other notable benefits in the area of *Advancement goals* and *Engineers*. Re-usability is an attribute that is desired for the advancement of engineering and since the units serve multiple projects in context of a specific functional area there is a drive towards unifying the knowledge and attaining re-usability in the process. Growth of engineers comprises learning and a desired mobility in the organization. A functional unit being specific to a discipline in software engineering, i.e., design, architecture, requirement provides the opportunity to attain both learning and mobility to desired units for engineers.

The abstract framework proposed in this work pose certain potential risks that, although not established a priori or empirically, demand attention. The project domains in the organization have a wide spectrum from embedded application to mobile and web applications. The complexity of carrying out the activities of these diverse domains by one unit poses a challenge in the development and so does the overhead of making a change in the development model across all projects. The *Business adaptability* factor suggest that project may have different constraints and demands for the rigor of practices. Not all projects demand the rigor of formal methods because the cost of error may not be as high as it is for safety-critical system thus posing a challenge in standardizing the practice in a unit across all domains. The next subsection provides a brief account of a simple alternative.

B. Alternate solution

Addressing the challenges in matrix structure there can be alternative solutions based on incentives. In this model instead of the structure empowering the engineering aspect by design, engineers are motivated to use standard practices by incentivisation. The incentives can be in a form that contributes to their performance and growth. The application of a practice by the engineer can be evaluated by an independent body based on certain guidelines. The benefit of this structure is the lack of overhead to the current process and this can be applied across all domains. However, a major pitfall is that the key point of balance between business and engineering is not addressed and is left as a choice that the incentive may fail to influence in favor of engineering. In contrast to the matrix structure, the incentive solution does not directly provide the benefits of maturity of practices and re-usability due to lack of focused functional units.

C. Candidate solution

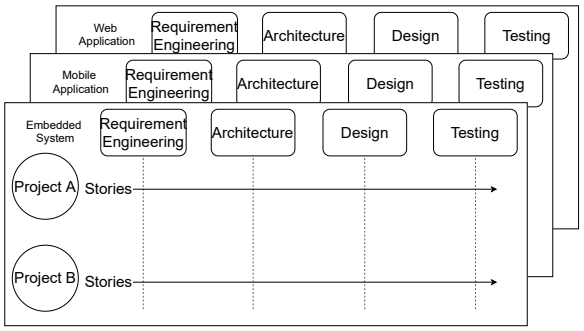

Figure 5. Domain based matrix structure

The directed and firm nature of the matrix structure towards addressing the *Motivation* and *Sustainability* aspects augmented by its contributions to *Advancement* and *Engineers* makes it a desirable solution. The aforementioned risks arising from *Project Management* and *Business adaptability* can be alleviated by applying the structure to individual domains instead of pursuing it across all projects. The domain based matrix structure retains the canonical form of the matrix but operates under the umbrella of a particular domain e.g., embedded systems, web applications. The simple modification as shown in Figure 5, apart from reducing the risk posed by the scale of change and multifariousness of domains, provides an additional advantage of domain specific re-usability that can lead to advancements like *Domain Engineering*.

D. Guidelines for selection of practices

The candidate solution aims to provide an environment to nurture standard practices. However, its effectuation demands a set of selected practices. The vast number of variables and choices available in different domains makes the selection of practices a challenging task. The engineering domain model presented in this work allows eliciting some guidelines that can help in the selection process. A comprehensive and systematic selection process would require consideration of various factors including quantitative valuation of the criteria, their precedence order and the procedural aspects of the process of selection. Such a study is not in the scope of this work.

 ISBN: 978-1-61208-776-4

TABLE I. PRACTICE SELECTION GUIDELINES

Association	*Criteria*
Quality metrics	Availability of desired quality metrics. The metric may correspond to a recognized software quality standard e.g., ISO/IEC 9126
SDLC	Produces artifacts that can be used by other practices. This can produce an artifact based workflow. Available tool support.
Training	Affordability and availability of resources for training
Engineers	Recruitment - Availability of experienced professionals with the competence required for the practice. Growth - Practice is recognized industry wide contributing to the growth of employees.
Business adaptability	Applicable directly or indirectly to wide range of business domains.
Project Management	Cost and turnaround time of the practice
Advancement goals	Assists the realization and maturity of concepts like automation and reuse that helps to lay the foundation for advancement of engineering

However, the points described in Table I can reasonably guide the selection process.

V. CONCLUSION AND FUTURE WORK

The current software engineering model in the organization successfully caters to the business needs in the software development process. However in order to sustain high level of quality irrespective of the size and complexity of the project, the model must heed the engineering aspect. Introduction of a set of standard engineering practices is an important step in that direction. This work attempts to lay a foundation to build the set of practices and proposes a framework to incorporate them in the software development process.

The premise of the work is a software engineering domain model that provides a landscape to elicit the set of guidelines for selection of standard practices and derive two solutions. Based on the benefits and drawbacks of each, one of the solutions is modified to reach a candidate for further empirical evaluation. The analysis and elimination process used to derive the solution strengthens the logical soundness of the approach. The solution proposed is a framework based on the matrix organization structure applied to specific domain. The framework notably empowers the engineering aspect of the development process thus providing a platform to apply and standardize engineering practices. The framework in its current state is a proposal and lacks the rigor of a theory. Thus, it does not produce a comprehensive set of testable hypothesis at this stage.

The work gives rise to two distinct lines of research to pursue in future. Firstly, the theory building process must be applied to the proposed framework, which includes the delineation of the term standard practice [13]. The formulated theory will provide the foundation for practical evaluation. Secondly, an empirical research strategy needs to be designed and conducted to test the hypothesis drawn from the formulated theory [14]. A test can provide the necessary experimental data required to establish the validity of the proposed framework. The scope of the preliminary empirical evaluation shall be confined to a single domain e.g., embedded system.

REFERENCES

[1] S. Sheppard, A. Colby, K. Macatangay, and W. Sullivan, "What is engineering practice?" International Journal of Engineering Education, vol. 22, no. 3, 01 2006, pp. 429–438.

[2] W. Humphrey, "Introduction to software process improvement," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-92-TR-007, 1992. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11611 [retrieved: 01,2020]

[3] Tr, "Information technology — software process assessment — part 2 : A reference model for processes and process capability." ISO, 1998.

[4] G. Valdés, M. Visconti, and H. Astudillo, "The tutelkan reference process: A reusable process model for enabling spi in small settings," in Systems, Software and Service Process Improvement, R. V. O'Connor, J. Pries-Heje, and R. Messnarz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 179–190.

[5] I. Richardson and C. G. Von Wangenheim, "Guest editors' introduction: Why are small software organizations different?" IEEE Software, vol. 24, no. 1, Jan 2007, pp. 18–22.

[6] E. Allman, "Managing technical debt," Queue, vol. 10, no. 3, Mar. 2012, p. 10–17. [Online]. Available: https://doi.org/10.1145/2168796.2168798 [retrieved: 01,2020]

[7] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, 1st ed. Addison-Wesley Professional, 2012.

