IARIA

SOFTENG 2024

The Tenth International Conference on Advances and Trends in Software Engineering

ISBN: 978-1-68558-178-7

May 26 - 30, 2024

Barcelona, Spain

SOFTENG 2024 Editors

Tadashi Dohi, Hiroshima University, Japan

Luigi Lavazza, Università dell'Insubria - Varese, Italy

SOFTENG 2024

Forward

The Tenth International Conference on Advances and Trends in Software Engineering (SOFTENG 2024), held between May 26-30, 2024 in Barcelona, Spain, 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 2024 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 2024. 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 2024 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that SOFTENG 2024 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 Barcelona provided a pleasant environment during the conference and everyone saved some time to enjoy the historic charm of the city.

SOFTENG 2024 Chairs

SOFTENG Steering Committee
Zeeshan Ali Rana, NUCES, Lahore, Pakistan
Tsuyoshi Nakajima(中島毅), Shibaura Institute of Technology, Japan

SOFTENG Publicity Chairs
Laura Garcia, Universidad Politécnica de Cartagena, Spain
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain

SOFTENG 2024

Committee

SOFTENG Steering Committee

Zeeshan Ali Rana, NUCES, Lahore, Pakistan
Tsuyoshi Nakajima(中島毅), Shibaura Institute of Technology, Japan

SOFTENG 2024 Publicity Chairs

Laura Garcia, Universidad Politécnica de Cartagena, Spain
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain

SOFTENG 2024 Technical Program Committee

Khelil Abdelmajid, Landshut University of Applied Sciences, Germany
Mo Adda, University of Portsmouth, UK
Bestoun S. Ahmed, Karlstad University, Sweden
Issam Al-Azzoni, Al Ain University of Science and Technology, UAE
Vahid Alizadeh, College of Computing & Digital Media - DePaul University, USA
Washington Almeida, Cesar School | Center of Advanced Studies and Systems of Recife, Brazil
Vu Nguyen Huynh Anh, Université Catholique de Louvain, Belgium
Pablo O. Antonino, Fraunhofer IESE, Germany
Darlan Arruda, University of Western Ontario, Canada
Jocelyn Aubert, Luxembourg Institute of Science and Technology (LIST), Luxembourg
Heitor Augustus Xavier Costa, Federal University of Lavras (UFLA), Brazil
Lerina Aversano, University of Sannio, Italy
Ali Babar, University of Adelaide, Australia
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea
Mohamed Basel Almourad, College of Technological Innovation - Zayed University, Dubai, UAE
Bernhard Bauer, University of Augsburg, Germany
Imen Ben Mansour, University of Manouba, Tunisia
Maya Benabdelhafid, Ecole Supérieure de Comptabilité et de Finances (ESCF) de Constantine, Algeria
Marciele Berger, University of Minho, Portugal
Marcello M. Bersani, Politecnico di Milano, Italy
Anna Bobkowska, Gdansk University of Technology, Poland
Pierre Bourque, ETS Montreal, Canada
Fernando Brito e Abreu, ISCTE-IUL & ISTAR-IUL, Portugal
Antonio Brogi, University of Pisa, Italy
Azahara Camacho, RTI - Real Time Innovations, Spain
Qinglei Cao, University of Tennessee, Knoxville, USA
José Carlos Metrôlho, Polytechnic Institute of Castelo Branco, Portugal
Luis Fernando Castro Rojas, University of Quindío, Colombia
Pablo Cerro Cañizares, Universidad Complutense de Madrid, Spain
Allaoua Chaoui, University Constantine 2 - Abdelhamid Mehri, Algeria