[8] E. Ras, J. Rech, and B. Decker, "Workplace learning in software engineering reuse," in Proc. Int. Conf. Knowledge Management, Special Track: Integrating Working and Learning, 2006, pp. 437–445.

[9] M. T. Baldassarre, A. Bianchi, D. Caivano, and G. Visaggio, "An industrial case study on reuse oriented development," in 21st IEEE International Conference on Software Maintenance (ICSM'05). IEEE, 2005, pp. 283–292.

[10] D. Turk, R. France, and B. Rumpe, "Limitations of agile software processes," In Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002), 05 2002, pp. 43–46.

[11] L. C. Stuckenbruck, "The matrix organization." Project Management Quarterly, vol. 10, no. 3, 1979, pp. 21–23.

[12] R. U. Process, "Best practices for software development teams," A Rational Software Corporation White Paper. TP026B, Rev, vol. 11, no. 01, 2001.

[13] D. I. Sjøberg, T. Dybå, B. C. Anda, and J. E. Hannay, "Building theories in software engineering," in Guide to advanced empirical software engineering. Springer, 2008, pp. 312–336.

[14] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in Guide to advanced empirical software engineering. Springer, 2008, pp. 285–311.

 ISBN: 978-1-61208-776-4

Broadening the Lens:

Toward the Collective Empathic Understanding of Product Requirements

Robert C. Fuller
Electrical and Computer Engineering
The University of British Columbia
Vancouver, Canada
e-mail: rfuller@ece.ubc.ca

***Abstract*— Many software product companies have embraced the agile paradigm and gone on to create cross-functional product development teams that fully own their product. The expectations of these teams are very different than of development teams in a disciplined software development environment. The study underway examines how these empowered cross-functional product teams, as a collective, create and nurture a shared mental model that accurately represents the external product domain and its realities and that provides the context for understanding the requirements. We also examine external factors that allow for these teams to develop these capabilities while less-empowered teams cannot. Using Constructivist Grounded Theory, we study individuals and teams in several companies and varied product domains. We find that certain organisational factors play a significant role and we also examine an essential dynamic of broadening the lens and blurring boundaries that cross-functional product teams employ in order to not only fully embrace product planning but also to grok the domain for their products.**

Keywords - empathy-driven development; collective sensemaking; design science; requirements validation; product team organisation.

I. INTRODUCTION

Product development is a social process; thus, the dimension of the organisational model and dynamics is the elephant in the room, a critical factor for success or failure of software products.

This study builds our earlier work [1] that studied software product teams that displayed varying degrees of collective grokking. In that study, we found that the organisational model surrounding the teams had a profound influence on whether the teams could grok the product requirements at all. Building upon that work, we use the Constructivist Grounded Theory method (described further in Section V) to examine characteristics of collective team grokking of the product domain and we also examine how the extra-team organisational model affects the team's ability to own increasingly comprehensive product planning.

We use the concept of broadening the lens as an explanatory mechanism that Cross-functional Product Teams (CFPTs) use to explore further and innovate more and we also look at some of the prerequisite conditions in order for teams to do this.

Grokking is cognitive empathy, coupled with skilled perspective-taking. We use a definition of cognitive empathy to be "*the ability to imaginatively step into another domain, understand the perspectives of those in that domain, and use that understanding to guide decisions"* [2,x]. Increasingly, the success of software product development teams depends on the degree to which the team *collectively* groks not only the product requirements themselves but also, and importantly, the *context* for those requirements.

The remainder of this paper is structured as follows: Section II – Background and Problem provides an overview of the historical background and description of the research problem. Section III – Research Motivation and Focus describes what we're aiming to achieve and a brief description of the research scope. Section IV – Related Work positions this study with respect to three related areas of research. Section V – Method and Status of the Research overview the research methodology chosen and current status respectively. Section VI – Emerging Observations and Discussion describes the findings to-date followed by Section VII – Conclusion and Future Work, offering thoughts about contribution thus far and what work remains to be done.

II. BACKGROUND AND PROBLEM

By the late 1990s, three forces had taken hold which dramatically changed the nature and challenge of software development. One was the emergence of the Internet which introduced new uses of information technology as well as business models. This, combined with much lower hardware costs, computing capability rapidly appeared on almost every desk and in almost every home. Third, the introduction of graphical user interfaces dramatically enriched user interaction with technology and also complicated software design and development. These three forces together resulted in more software being developed as *products* for a market instead of predominantly bespoke system development that was the norm prior. This shift towards product development introduced substantially more uncertainty into much of the software development activities.

 ISBN: 978-1-61208-776-4

In response to this, a Kuhnian "model revolution" [3] emerged that took a new view on change, risk, and uncertainty in software development. This 'agile' approach accepted that requirements could change or that further understanding would emerge throughout the development effort in contrast to more disciplined Software Development LifeCycles that strived to lock down requirements in the specification and planning stages.

The agile model placed greater focus on the development team, recognizing that prescriptive processes were insufficient to ensure project success in these complex and emergent conditions and that the dynamics of the development team, which was now usually cross-functional and empowered to truly own the software product, was considered a critical success factor in delivering software.

While the agile approaches improve many of the issues that were breaking down during the crisis period, many still cling to the notion that there is a customer (or, an internal surrogate), an authoritative voice that the development team can iteratively interact with to clarify requirements and validate results. However, as software solutions address more complex and subtle needs and as development is often more product-oriented, intended for a market rather than a single customer, a new and critical challenge emerges for software teams and that is how to gain a deep understanding of the world for which the product is intended, an understanding that cannot be passed on to the team by an internal market surrogate. Certainly, techniques to 'hear' from the market are helpful but, as Polyani [4] noted, market participants have tacit knowledge -- people can know more than they can tell and they know more than can be easily observed.

In early times when requirements were less complex, could be more precisely expressed, and quite often coming from an identifiable customer, techniques such as having at least one domain expert on (or available to) the team were often sufficient. Today, however, with much more technical and problem complexity, heterogenous customer targets, and competitive uncertainties, it is insufficient to simply have one person with this deep understanding, typically creating the requirements specification. Yet, many software development organisations operate this way, often resulting in *requirements fixation* [5].

Rather it is important that everyone on the team have a deep domain understanding. It is also critical that the entire team understands it in a compatible and consistent way because team members (individually, in sub-teams, and across all functional roles) make decisions almost continually based on their individual understanding of the context of the requirements, and much of that context understanding is tacit. This challenged is expressed well by Berry [6] when discussing assumptions in requirements engineering amongst team experts:

"It seems that among experts, a common disease is the presence of unstated assumptions. Because they are unstated, no one seems to notice them. Worse than that, it seems that no two people have the same set of assumptions, often differing by subtle nuances that are even more tacit than the tacit assumptions. It is these assumptions that confound attempts to arrive at consensus, particularly because none of the players is even consciously aware of his or her own assumptions and certainly not of the differences between the players' assumptions" (p.180)

Thus, product development teams have to strive for a deep *collective* understanding of the context of their product, a shared mental model of the supra-domain, since many decisions are unconsciously made within the team's understanding of the domain context. Some teams achieve success in this aspect more than others and software development leaders have no theories that help explain why.

We observed earlier [1] that the organisational model surrounding the cross-functional teams has an impact on the team's ability to grok, hence the scope of this inquiry expands from there to examine additional factors both internal and external to the teams.

III. Research Motivation and Focus

This study aims to develop theory offering insights into factors that support or impede CFPTs in collectively achieving a deep understanding of the context of their products.

The differences between teams that achieve a reasonable degree of collective grokking in terms of team vision, cohesion, and quality of work product is observable by practitioners and researchers, yet the reasons are generally unclear. Without explanatory models, industry leaders are unable to proactively create and nurture the relevant factors. This study is aimed at helping industry practitioners explain why certain prevailing techniques and empirical approaches for understanding software solution needs are often inadequate, why some succeed while others do not.

The focus of this research is practicing software product teams in action, including teams empowered to own their product and those that are not. For contrast, we also include organisations that are not product companies. The study examines the empirical adaptations these teams make toward furthering their understanding of the context in which their users operate. We also examine important organisational factors that either allow or inhibit a team's ability to collectively grok the domain.

IV. Related Work

We reviewed published material in 3 areas - requirements engineering, design science, and collective sensemaking.

This inquiry is primarily related to requirements engineering (attempting to obtain and understand the true needs). Reviewing all the papers at the IEEE International Requirements Engineering Conference over the past decade, plus many other published papers in the area, we found growing sentiments expressed about the shortcomings of prevailing approaches to requirements engineering which

 ISBN: 978-1-61208-776-4

tend to focus on techniques and methods rather than deepening practitioner and team understanding, e.g., (Schon et al. [7], Ralph and Mohanani [8]). This general sentiment led to the formation of the NaPiRE initiative (Naming the Pain in Requirements Engineering) [9], a community endeavour run by a multitude of researchers world-wide. While there are certain domains where the 'techniques and methods' approach is entirely adequate and appropriate, our focus is on problem domains that do not lend themselves well to complete and unambiguous specifications and, therefore, where it is necessary for the CFPTs to have their own deep understanding of the product domain beyond just the requirements specifications.

The design science space has considerable material regarding empathy-driven design (translating human needs to experiences), e.g., (Koppen and Meinel [10], van Rijn et al. [11], Postma et al. [12], Woodcock et al. [13], Dong et al. [14], Kourprie and Visser [15], Kolko [16]). However, we found this falls short of addressing our inquiry question in three critical respects: 1) focus on the design activity as part of an essentially sequential product development process rather than design as part of an on-going continuous product development effort, 2) it tends to focus on the design individual or only the design team rather than the whole development team and, 3) when even the design team is considered, it tends not to be viewed as a *unit* regarding its empathic ability. Design science models described by Wieringa [17] acknowledge the challenge that empathy-driven requirements understanding attempts to address but stops short of suggesting how those challenges could be addressed. We aim to offer insights into how this level of understanding is achieved and how to nurture the pre-requisite conditions.

Collective sensemaking (the process by which people give meaning to their collective experiences) does consider the collective (team) but only with respect to its relationship to the organisation, not to its understanding of an external domain. Of interest in this area is the Cynefin framework (Kurtz and Snowden, [18]) which is a sensemaking framework that is designed to allow shared understandings to emerge which could be insightful with respect to how teams ingest, socialise, and collectively store insights. As with other collective sensemaking models, however, it has resonance in early problem-solving stages and for formal and finite periods of time whereas our focus is on the full product lifecycle.

V. METHOD AND STATUS OF THE RESEARCH

We take an interpretive epistemological stance, employing the Constructivist Grounded Theory qualitative research methodology described by Charmaz [19]. Constructivist Grounded Theory is highly applicable in research such as this because the method is explicitly emergent, taking an inductive approach where no adequate prior theory exists. This method is particularly appropriate for a "What is going on here?" type of qualitative inquiry as this study is. The use of Grounded Theory in computer science research has risen significantly since 2005 and specifically used successfully to study Agile software development teams, e.g. Adolph et al., [20], Dagenais et al., [21], Coleman and O'Connor, [22], Martin, [23], Hoda, [24], Stol et al, [25].

Using theoretical sampling where the analysis of the data collected prior informs the selection of and inquiry with the next participants, individual participants and corporate sites selected are ones involved with software product development (teams developing software for market) and that claim to have cross-functional product development teams. The primary data collection methods are observations of team meetings and team interactions, enriched by semi-structured interviews (recorded and transcribed) with open-ended questions that can allow real issues to emerge. Thus, the method is grounded in the participants' world and the emerging and evolving theory is constructed by the researcher and the participants.

We employ various strategies (Maxwell, [26]) to mitigate threats to validity (credibility, dependability, reliability). Intensive, on-going involvement, e.g., extended participation and the ability to live in the participants' workplace, provides richer data and data types, less dependence on inference, and opportunity for repeated observations and interviews, all which will help rule out spurious associations and premature theories. Participant checks (obtaining participant and peer feedback on the data collected and conclusions drawn) help rule out possibilities of misinterpretation. Select codes and concepts from the analysis are highlighted below as ***bold italics***.

To date, we are working with six software firms. Four of these firms produce commercial enterprise-class software products, one creates sophisticated virtualisation solutions, while another develops large-scale aerospace systems as bespoke system development. Three of these firms have adopted agile as a paradigm, two as a methodological approach, while the other employs a highly prescriptive methodology due to the dictates of its market. The firms range in size from ten to several hundred employees and the firms range in age from 2 to 50 years old. To-date, 18 product development teams across these companies have participated, resulting in 26 individual semi-structured interviews and 19 team observation sessions. The individuals interviewed have been 2 senior managers, 8 senior engineer / team leads, 5 product managers, 1 quality assurance specialist, and 10 intermediate-level software engineers. Participant sampling and data gathering is on-going.

VI. EMERGING OBSERVATIONS AND DISCUSSION

We have identified three contexts that contribute to a CFPT's ability to collectively grok the product requirements. The first is the organisational context which we identified in isolation in our earlier work [1]. The second is the product planning context (the ability of CFPTs to own

 ISBN: 978-1-61208-776-4

a broader responsibility for product planning), influenced by the organisational context but having its own independent dynamics. The final context is the product domain context itself (the ability for the CFPT to grok the product domain), heavily dependent on the previous two contexts.

A. The Organisational Context and Its Impact on CFPTs

Fuller [1] described the impact the broader organisational model has on the CFPT team. Impacts of note were intra-team deference, the team's concern and ownership horizon, and the team robustness. Certain aspects of individual participation on the team (e.g. primary affiliation, individual agenda) were also highlighted. In this sub-section, we summarise those findings.