Stefano Cirillo, University of Salerno, Italy
Andrea D'Ambrogio, University of Rome Tor Vergata, Italy
Lilian Michele da Silva Barros, Instituto Tecnológico de Aeronáutica, Brazil
Luciano de Aguiar Monteiro, Institute of Higher Education iCEV - Teresina-Piauí, Brazil
Serge Demeyer, Universiteit Antwerpen, Belgium
Amleto Di Salle, University of L'Aquila, Italy
Juergen Doellner, Hasso-Plattner-Institute for Digital Engineering | University of Potsdam, Germany
Tadashi Dohi, Hiroshima University, Japan
Sigrid Eldh, Ericsson AB, Sweden
Gencer Erdogan, SINTEF Digital, Norway
Fernando Escobar, PMI-DF Brasilia, Brazil
Vladimir Estivill-Castro, Universitat Pompeu Fabra, Spain
Naser Ezzati Jivan, Brock University, Canada
Faten Fakhfakh, National School of Engineering of Sfax, Tunisia
Stefano Forti, University of Pisa, Italy
Barbara Gallina, Mälardalen University, Sweden
Atef Gharbi, National Institute of Applied. Sciences and Technology, Tunisia
Pablo Gordillo, Universidad Complutense de Madrid, Spain
Adriana Guran, Babes-Bolyai University, Cluj-Napoca, Romania
Ulrike Hammerschall, University of Applied Sciences Munich, Germany
Noriko Hanakawa, Hannan University, Japan
Qiang He, Swinburne University of Technology, Australia
Philipp Helle, Airbus Group Innovations - Hamburg, Germany
Samedi Heng, Université de Liège, Belgium
Jang Eui Hong, Chungbuk National University, South Korea
Stijn Hoppenbrouwers, HAN University of Applied Sciences / Radboud University, Netherlands
Fu-Hau Hsu, National Central University, Taiwan
LiGuo Huang, Southern Methodist University, USA
Rui Humberto Pereira, ISCAP/IPP, Portugal
Carlos Hurtado Sánchez, Tecnológico Nacional de México - campus Tijuana, Mexico
Miren Illarramendi, Mondragon University, Spain
Shinji Inoue, Kansai University, Osaka, Japan
Anca Daniela Ionita, University Politehnica of Bucharest, Romania
Takashi Ishio, Future University Hakodate, Japan
Faouzi Jaidi, University of Carthage - Higher School of Communications of Tunis & National School of Engineers of Carthage, Tunisia
Jiajun Jiang, Tianjin University, China
Mira Kajko-Mattsson, Royal Institute of Technology, Sweden
Atsushi Kanai, Hosei University, Japan
Afrina Khatun, BRAC University, Bangladesh
Wiem Khlif, Mir@cl Laboratory | University of Sfax, Tunisia
Alexander Knapp, Universität Augsburg, Germany
Sondes Ksibi, University of Carthage | Higher School of Communications of Tunis, Tunisia
Luigi Lavazza, Università dell'Insubria, Italy
Dieter Landes, University of Applied Sciences Coburg, Germany
Seyong Lee, Oak Ridge National Laboratory, USA
Maurizio Leotta, University of Genova, Italy
Horst Lichter, RWTH Aachen University, Germany

Bruno Lima, INESC TEC | FEUP, Porto, Portugal
Panos Linos, Butler University, USA
Hsin-Yu Liu, University of California San Diego, USA
Xiaobo Liu-Henke, Ostfalia University of Applied Sciences, Germany
Qinghua Lu, CSIRO, Australia
Yingjun Lyu, University of Southern California, USA
Damian M. Lyons, Fordham University, USA
Jianbing Ma, Chengdu University of Information Technology, China
Eda Marchetti, ISTI-CNR, Pisa, Italy
Johnny Marques, Aeronautics Institute of Technology, Brazil
Imen Marsit, University of Sousse, Tunisia
Danilo Martínez Espinoza, ESPE, Ecuador / Technical University of Madrid, Spain
Núria Mata, Fraunhofer Institute for Cognitive Systems, Germany
Mohammadreza Mehrabian, South Dakota School of Mines and Technology, USA
Weizhi Meng, Technical University of Denmark, Denmark
Edgardo Montes de Oca, Montimage, Paris, France
Fernando Moreira, Universidade Portucalense, Portugal
Ines Mouakher, University of Tunis El Manar, Tunisia
Tsuyoshi Nakajima(中島毅), Shibaura Institute of Technology, Japan
Krishna Narasimhan, Itemis AG, Stuttgart, Germany
Risto Nevalainen, FiSMA (Finnish software measurement association), Finland
Virginia Niculescu, Babes-Bolyai University, Cluj-Napoca, Romania
Stoicuta Olimpiu, University of Petrosani, Romania
Rafael Oliveira, UTFPR - The Federal University of Technology - Paraná, Brazil
Nelson Pacheco Rocha, University of Aveiro, Portugal
João Pascoal Faria, University of Porto, Portugal
Antonio Pecchia, Università degli Studi di Napoli Federico II, Italy
Fabiano Pecorelli, University of Salerno, Italy
Michael Perscheid, SAP Technology & Innovation, Germany
Dessislava Petrova-Antonova, Sofia University, Bulgaria
Tamas Pflanzner, University of Szeged, Hungary
Fumin Qi, National Supercomputing Center in Shenzhen (Shenzhen Cloud Computing Center), China
Zhengrui Qin, Northwest Missouri State University, USA
Stefano Quer, Politecnico di Torino, Italy
Łukasz Radliński, West Pomeranian University of Technology in Szczecin, Poland
Raman Ramsin, Sharif University of Technology, Iran
Zeeshan Ali Rana, National University of Computer and Emerging Sciences (FAST-NUCES), Lahore, Pakistan
Miary Andrianjaka Rapatsalahy, University of Fianarantsoa, Madagascar
Hajarisena Razafimahatratra, University of Fianarantsoa, Madagascar
Mohammad Reza Nami, Islamic Azad University-Qazvin, Iran
Oliviero Riganelli, University of Milano - Bicocca, Italy
Simona Riurean, University of Petrosani, Romania
António Miguel Rosado da Cruz, Higher School Technology and Management - Polytechnic Institute of Viana do Castelo, Portugal
Gunter Saake, Otto-von-Guericke-Universitaet, Magdeburg, Germany
Sébastien Salva, University Clermont Auvergne, France
Hiroyuki Sato, University of Tokyo, Japan