When a functional organisational structure exists in the software product enterprise, e.g., separate engineering, design, product management departments, each contributing individuals to form CFPTs, team members are more likely to limit their contributions to topics directly relating to their area of functional expertise and tend to show marked deference to team members of other functions on topics outside their area of primary functional expertise. The individual sense of primary affiliation was stronger toward their functional department than it was with the software product team. Simply put, an individual in this organisational environment is located via function more than via team membership. This results in team members being much more concerned about *how* a product is to be built and defer to others regarding the *what* and *why*. Illustrative comments from team members were "***I just do what I'm asked to do***" when referring to involvement with requirements specifications or "***They're the experts, I trust them***" when referring to team members in other functional roles.

Team members in this model tended to show less investment in the overall success or failure of the product and the teams themselves much less likely to take collective responsibility for success or failure of the product. They are more likely to shift responsibility to management decisions or to other teams/functions rather than attempt to reconcile differing mandates of the participating functional departments.

In contrast, organisations without a functional structure surrounding the CFPTs seem more likely to have teams with richer intra-team interactions with softer (sometimes an absence of) functional interfaces amongst individual team members, placing the interests of the product foremost and above any functional tensions. In short, the sense of team and commitment to the product tended to be much stronger. In one of our participant companies with multiple products, it is common for product team members' LinkedIn profiles showing the product name as the company they work for with no reference to the overall firm, making it very clear where they belong and what they are committed to.

Studies by Gladstein [27] and Anacona [28] noted that contextual factors have a greater influence on team effectiveness than do internal team processes. Our emerging results to-date support this and suggest further that the two are not unrelated – that the operating context of the team has a significant impact on internal team factors, which include internal team processes.

In summary, a CFPT's progression along a spectrum from an assembly of experts to a true empowered cohesive team is heavily influenced by whether a broader functional department organisational structure exists around the team and how strong those departmental distinctions are.

B. The Product Planning Context and CFPTs

We observed that CFPTs that have strong internal connections and softer functional role deference showed more interest in the broader product planning context. These teams ask broader questions, are more curious, and attempt to explore more - essential ingredients for innovation.

However, our observations also included teams in some companies that did not have the organisational structure and/or culture that allowed their CFPTs to own as much of the product planning process that the teams often wished they could own. This was often the case where strategic planning for product occurred in another functional area and communicated to the product development group to execute upon. Some companies will take this even further and have a separate product management function that define product evolution details that are then handed off to software engineering for development. We observed that CFPTs with strong internal cohesion have a propensity to own *something* and will, therefore, ***narrow* or *broaden their lens*** on the product development work to match what they are permitted to own. This action of ***Broadening the Lens*** allows the team to identify control boundaries and also to see patterns and relationships so that they may more purposefully and knowledgeably re-focus.

This lens adjustment also aligns their definition of success with what the company expects. Individuals and teams will ***colour within the lines*** they are given or allowed. This is reflected in what completed work the development teams celebrate, e.g. a successful iteration, meeting a release deadline, or being part of a successful product in the market.

The spectrum of this context ranges from full strategic and execution ownership of the product on one end to the team being ***spoon-fed*** tasks on the other.

As with the Organisational Context, the focus of a team's product planning lens also shows in the verbal language used by the teams. The broader the team's planning scope is, the more the conversations will indicate a deep understanding of (or, at least references to) product needs from the domain perspective, product/market opportunities, etc. Teams low on this spectrum reference those considerations less and make more reference to internal entities and artefacts such as other functions/teams, processes, specifications, etc.

 ISBN: 978-1-61208-776-4

C. The Product Domain Context and CFPTs

As Fuller [1] observed, empowered and cohesive CFPTs play a longer game. With less internal deference in the team and less individual tentativeness with respect to their membership on the team, conditions exists that encourage full participation and commitment (both individual and collective) to the long-term product roadmap.

In our analysis to-date, we find that CFPTs that exhibit little to no functional deference across functions within the team and who are not being ***spoon-fed*** their development tasks almost always exhibit *some* degree of collective grokking of the context of their product domain and, hence, the product requirements.

This is significant because all software is developed in context and it is context that guides decisions. If the team is cohesive, their context will be more collective than if it is not (Organisational Context). If the team owns more of the product planning, that context will be more comprehensive than if they own less (Product Planning Context). And if they collectively grok the product domain to a reasonable degree, their context will more accurately reflect the world for which their product is intended (Product Domain Context).

The spectrum of collective domain understanding ranges from ***just do what the story says*** to intellectual domain understanding (deep knowledge of vocabulary, workflows, objectives, etc.) to true felt (lived) understanding of the domain. The further a team moves along this spectrum, the more the team groks - blurs the boundaries between itself and the domain in order to achieve some degree of empathic understanding.

In this context of requirements engineering, we suggest that empathy, specifically collective cognitive empathy, is a fundamentally important ability in order to deeply understand a domain which the team is otherwise unfamiliar with. Exercising that ability, stepping into that other domain, involves a certain temporary softening of the distinction between the collective and the domain, a ***blurring of the boundaries***, in order to truly understand perspectives in that domain. ***Broadening the lens*** is necessary for the team to be able to see the other domain and its context, the ***blurring of the boundaries*** is an effort to understand. Smith et al. [29] suggest that empathy can become collective and that it can be an attribute of the group that is more than just the aggregation of individuals' attributes.

We observed some teams that did not even attempt to grok the product domain, a reflection of the culture of the team and its organisational environment. Certain other teams that did try had modest success due to influences from the organisational and/or product planning contexts.

For a CFPT to be able to *collectively* step into another domain, it is necessary for it to see itself as a cohesive unit. This can only be achieved when there is a high level of transparency across all functions on the team, little to no deference shown within the team, and a strong sense of collective ownership for the product. In other words, a true *team* with a strong product mandate – blurred boundaries with strong connections. It requires team members to feel psychologically safe, have open minds, and a strong sense of curiosity. If any of these are weak or missing, discoveries, innovation, and collective grokking are inhibited [30].

As we examined the teams that made some progress at collective grokking of the product domain, we observed a special form of the ***broadening the lens*** behaviour that teams performed when refining their product planning context. In these cases, the teams were *purposefully* ***blurring boundaries*** in order to achieve a deeper collective empathic understanding of the product domain.

VII. Conclusion and Future Work

Many of these observations sit in opposition to common organisational practices that emphasise specialisation (for management and control convenience) and focus (to meet deadlines). Further work is needed to bring more clarity about whether there are other, more subtle, factors at play.

Our data strongly indicates that ***blurred boundaries*** within CFPTs are a reflection of blurred boundaries outside of the teams and, similarly, there may well be even further team environmental factors to explore.

There appears to be a certain blurring of functional and domain boundaries necessary for a team to become a true product team rather than a collection of functional experts assembled around a product. Further, this appears to be a pre-condition for the team to be able to behave as a collective and achieve some degree of collective grokking of the context of the product requirements.

While we observed teams using the ***broadening the lens*** mechanism to ***blur the boundaries*** between the team and the domain, we allow that this mechanism and the pre-requisite or enabling conditions may paint only a partial picture. Thus, we believe there remains much to explore with respect to why some teams, even in the same organisational context, observably achieve more grokking of the product domain than do other teams.

Acknowledgment

This work is supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

References

[1] R. Fuller, "What T-shirt Are You Wearing? Towards the Collective Team Grokking of Product Requirements," in *SOFTENG 2019, The Fifth International Conference on Advances and Trends in Software Engineering*, pp. 37–40, 2019.

[2] R. Krznaric, *Empathy: why it matters, and how to get it*. New York: Penguin Random House, 2014.

[3] T. S. Kuhn, *The Structure of Scientific Revolutions*. 4th ed. University of Chicago Press, 2012.

[4] M. Polanyi, *The tacit dimension*. Chicago: University of Chicago Press, 2009.

 ISBN: 978-1-61208-776-4

[5] R. Mohanani, P. Ralph, and B. Shreeve, "Requirements Fixation," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 895–906, 2014.

[6] D. M. Berry, "The importance of ignorance in requirements engineering," *Journal of System Software*, vol. 28, no. 2, pp. 179–184, 1995.

[7] E. M. Schön, D. Winter, M. J. Escalona, and J. Thomaschewski, "Key challenges in agile requirements engineering," in *Lecture Notes in Business Information Processing*, 2017.

[8] P. Ralph and R. Mohanani, "Is Requirements Engineering Inherently Counterproductive?," in *Proceedings - 5th International Workshop on the Twin Peaks of Requirements and Architecture, TwinPeaks 2015*, 2015.

[9] D. Mendez, S. Wagner, M. Kalinowski, M. Felderer et al.. NaPiRE: Naming the Pain in Requirements Engineering, http://napire.org.

[10] E. Koppen and C. Meinel, "Knowing People: The Empathetic Designer," *Design Philosophy Papers*, vol. 10, no. 1, pp. 35-51, 2012.

[11] H. Van Rijn, F. S. Visser, P. J. Stappers, and A. D. Özakar, "Achieving empathy with users: the effects of different sources of information," *CoDesign*, vol. 7, pp. 65–77, 2011.

[12] C. Postma, E. Zwartkruis-Pelgrim, E. Daemen, and J. Du, "Challenges of Doing Empathic Design: Experiences from Industry," *Int. J. Des*. Vol 6, No 1, pp. 59-70, 2012.

[13] A. Woodcock, D. McDonagh, J. Osmond, and W. Scott, "Empathy, Design and Human Factors," *Advances in Usability and User Experience*, pp. 569-579, 2018.

[14] Y. Dong, H. Dong, and S. Yuan, "Empathy in Design: A Historical and Cross-Disciplinary Perspective," *Advances in Neuroergonomics and Cognitive Engineering*, pp. 295-304, 2018.

[15] M. Kouprie and F. S. Visser, "A framework for empathy in design: stepping into and out of the user's life," *J. Eng. Des.*, vol. 20, no. 5, pp. 437–448, 2009.

[16] J. Kolko, *Well-Designed: How to create empathy to create products people love*. Harvard Business Review Press, 2014.

[17] R. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer, Berlin, 2014.

[18] C. F. Kurtz and D. Snowden, "The New Dynamics of Strategy: Sense-making in a Complex-Complicated World," *IBM Syst. J.*, vol. 42, no. 3, pp. 462–483, 2003.

[19] K. Charmaz, *Constructing grounded theory* (2nd ed.). London: Sage, 2014.

[20] S. Adolph, W. Hall, and P. Kruchten, "Using grounded theory to study the experience of software development," *Empirical Software Engineering.*, vol. 16, no. 4, pp. 487–513, 2011.

[21] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. De Vries, "Moving into a New Software Project Landscape," in *ICSE '10 Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pp. 275–284, 2010.

[22] G. Coleman and R. O'Connor, "Using grounded theory to understand software process improvement: A study of Irish software product companies," *Information Software Technology*, vol. 49, no. 6, pp. 654–667, 2007.

[23] A. M. Martin, "The Role of Customers in Extreme Programming Projects," PhD thesis. Victoria University of Wellington, New Zealand, 2009.

[24] R. Hoda, "Self-Organizing Agile Teams : A Grounded Theory," PhD thesis. Victoria University of Wellington, New Zealand, 2011.

[25] K. J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proceedings – International Conference on Software Engineering*, vol 14-22. pp. 120-131, 2016.

[26] J. A. Maxwell, *Qualitative research design: An interactive approach*. Thousand Oaks, Calif.: SAGE Publications, 2012.

[27] D. L. Gladstein, "Groups in Context: A Model of Task Group Effectiveness," *Adm. Sci. Q.*, vol. 29, no. 4, p. 499, Apr. 2006.

[28] D. G. Ancona and D. F. Caldwell, "Demography and Design: Predictors of New Product Team Performance," *Organ. Sci.*, vol. 3, no. 3, pp. 321–341, Oct. 2008.

[29] E. R. Smith, C. R. Seger, and D. M. Mackie, "Can Emotions Be Truly Group Level? Evidence Regarding Four Conceptual Criteria,". *J. Pers. Soc. Psychol.*, 2007.

[30] M. Harms and R. Reiter-Palmon. *Team Creativity and Innovation*, Oxford: Oxford University Press, 2018.

 ISBN: 978-1-61208-776-4

Test Coordination and Dynamic Test Oracles for Testing Concurrent Systems

Bernard Stepien, Liam Peyton
School of Engineering and Computer Science
University of Ottawa
Ottawa, Canada
Email: {bstepien | lpeyton}@uottawa.ca

***Abstract*—Testing concurrent systems is complex. In traditional software unit testing, a test sequence is always composed of a stimulus and its corresponding fully predictable response. With concurrent systems, this simple model no longer holds as the state of the System Under Test (SUT) changes while several users place their requests. Race conditions are a particularly challenging problem for testing, since they will occur and must be identified, but are very disruptive to the test environment. In this paper, a case study, using the formal test specification language TTCN-3, illustrates the challenges for test coordination, especially race conditions, and propose techniques to address them. We also introduce shared variables and the use of semaphores in the TTCN-3 parallel test component model as a mechanism to implement dynamic test oracles.**

Keywords- software testing-concurrent systems; TTCN; test oracles; race conditions.

I. INTRODUCTION

Testing concurrent systems is complex. In traditional software unit testing, a test sequence is always composed of a stimulus and its corresponding fully predictable response [4]. With concurrent systems, this simple model no longer holds as the state of the system under test (SUT) changes while several users place their requests. Race conditions are a particularly challenging problem for testing, since they will occur and must be identified, but are very disruptive to the test environment.