Daniel Schnetzer Fava, University of Oslo, Norway
Ruth Schorr, Frankfurt University of Applied Sciences, Germany
Josep Silva Galiana, Universitat Politècnica de València, Spain
Rocky Slavin, University of Texas at San Antonio, USA
Cristovão Sousa, Polytechnic Institute of Porto / INESC TEC, Portugal
Sinan Tanilkan, Norwegian Computing Center, Norway
Christos Troussas, University of West Attica, Greece
Harsh Vardhan, Vanderbilt University, USA
Miroslav Velev, Aries Design Automation, USA
Colin Venters, University of Huddersfield, UK
Flavien Vernier, Université Savoie Mont Blanc, France
László Vidács, University of Szeged, Hungary
António Vieira, University of Minho, Portugal
Gianmario Voria, University of Salerno, Italy
Shaohua Wang, New Jersey Institute of Technology, USA
Ralf Wimmer, Concept Engineering GmbH / Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
Xiaofei Xie, Nanyang Technological University, Singapore
Rui Yang, Xi'an Jiaotong-Liverpool University, China
Cemal Yilmaz, Sabanci University, Istanbul, Turkey
Levent Yilmaz, Auburn University, USA
Peter Zimmerer, Siemens AG, Germany
Alejandro Zunino, ISISTAN, UNICEN & CONICET, Argentina
Aditya Zutshi, Galois 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

Project Smells for Early Detection of Problems with Benefits Realization 1
Sinan Sigurd Tanilkan and Jo Erskine Hannay

Software Bug Prediction Based on Semi-definite Logistic Regression Model 11
Tadashi Dohi, Jingchi Wu, and Hiroyuki Okamura

On Reducibility of Developer-Written Unit Tests in C# 17
Arpit Christi and David Weber

Project Smells for Early Detection of Problems with Benefits Realization

Sinan Sigurd Tanilkan
Department of Applied Research in Information Technology
Norwegian Computing Center
Pb. 114 Blindern, Oslo, Norway
Center for Effective Digitalization of the Public Sector
Simula Metropolitan
Pb 4, St.Olavs Plass, Oslo, Norway
Email: sinan@nr.no
0000-0003-4216-5172

Jo Erskine Hannay
Center for Effective Digitalization of the Public Sector
Simula Metropolitan
Pb 4, St.Olavs Plass, Oslo, Norway
Email: johannay@simula.no
0000-0002-8657-7593

***Abstract*—Although substantial research has provided guidance on how to identify and manage the benefits of new software solutions, ensuring the realization of those benefits remains a challenge. Inspired by the notion of code smells for software quality, we develop a concept of *project smells for benefits realization*. We conducted 22 in-depth interviews with participants in nine public-sector digitalization projects, and elicited seven project smells: 1. Dilemma between enthusiasm and formality, 2. Situational differences, 3. Resistance to realization, 4. Slipping opportunities, 5. Loss of focus due to project size, 6. Lacking commitment, 7 Insufficient contact with recipients. We argue that these project smells are a complement to traditional project metrics which focus on time, cost and scope, or the evaluation of benefits after a project is finished. Each smell comes with a set of questions intended to help practitioners identify the odour of their projects. The intention is that project smells can function as low-cost, early indicators helping practitioners adjust work readily and rapidly to ensure benefits realization of their software development investments, thereby focusing actively on the project's product, rather than myopically on the project itself.**

***Keywords*— Software Project; Continuous Product Development; Benefits Realization; Agile; DevOps; BizDev.**

I. INTRODUCTION

Keeping track of the status of a software engineering initiative is important for all who have stakes in investing, developing and benefiting from the system under development. The purpose of such monitoring is to ensure project and product success.

The iron-triangle metrics time, cost and scope are often habitually monitored during software development. However, keeping track of the benefit of the system under development is not regularly done [1]. This may result in suboptimal choices in terms of what functionality to develop early and may lead to poor decisions on development progress, and even termination, based on time, cost and scope alone. Indeed, a recent study found that software professionals perceive that important decisions in software development are mostly based on rationales in terms of time, cost and scope, rather than on benefit or the benefit/cost ratio, but that they think decisions should be based on benefit or the benefit/cost ratio to a greater extent [2].

Early on, Baccarini made the distinction between project management success and product success and defined overall project success as a healthy balance between the two [3]. Project management success is delivering on time, cost and scope, while product success is delivering software that generates value, or benefit. In other words, software engineering is, in many cases, suffering from myopia on project management success, even though practitioners, policy makers and academics call for a greater focus on organizing work to ensure that the software is capable of delivering benefit.

The field of *benefits management* arose to address the lack of attention on benefits in development initiatives [4]–[10]. Even though studies suggest that projects that engage in benefits management activities during project execution [1] are more successful on delivering benefit (but also on time, cost and scope), the adoption of such activities has been low [1], [11]. Of several reasons for this, we highlight that benefits management is for the most part formulated at the portfolio and project program level (such as in Managing Successful Programs®), and benefits management activities during development mostly relate to strategic and organizational aspects, such as paying attention to the business case and the benefits realization plan, but with no operational means for doing so.