Some definitions and implementations of parallel testing can be found in [6][7][8][9]. Obviously there are different kinds of parallel testing. In the previous reference, the main concern is to run sequential tests in parallel in order to save time. Instead, we focus on concurrent testing of states in a system under test (SUT) states as the test purpose. There are two main categories of concurrent test systems:

- Response time testing when a large number of requests are sent to a server as shown in Figure 1. This is addressed using TTCN-3 in [10].
- Testing the actual processing logic of the SUT when confronted by several requests from parallel users where the state of the SUT is changing as a result of requests of the users and thus affecting each user's behavior.

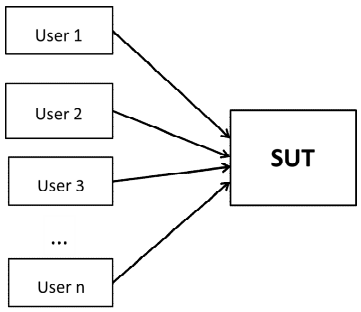

Figure 1. Parallel system configuration

In this paper, a case study, using the formal test specification language TTCN-3, illustrates the challenges for test coordination, especially race conditions, and proposes techniques to address them. We also propose shared variables and semaphores in the TTCN-3 parallel test component model as a mechanism to implement dynamic test oracles. Overall, the motivation to use a formal method such as TTCN-3 and its related available execution tools is to take full advantage of its logging information in order to rapidly detect faults due to race conditions. We also propose enhancements to the TTCN-3 language to make our testing concurrency problem statement usable.

II. A CASE STUDY

In sub-section A we define the dynamic state problem to be addressed, in sub-section B we propose thee methods to specify concurrent systems tests.

A. Defining the problem

Although, we have studied extensively testing concurrency problem in industrial applications [11], the following simplified case study is about testing the transition of the state of a system and the kind of responses it should reply with. Here we have parallel users that send a request to a book ordering system and get two kinds of replies depending on the two possible states of the SUT: has stock; or out of stock. The problem is that it is impossible to predict the test oracle (predicted response) since each user is independent from each other and thus does not know the state of the SUT. This is similar in e-commerce applications like on-line ordering of merchandise and hotel booking and train or airline reservations systems. A typical warning message for a hotel reservation system is to warn the customer that there

 ISBN: 978-1-61208-776-4

is only one room left at a given rate. Thus from a tester point of view, it is hard to predict if a response corresponds to a success or a failure. However, if the users are coordinated, the response to a given user can be predictable.

The interesting aspect of this simple example is that we have tried various approaches of coordination and some resulted in race conditions problems, thus disturbing the test process altogether. Table I shows the values of test oracles depending of the state of the SUT, in our case: has stock; or out of stock. In short a test passes if an invoice and shipping confirmation is received when there is inventory left or when out-of-stock is received and the server is out of stock. All other cases are failures.

Unit testing would consists in putting the SUT in the appropriate state and check the individual responses.

What is missing from a unit test is the dynamic aspect of seeing the state change as the maximum available inventory is reached.

TABLE I. EXPECTED TEST ORACLES DEPENDING ON THE STATE OF THE SUT

Response to the User/state	Has stock	Out of stock
Invoice	pass	fail
Out of stock	fail	pass

B. TTCN-3 implementation

The TTCN-3 implementation of the user parallel test component (PTC) is based on a simple request/response behavior pattern with the response being analyzed with the four possible configurations of two states and two corresponding responses making use of the TTCN-3 *alt* (alternative) construct. Each alternative is guarded with the predicted state of the SUT. The receive statement contains what the received message from the SUT should match and the predicate between square brackets, the predicted state of the SUT.

```
function ptcBehavior() runs on PTCType
{
  p.send("purchase");

  alt {
     [state == "has_stock"]
        p.receive("invoice") {
            setverdict(pass);
        }
      [state == "out_of_stock"]
        p.receive("invoice") {
            setverdict(fail);
        }
      [state == "out_of_stock"]
         p.receive("out_of_stock") {
            setverdict(pass);
         }
      [state == "has_stock"]
         p.receive("out_of_stock") {
            setverdict(fail);
         }
  };
}
```

Figure 2. PTC Client test verdicts situations

Instead, unit testing would break down the problem into two separate test cases and especially without the need for PTCs. Here the unit is represented by a given state.

First unit test case:

```
function unitTestBehavior_1() runs on
                              MTCType {
  p.send("purchase");

  alt {
     [] p.receive("invoice") {
            setverdict(pass);
        }
     [] p.receive("invoice") {
            setverdict(fail);
        }
```

Second unit test case:

```
     [] p.receive("out_of_stock") {
            setverdict(pass);
     }
     [] p.receive("out_of_stock") {
            setverdict(fail);
     }
   }
```

The predicates are empty because the state is predictable due to the manipulation of the SUT by the tester by emptying the data base in the first case and populating the database in the second case. Another drawback of unit testing is that the testing process would not be entirely automated since it requires a manual intervention of the tester between the two states.

Assuming that the SUT has three books on hand, the ideal testing results would be to get an invoice response for the first three users and an out of stock response for the remaining users as shown on Figure 3 and an overall pass verdict for the test.

However, the results shown in Figure 3 are only ideal and rarely happen. Instead, we see more results of the kind of Figure 4 that show the full effect of race conditions because each PTC starts at different times.

 ISBN: 978-1-61208-776-4

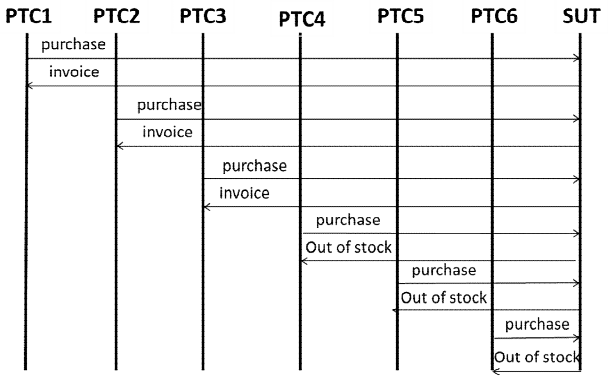

Figure 3. Ideal testing responses

The failures shown in Figure 4 are the result of mismatches between expected and received messages when tests are executed without coordination.