Considerations of the benefit of a system under development may belong at the strategic level of business cases and at the organizational level for planning how to use a system under development to produce value in organizations and in society. Nevertheless, it has been argued convincingly that to realize such high-level plans, one must also move benefits considerations from the strategic level onto the operational level. A step on this route is to measure the realization of intended benefits [4], [5], [10]. If this is done in an incremental-development setting, measurements on how beneficial an increment is can be fed back to provide project learning on benefit for successive increments. However, such data is usually not used in this way, but rather collected after a project is completed as part of reporting [5], [10].

Methods have been developed for monitoring how well the project is doing in producing valuable software during project execution, for example, using *lead indicators* of benefits [5], [12]. Another approach is to use *benefits points* (and related

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

techniques) [13]–[16] to estimate and keep track of a system's potential for realizing benefits during project execution [17]. This includes using lag indicators to update lead indicators to take advantage of learning during project execution. However, getting industry to take such techniques into use in a broad scale requires integrated tools for managing daily work (e.g., Jira), which are under development but not operational [18], and an appropriate organizational mindset, which seems to depend more on vogue grey literature than academic studies.

In this article, we approach the problem of keeping track of software engineering work, and specifically with respect to benefits creation, not from a managerial standpoint, but from the point of view of the working software engineer. To update lead and lag indicators for benefit one needs ground truth from those close to development and close to stakeholders affected by the system. To provide a lower threshold utility, and inspired by the work of Fowler et al. on code smells [19], we elicit and elaborate the notion of *project smells for benefits realization*. While *code smells* help to identify software design flaws, these *project smells* help to identify concerns regarding the benefits from software projects. We will argue that project smells can help provide a shared view of project status between software engineering teams and their managers. This addresses the issue that project participants may be aware of projects that are in trouble, while management is unaware [20]. This topic has been studied under the term employee silence – when individuals remains silent about project concerns or problems, resulting in a situation where the project's true status is not known [21]. Project smells for benefits realization can help give voice to such concerns.

Section II presents relevant background. Section III describes the development of the project smells notion, moving from interviews, through coding and concept development to concept operationalization in smells. We present and discuss the results in Sections IV and V. We consider limitations in Section VI and conclude in Section VII.

II. Background and Previous Work

The term *project smells* has been used previously, with different meanings. The first academic mention of project smells in 2007 focused on end-of-project retrospectives. No elaborations of the project smells were provided, but the proposed purpose of project smells is to "... alert us to a broken or ill-fitting process" [22].

Three months later, the term project smells was used again, in the context of software testing. Three categories of test smells were proposed: 1) test code smells, 2) automated test behaviour smells, and 3) project smells [23]. Factors to identify project smells in this context are: i) buggy tests, ii) developers not writing tests, iii) high costs of test maintenance, and iv) bugs in production. The purpose of project smells in this context is that "... project smells are likely to be the first hint they get that something may be less than perfect in test automation land" [23].

The third academic reference for project smells was published recently, within machine-learning project management [24], where project smells are presented as a holistic view of software quality in machine learning projects, and code smells are considered to be a part of project smells. Here, characteristics of project smells are lack of 1) dependency management, 2) version control, 3) unit testing, 4) proper configuration of continuous integration, and 5) effective static analysis tooling [24]. Project smells are evaluated using a static analysis tool on the source code, the data and the tools configuration in the projects. The purpose of project smells in this setting is thus to provide feedback to practitioners on their use of a set of practices for machine-learning quality assurance.

Smells have also been used for the assessment of agile practices in organizations. Agile smells [25] constitute a catalogue of practices that are considered suboptimal in an agile context. Simlilarly, Mike Cohn, has written an article about scrum smells [26] – a catalogue of suboptimal practices in scrum.

The above smells revolve around appropriate work practices and delivering on time, cost and scope, and also software (intrinsic) quality. What is absent, then, from our vantage point, are smells concerning the benefits of the software.

The field of managerial problem solving brings relevant perspectives to the table and regards problem solving as consisting of two parts: 1. problem formulation and 2. problem solving [27]. Project smells for the early detection of problems (ground truth) with benefits realization falls into a subset of the first category, specifically *problem detection*, concerning observations "... that events are taking an unacceptable trajectory and may require action" [28]. Once a problem is detected, people can choose from a set of option categories: look for more information, pay more attention to related events, attempt to identify the underlying problem, discuss the concern with others, explain away the observation or take action to manage the problem by mitigating actions or accepting a change in situation and updating plans and goals [28].

However, the above options are only available once the problem has been identified. An interesting aspect of managerial problem detection is its contrast to the detection of managerial opportunities and crises in terms of the relevant stimuli. While both opportunities and crises are often stimulated by one single idea or triggering event, problems often require multiple stimuli, and problem-stimuli are often milder than stimuli from opportunities and crises [29]. Also, decision makers have a tendency to desire more information about problems before they act [30]. This makes problem detection more complex and less clearly delineated than other critical events in management.