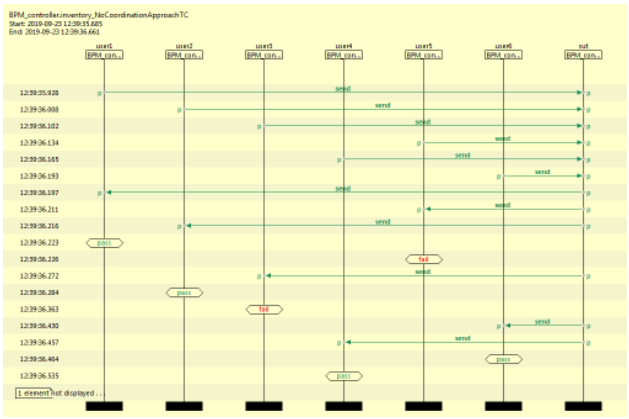

Figure 4. Uncoordinated execution results

Figure 5 shows the TTCN-3 tools data inspection feature [2][3] that provides detailed message and test oracle contents that enable the tester to understand the reasons for failure.

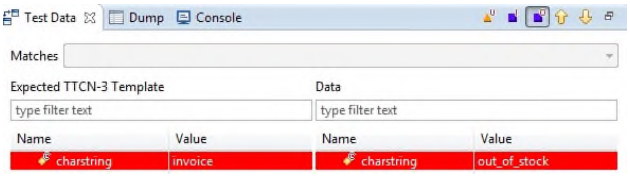

Figure 5. Expected vs received values

In this case, one may wonder where the state value comes from. This is where the test coordination is taking place. TTCN-3 has the concept of main test component (MTC) that precisely looks after that.

In our case the coordination is achieved via abstract coordination ports *cp* that link the master test component and the PTCs as shown in Figure 6.

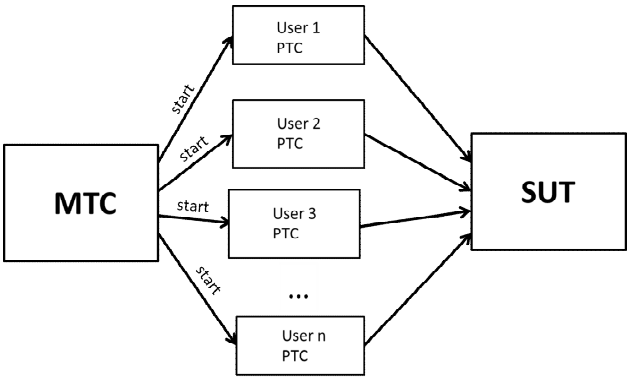

Figure 6. Test coordination with MTC

There are three ways to address test coordination.

1) Using coordination messages

The approach consists in using coordination messages between the MTC and the PTCs that contain the predicted state of the SUT. On the user PTC's side we need an additional line that receives the state from the MTC before the user attempts to test the SUT:

```
cp.receive(charstring:?)->value state;
```

On the MTC side, we send a message containing the state to the PTC that the tester thinks that the server is supposed to be in. In our case this is achieved by changing the state once three requests have been placed as follows:

```
testcase coordinated_msgs_test()
   runs on MTCType system SystemType {
   ...
   cp.send("has_stock") to user1;
   cp.receive("ack") from user1;

   cp.send("has_stock") to user2;
   cp.receive("ack") from user2;

   cp.send("has_stock") to user3;
   cp.receive("ack") from user3;

   // after three purchase requests,
   // the item is now out of stock

   cp.send("out_of_stock") to user4;
   cp.receive("ack") from user4;

   cp.send("out_of_stock") to user5;
   cp.receive("ack") from user5;

   ...
}
```

Figure 7. Test coordination by MTC

 ISBN: 978-1-61208-776-4

In TTCN-3, the *receive* statement is blocking. Thus, the rest of the behavior of the PTC will not execute while the coordination message has not been received.

Note the returned *ack* message. The *ack* is used to prevent racing. In other words, a new individual test cannot occur before the previous test has fully completed, otherwise more requests are being sent to the server which may change its state before a response is sent back to a user resulting in failure. We have observed that removing the ack effectively produces race conditions. We leave this verification as an exercise to the reader.

2) Coordination using PTC Threads operations

PTCs are in fact translated by the TTCN-3 compiler that produces an executable in a general purpose language (GPL) such as Java or C++ and many others using threads. Thus, one typical Thread operation that is available in TTCN-3 is to check if the thread has terminated. This is represented in TTCN-3 with the keyword *done*. Here, as shown in Figure 8, each PTC is started using a parameter representing the function behavior that carries the predicted state of the SUT.

There are in fact two ways to use this feature: the first one consists in placing the done statement immediately after the corresponding start statement. This would result in transforming a concurrent system into a sequential execution system with effects similar to the coordination messages solution shown in the previous section.

```
testcase thread_operations_test()
   runs on MTCType system SystemType {
   …
   user1.start(purchasingBehavior
                       ("has_stock"));
   user2.start(purchasingBehavior
                       ("has_stock"));
   user3.start(purchasingBehavior
                       ("has_stock"));

   user1.done;
   user2.done;
   user3.done;

   user4.start(purchasingBehavior
                     ("out_of_stock"));
   user5.start(purchasingBehavior
                     ("out_of_stock"));
   user6.start(purchasingBehavior
                     ("out_of_stock"));

   user4.done;
   user5.done;
   … }
```

Figure 8. MTC behavior using PTC threads operations

In this second approach, we have chosen to place all the done statements after all the start statements for the first three PTCs to simulate the database reaching its maximum inventory. This has the advantage to at least conserve some of the concurrent behavior of the system and thus avoiding a full sequential test execution of PTCs.

3) Introducing semaphores to TTCN-3

In a way the second approach is less sequential than the first one but still somewhat sequential. Thus, we have explored a third solution that would eliminate some aspects of the sequential aspect of this test behavior. The method consists in using shared variables and semaphores among PTCs. The shared variable keeps track of the inventory on hand and enables a PTC to determine the state of the SUT on its own. However, TTCN-3 does not have the concept of shared variables, neither semaphores and thus we recommend modifying the standard. In our case, we need to declare the inventory variable as shared. TTCN-3 test suites are always translated in a GPL that is then compiled and executed. Since this feature is not available in TTCN-3 we have used an implementation in Java that would be typically comparable to the one generated by the TTCN-3 compiler but somewhat simplified to make it easier to understand.

```
public static void main(String args[])
      throws InterruptedException {

   PTCtype ptc1 = new PTCtype("ptc1");
   PTCtype ptc2 = new PTCtype("ptc2");
   PTCtype ptc3 = new PTCtype("ptc3");
   PTCtype ptc4 = new PTCtype("ptc4");
   PTCtype ptc5 = new PTCtype("ptc5");
   PTCtype ptc6 = new PTCtype("ptc6");

   ptc1.start();
   ptc2.start();
   ptc3.start();
   ptc4.start();
   ptc5.start();
   ptc6.start();

   ptc1.join();
   ptc2.join();
   ptc3.join();
   ptc4.join();
   ptc5.join();
   ptc6.join();

  }
```

Figure 9. MTC behavior using semaphores

Note that the java main method of Figure 9 corresponds to the TTCN-3 MTC test case behavior. The basic difference with the TTCN-3 version shown in