Then, the three most important factors for identifying a problem is: *expertise*, *stance* and *attention management* [28]. While expertise is intuitively understood, and most would agree that an experienced project manager is more likely to spot project problems, stance and attention management warrants a short description. Stance is a person's position towards a situation [31]. Stance (or general alertness) can range from denial (nothing can go wrong) to being confident that any

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

obstacle can be overcome, to an alertness that problems may arise, to hysteria (over-reacting to every minor indication) [28]. Attention management, on the other hand, focuses on what is monitored (and ignored) [28]. This is central to our discussion, because we have a tendency to overlook relevant information, even when it is just in front of us [32], [33]. Although the project smells are unlikely to affect people's expertise and stance, it is our hope that the project smells can help guide practitioners' attention to factors that help them with early detection of problems with benefits realization.

III. Research Method

The present study is an elaboration of a particular concept that was elicited in a larger qualitative study based on thematic analysis of interview data. Several concepts emerged in that larger study, and it is the further elaboration of the concept *Characteristics of projects that affects the realization of benefits* that is presented here. More information about the full study can be found at [34]. Focusing on a single concept from a larger study is recommended to allow one to go into detail in that particular concept [35]. That concept is the basis for the seven project smells for benefits realization that will be presented in Section IV.

The research was conducted using the Stepwise-Deductive Induction (SDI) method [36]. The SDI method is a structured qualitative method for building concepts and theories, grounded in empirical data. Our intention is not to develop new theories, so the last step of the SDI method – theory development – is omitted. The phases of the SDI method used here are:

1) Case selection and data generation
2) Processing of raw data
3) Coding
4) Code grouping
5) Concept development

Moving from one phase to the next is an inductive move. Deduction is used to test the results from each phase by comparing the results to the data that formed the input to the phase. Abduction is used in the latter phases (primarily during concept development), to find plausible explanations for the observations [37].

In addition to the phases of the SDI method we added a sixth phase, which operationalizes the developed concepts into actionable tools; in this case, project smells and questions to keep in mind to become aware of the smells:

6) Operationalization of the concept

Braun & Clarke [38] distinguish between different approaches to thematic analysis. The *neopositivst* approach, at one end of the scale, focuses on objective and unbiased coding. In that approach, it is common to use a predefined codebook and have multiple coders, so that agreement between coders can be measured numerically as a measure of coding reliability [39]. At the other end of the scale is the *reflexive* approach, where coding is "... open and organic ..." [38], with no predefined codebook. An important distinction between the two ends of the scale is when themes or concepts are developed. In the neopositivst approach, themes are developed early, often prior to coding, while in the reflexive approach, themes and concepts are the final outcome of the analysis. The SDI method used here is at the reflexive end of the scale, where codes are developed inductively from the data. Still, coding reliability, which is often not a concern in reflexive thematic analysis, has a strong focus in the SDI method. While the neopositivist thematic analysis approach to coding reliability is often handled by codebook design and using multiple coders, coding reliability in the SDI method is handled by adhering to strict coding rules (see Section III-C).

Coding for this study was conducted by the first author only. To ensure conceptual clarity at the higher levels (Steps 5 and 6), the code groups of the concept under study and their operationalization in smells were discussed extensively between both authors.

The following sections describe how the the six phases were applied in this study.

A. Case Selection and Data Generation

Due to the low adoption of benefits management in practice [40], it is challenging to find organizations where its use can be studied. The Norwegian Digitalization Agency has a funding program for public-sector digitalization projects, where one of the conditions for funding is the active use of certain benefits management practices, as described in [41].

We invited all the projects that received funding in 2016 to participate in the study. Nine out of twelve projects chose to participate. All included projects had a duration of three years, except one, which had a duration of two years, and funding was granted up to 50% of the net project costs with an upper bound to funding at NOK 15 million (approx. USD 1.9 million at project completion time).

All the studied projects involved the creation of a new software solution for digitalization in the public sector. The new software included solutions for:

- data sharing,
- unique data storage (to avoid a situation where data is duplicated and out of sync across organizations),
- providing individuals and organizations with self-service to public-sector data and applications,
- automating previously manual and/or paper-based case processing,
- guiding individuals and organizations on how to use public-sector services.

Examples of benefits from the new software solutions included more efficient use of resources in the public sector, improved quality of data in the public sector, faster response times when interacting with the public sector and improved rules of law.

Data was collected through 22 face-to-face interviews with professionals involved in the projects. The interview questions are available at [34]. We chose a semi-structured approach in order to follow (other) topics that respondents brought up as

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

relevant to benefits management and realization. Interview participants included project sponsors, project managers, people responsible for benefits realization, project team members and one benefits recipient.

Interview duration ranged from 25 to 120 minutes, depending on the amount of relevant information the respondent had to provide. For all interviews, two researchers and one respondent took part, except for five interviews. Four interviews were conducted with one researcher and one respondent, and one interview was conducted with one researcher and two respondents. Interviews were conducted face-to-face, in the premises of the studied organizations.

All interviews were recorded using an audio recorder and notes were taken. Due to strict confidentiality agreements, none of the collected raw data is made available; neither are organization names nor exact project topics.

B. Processing of Raw Data

The recorded audio files were transcribed, resulting in 612 pages of transcribed text. After transcription was completed all audio files were listened to while simultaneously reading the transcribed texts to ensure correct transcriptions.

C. Coding

All the transcribed texts were coded using NVivo (release 1.7.1). In total, 274 codes related to the concept focused on in this study were created.

A two-step code test was applied to all codes to ensure that the codes represented the respondents' statements [36]. This approach is designed to reduce the potential biases of having only one coder [39], when relevant. The applied code tests suggested by [36] emphasize groundedness in data, and semantic codes, rather than mere sorting codes: 1. If the code could have been created prior to seeing the data, this is considered an *a priori* code, and a different code should be created based on the data. 2. If the code only labels topics in the data, e.g. "quantification of benefits", this is considered an unnecessary sorting code, and a different code should be created that reflects what the respondent expressed, e.g., "Numbers makes people lose interest".

To give the reader an impression of the codes and the respondents' statements that the codes represent, examples of codes and corresponding extracts from the interviews are included in the results section (Section IV).

D. Code Grouping

Codes were grouped thematically, using a code grouping test. Rather than applying this test at the end of the phase (as suggested by [36]), the grouping test was used as a guide or condition when placing codes into groups. The grouping test checks that when adding a code to a group, the group should still be thematically different from the other groups and the content of the group should still be consistent. If a code cannot be placed in any group (and still fulfil these conditions), a new group should be created.

Due to the large number, the code groups were divided into two levels. This approach is supported by [36, p. 210], who suggests that when there are more than 3–5 code groups, it can be useful to organize them into more than one level. The five high-level and eight low-level code groups are the primary building blocks of the concept under elaboration here, and the results section (Section IV) is organized accordingly.

E. Concept Development

The concept *Characteristics of projects that affects the realization of benefits* evolved using abduction – moving back and forth between the code groups and relevant background knowledge and theories [42] to consolidate the concept. Recall that the abduction process was performed in the context of the larger study, and the concept was generated in relation to the other concepts in that encompassing study. All these were tested by considering how well they described different subsets of code groups.

F. Operationalization of the Concept

From that concept we distilled the project smells for benefits realization. Within each high-level code group, this was done by operationalizing the lower-level code groups into actionable tools for practitioners; namely indicators (smells) and actions (questions to be asked). These proposed indicators and actions can be found as subsections of Section IV, starting with "Smell:".

We tested the notion project smells with other candidate denotations, such as "characteristics of benefits", "project heuristics", "product smells" and "project status detectors" by presenting them to practitioners with experience from software projects, with continuous product development [43], [44], and with project managers of construction projects, to get feedback and see how well the different denotation resonated in the different settings. The denotation "project smells for benefits realization" was kept.

IV. Results

The total SDI-analysis resulted in several conceptual topics, where one of these concepts is *Characteristics of projects that affects the realization of benefits*, which is the topic of this paper. The concept is built on the following high-level code groups (1–5) and low-level code groups (a–d):

1) Motivation
    a) The importance of caring about benefits
    b) Factors that affects peoples' motivation for benefits
2) Understanding
    a) Familiarity
    b) Proximity to domain
    c) The ability to understand resistance to benefits realization
    d) The ability to understand possibilities
3) Project size
4) Dependencies
    a) Changes in regulations
    b) Contributions from other organizations
5) The need and ability to reach benefits recipients

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

In the following subsections we describe the code groups, exemplifying them with extracts from the interviews and the accompanying codes. The project smells are given after the corresponding code groups have been presented.

A. Motivation

1) The importance of caring about benefits: Simply caring about the benefits of the system under development is reported to be important, both to motivate people to conduct measurements and to make the necessary adaptions when needed, as exemplified in the following excerpt (code: *Caring affects effort*):

> It is important that you care about realizing the benefits. If not, you will neither collect the necessary measurements, nor take the necessary actions when deviations from the plan occur.

While the above example illustrates that caring about the benefits is important for those working to provide them, caring is also important for those receiving benefits. After explaining an unanticipated benefit that was raised by an external organization, the following respondent stated that the benefits recipients' motivation was important in making this benefit materialize (code: *Interested recipients lead to unanticipated benefits*):

> They [the benefits recipients] are perhaps more than average interested ... and it turns out that they found benefits that we had not anticipated.

This observation is important, because it indicates that highly motivated recipients can lead to more, or further, benefits than anticipated.

2) Factors that affects peoples' motivation for benefits: Respondents report that people are easily motivated to work for society, as in the following excerpt (code: *Easy to motivate people for societal benefits*):

> It is not difficult to motivate [role] to work for society because they see that this is beneficial. They are driven by ... that is, they get energy from it. This is not merely an academic exercise, this is production of benefits for society.

In addition to the type of benefit, such as societal benefits in the above example, the way information about the benefits is shared also seems to affect peoples' enthusiasm for the benefits. In particular, verbalizing benefits, without emphasizing metrics and returns has been advocated (code: *Talk about benefits but not the numbers*):

> I think it helps just to talk about the benefits. Turn them into something concrete. That is, without focusing on the numbers, that is perhaps not something that motivates people.