 ISBN: 978-1-61208-776-4

Figure 8 is the presence of the join method that needs to be added to the TTCN-3 standard and the absence of state indication sent to the PTCs. The *join* statement does not exist in TTCN-3 and is part of our recommendation in modifying the standard. Now, we need to show the different modification required in the definition of the PTCs behavior as shown in Figure 2 to implement the semaphores. The PTC type needs first to declare a semaphore as does the Java version. The new TTCN-3 Semaphore data type would merely be translated to the corresponding Semaphore class in Java.

```
class PTCType extends Thread {
      Semaphore sem;
      String threadName;
      String state = "";
```

Then the Semaphore instance needs to have an *acquire* statement as in Java:

```
      sem.acquire();
```

The shared variable *inventory* is then used to compute the predicted state that can be used in the TTCN-3 *alt receive* statement:

```
      if(inventory > 0) {
         inventory--;
         state = "has_stock";
      }
      else
         state = "out_of_stock";

      // place the alt statements as shown on Figure 2 here.
```

And finally add a semaphore *release* statement at the end of the PTC behavior as follows:

```
      sem.release();
```

4) Evaluation

We have observed that the semaphore version of this problem produces a sequence of execution very similar to the first approach using coordination messages. The only difference being that the sequence of the executed PTCs is not entirely in the order of the start of each PTC, i.e. from 1 to 6. Instead the semaphore version produces various sequences of PTC execution but in all remain sequences thus preventing discovering concurrency problems. Thus, we think that the second approach that consists in running PTCs in batches of states is possibly a better approach. However, the second method may run into problems when complex templates are used for depicting for example shopping baskets where the various items may have different limits. In any case, this method is much better than unit testing.

III. TTCN-3 AS A MODELLING LANGUAGE

Normally, testing activities can take place only once the SUT has been fully developed and is runnable. However, planning and developing automated test cases can be done in parallel to the SUT development phase. More importantly, the missing SUT can be emulated using TTCN-3. This enables us to find any flaws in the automated test suites before we apply them to the SUT and thus reduce time to market.

In our case study, this means finding a way to portray a behavior that replies with “invoice” when there is inventory on hand and replies “out of stock” when inventory has reached zero. At the abstract level, there is no need to implement a full system, in our case probably a web application and a related database. The implementation of such an abstract system is as follows:

```
function SUTbehavior() runs on SUTType
{
   var integer inventory := 3;
   var PTCType ptc := null;
   var MTCType mtc_ := null;

   alt {
      [] p.receive("purchase") ->
                          sender ptc {
         if(inventory > 0) {
            p.send("invoice") to ptc;
            inventory := inventory -1;
         }
         else {
            p.send("out_of_stock") to
                                 ptc;
         };
         repeat
      }
      [] ap.receive("stop")
                  -> sender mtc_
         setverdict(pass)
      }
   }
}
```

Figure 10. SUT behavior

We use a simple variable to portray the inventory that we set at 3 units. Every time a request to purchase an item comes in, we decrease the inventory. A simple if-then-else statement provides the correct response of *invoice* or *out-of-stock* state. At the abstract level, this is all we need.

Also, the test suite is developed in two different levels of abstraction. First, we use simplified messages like here simple strings with values. Once we simulate the abstract system and we are happy with the results, in a second step we merely redefine the abstract data types and its corresponding templates (test oracles for received messages and data content for sent messages) as follows:

 ISBN: 978-1-61208-776-4

1st step: Data types and templates declarations:

```
type charstring RequestType;
type charstring ResponseType;

template RequestType myRequest_t :=
                        "purchase";

template ResponseType
           myInvoiceResponse_t
                        := "invoice";
template ResponseType
           myOutOfStockResponse_t:=
                    "out_of_stock";
```

Figure 11. simplified data types and templates

2nd step: Real data types and templates:

```
type record RequestType {
    charstring bookName,
    charstring ISBN
}

type record ResponseType {
    charstring bookName,
    charstring ISBN,
    charstring status,
    charstring action
}

template RequestType myRequest_t := {
    bookName := “ttcn-3 in a
nutshell”,
    ISBN := “978-2-345-678”
}

Template ResponseType myResponse_t :=
{
      bookName := “war and peace”,
      ISBN := “978-2-345-678”,
      Status := “on hand”,
      Action := “invoice”
}
```

Figure 12. Fully realistic data types and templates

Note that both datatypes and templates are defined using the same identifiers. Only their content is different.

IV. CONCLUSION

Despite its long history, testing concurrent systems remains complex and does not always provide accurate results. In this paper we have shown that using formal methods for testing such as TTCN-3 helps to locate problems accurately because of the wide choice of results visualization features that the various commercial and open source editing, and execution tools provide. We also recommended enhancing the TTCN-3 standard by providing shared variables and semaphore features for the MTC and the PTCs. We also have shown a way to partly avoid sequencing PTC test by using batches of concurrent tests by using the current features of TTCN-3.

ACKNOWLEDGMENT

The authors would like to thank NSERC for funding this research.

REFERENCES

[1] ETSI ES 201 873-1, The Testing and Test Control Notation version 3 Part 1: TTCN-3 Core Language, May 2017. Accessed March 2018 at http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.09.01_60/es_20187301v040901p.pdf

[2] TTworkbench,Spirent, https://www.spirent.com/Products/TTworkbench

[3] Titan, https://projects.eclipse.org/proposals/titan

[4] E. Boros and T. Unluyurt, Sequential Testing of Series-Parallel Systems of Small Depth in ISBN 978-1-4613-7062-8

[5] A. Bertolino, Software Testing Research: Achievements, Challenges, Dreams in proceedings of FOSE ’07 pp 85-103

[6] T. Hanawa, T. Banzai, H. Koyzumi, R. Kanbayashi, T. Imada and M. Sato, Large-Scale Software Testing Environment Using Cloud Computing Technology for Dependable Parallel and Distributed Systems in 2010 Third International Conference on Software Testing, Verification and Validation Wokshops procedings

[7] A. M. Alghamdi and F. Eassa, Software Testing Techniques for Parallel Systems: A Survey in IJCSNS International Journal of Computer Science and Network Security, vol 19. No 4, April 2019, pp 176-184

[8] L. Parobek, 7 Reasons to Move to Parallel Testing in white paper on https://devops.com/7-key-reasons-make-move-sequential-parallel-testing/

[9] B. Rao G. , K. Timmaraju, and T. Weigert, Network Element Testing Using TTCN-3: Benefits and Comparison in SDL 2005, LNCS 3530, pp. 265–280, 2005

[10] G. Din, S. Tolea, and I. Schieferdecker, Distributed Load Test with TTCN-3, in Testcom 2006 proceedings, pp 177-196

[11] B. Stepien, K, Mallur, L. Peyton, Testing Business Processes Using TTCN-3, in SDL Forum 2015 proceedings, Lecture Notes in Computer Science, vol 9369. Springer, Cham.

 ISBN: 978-1-61208-776-4