Indeed, quantifying benefits seems to be a proper turn-off (code: *Numbers makes people lose interest*):

> Talking about the numbers in the excel sheet makes many people lose interest.

The above demonstrates a preference for verbal descriptions of the benefits over the numbers relating to benefits. This is further corroborated by the following statement highlighting story-telling (code: *Storytelling provided common direction*):

> I believe the pilot project was very important. But I was the only person who took part in that and the current project, and I think it was the story that was developed in the pilot project that people bought into. They immediately understood where we wanted to go, or the direction we were going in.

There also seems to be more engagement in benefits that people can envision thorough their daily work over the benefits reported in the benefits plans (code: *Driven by real, not artificial benefits*):

> *Respondent*: It has been a collaboration. And she has worked with them very much, also with the municipalities, in order to realize the benefits. But it's not... I don't think the benefits is what drives her. I think it is... That is, she genuinely cares about people, including the municipalities ... There are many qualitative benefits here that she is able to realize as a result of dialogue with the municipalities and other agencies.
>
> *Interviewer*: When you say that she was not driven by benefits, but that she genuinely cared. What is the difference between the two?
>
> *Respondent*: I think there is a difference between the benefits in the benefits plan and the benefits talked about in daily work, but I don't know how to explain it. One of them [the benefits] is perhaps something you have under your skin. Something you feel ownership of. You feel an ownership of a product, and by... You understand that this can lead to a process improvement for example, or that this simplifies things, makes them more efficient etc. That is, we spend much time in the domain and understand what is good for the user ... While the benefits in the benefits plan are much more narrow. And perhaps a bit artificial.

The engagement and enthusiasm that good stories and personal experience create seems to be pivotal for the motivation to realize benefits. Structured, and perhaps quantified, specifications of benefits do not seem to motivate to the same degree and even seem to demotivate.

Somehow, though, size (quantification) still matters (code: *Small benefits feels meaningless*):

> Sometimes the benefits are small, and that feels meaningless ...

Although the motivation for benefits realization might hinge on engagement and enthusiasm rather than more formal specifications, we will not conclude that the former should be chosen over the latter. However, the findings suggest that it is important to be aware of this possible dilemma, since suggested methodologies both by academia and governing bodies emphasize the explicit, clear and measureable specification of benefits. To make this dilemma explicit, we declare the

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

following smell, and propose to make a habit of asking the accompanying question:

3) Smell: Dilemma between enthusiasm and formality, Questions to ask:: Are the benefits and the motivation of the relevant stakeholders in harmony? How do the specified and the unspecified benefits relate to each other?

B. Understanding

1) Familiarity: Having experience from previous work that is similar to what lies ahead is reported as a success factor in realizing benefits (code: *Previous experience from similar project aided in realizing benefits*):

> We had a similar project for another internal portal... which had many similarities. The majority of challenges were in areas where the projects were different.

2) Proximity to domain: When benefits span multiple organizations, the ability to recognize possibilities seems to decrease when a stakeholder's distance to the relevant domain increases (code: *Distance between domains reduces understanding*):

> They are further away, so you need more frequent meetings ... they might not have the same semantic understanding of the information provided to them ... It's like using a topographic map to navigate at sea, which can have grave consequences. If you are within the same agency, you usually have a pretty good understanding. If you are within the same sector, you might still have a pretty good understanding. When we start to talk about services that go across sectors, understanding starts to be reduced, and when you move into the private or municipal sector it is even worse.

Both familiarity with similar work and proximity to domain are well-known success factors when it comes to controlling the iron triangle factors cost, scope and time, and caution has been raised toward using data and experience from earlier initiatives that are not very similar [45]. Here, these perspectives appear for benefits realization as well.

3) Smell: Situational differences, Questions to ask:: Do those who need to understand the benefits, the conditions for the benefits, and the relevant situational factors have sufficient understanding? Do we have the necessary conditions (such as time and mindset) and data (on situational factors) to increase our understanding?

4) The ability to understand resistance to benefits realization: When encountering impediments or resistance to benefits realization, understanding the domain where resistance occurs is important in order to evaluate 1) if the resistance is warranted or based on false assumptions and 2) how to mitigate the impediments/resistance.

An example of warranted resistance can be seen in the following excerpt. Here the new process resulted in loss of access to information for a user group. This information was necessary for the users to complete their job assignments. Through dialogue with the users and understanding of the users domain, a solution was found by changing regulations and changing the new software solution so it would provide an aggregated version of the data to the users (code: *Resistance mitigated by providing new functionality*):

> And this brings us to the [name of user group]. We had not thought about them. Losing access to the data from the previous process, they could not complete their responsibility of [responsibility] ... The ministry of health and care services helped us change the relevant regulations, providing us with the legal basis for including the statistics that [name of user group] needed.

Resistance to the new solution can also be unwarranted. One organization that lost access to information when introducing a new software solution saw this as a problem, because the information was important for their tasks. However, it turned out that the process used by the organization was not in accordance with current regulations. Even though representatives from the organization had wanted to continue using the old process when the new software solution was introduced, they learned that this was not an option, without changes to regulations (code: *Resistance mitigated by understanding and sharing information*):

> They were very worried about losing access to [information name] ... We prepared well and invited the [organization name] to a meeting, with our lawyers. In the meeting we explained that the processes they followed today were not in accordance with the relevant regulations. And if they need to continue as before, they need a change in regulations in their sector. So during the meeting they did not get any of what they wanted, but it was still a positive meeting, and they thanked us for informing them and preparing and presented the case well. The key was involvement, and that functioned well.

In both the above examples, understanding – and working to understand – was key to 1) evaluate if the resistance was warranted, and 2) decide how to handle the resistance.

5) Smell: Resistance to realization, Questions to ask:: What resistance to benefits realization are there among stakeholders? How warranted is that resistance?

6) The ability to understand possibilities: The ability to understand possibilities is reported as important for successful benefits realization. A challenge raised in this regards is that many people seem to have a "linear" way of thinking, which limits their perception of new possibilities (code: *Some have a linear way of thinking*).

> We were struggling to get ownership in the line organization. We still struggle with that ... When we have new needs, they don't consider self service to be an option. They recreate their work process in a modern architecture. So if we are talking about processing of [case type anonymized], such as changing the name of a [anonymized object type], they expect a

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

> more efficient system for registering this ... While in reality, the users can do it by themselves ... It is characterised by not thinking about new ways to do things. A linear way of thinking based on how we used to work.

The ability to recognize opportunities requires personal chracteristics, including competence on digitalization and on the relevant domains (code: *Need digitalization and domain competence to see possibilities*):

> To look up and see the possibilities, then you need a person that is able to look outward, and onward. Understand how data can be used and understand user needs ... You are dependent on a person who can think in terms of digitalization, think new ... And you need to understand the tools ... You need IT competence and you need domain competence in order to understand the user.

7) Smell: Slipping opportunities, Questions to ask:: Do we have the conditions necessary for recognizing new opportunities for benefits realization? Do those who are in position to recognize new opportunities have the necessary understanding of those conditions and the competence of the relevant domains?

C. Project Size

In general, respondents report that it is more challenging to succeed with realizing the benefits from larger projects than from smaller projects. Reasons include that smaller projects require less followup and that it is easier to make people interact when the project is small. Further, larger projects have more tasks that are unrelated to benefits creation, and there are more things that can go wrong in larger projects

Still, smaller projects can suffer from a lack of priority, having key resources who are allocated late or shared with other projects, and difficulties in obtaining assistance.

As the statements related to project size are straight-forward, we do not include excerpts here. However, what seems to be the common denominator mentioned with regards to project size is the challenge to maintaining a focus on benefits.

1) Smell: Loss of focus due to project size, Questions to ask:: Is the project maintaining a focus on benefits realization in the face of organizational size issues, such as overhead and complexity (large initiatives) and lack of priority and visibility (small initiatives)?

D. Dependencies

Dependencies at work outside of the project or within the organization can affect benefits realization negatively.

1) Changes in regulations: Digitalization in the public sector often involves the processing of personal data. In the digitalization projects we studied, it happened that the project uncovered that processes were not defined according to regulations or that the new process required changes in regulations (both of which are exemplified in Section IV-B). Dependencies on changes in regulations puts benefits at risk, especially when there are uncertainties about the regulations.

2) Contributions from other organizations: The observed collaborations have been less focused on contractual agreements and more focused on pragmatic collaboration to realize benefits. An effect of this collaborative basis is that people's and organizations' contributions are based on the different stakeholders' perceptions of benefits, rather than on a set of agreed-upon common benefits to be achieved.

Respondents reported problems with this collaborative basis when other organizations did not contribute with what was necessary. However, one project which seemed especially successful in ensuring contributions from the other organizations described how they worked actively to keep organizations involved (code: *Ensured contribution by keeping organizations involved*):

> This work [ensuring external organizations' contribution] started at day one. When we wrote the mandate for the pilot project, we collaborated with [contributing organization]. And all the contributing organizations were involved in the pilot project. We had defined seven domains and spent a week exploring each of them to understand the situation ... So they had taken part in describing the problem, as much as they had contributed to designing the solution. And then we had them with us. Since then we have had a regular meeting every Wednesday ... to work with the needs, look at and comment designs and user stories, and to test each iteration before final user testing. These meetings has continued even after the project was finished.

The degree of involvement that makes sense for each contributing organization is likely to vary from situation to situation, but being aware of the potential fragility of their involvement and contribution has been raised as a concern.

3) Smell: Lacking commitment, Questions to ask:: Which external parties are we dependent upon and how confident are we in their (continued) involvement and contribution?

E. The Need and Ability to Reach Benefits Recipients

It is often necessary to interact with those who are supposed to benefit from a system (the benefits recipients) to help or make them use the system in their work or life processes. Direct benefits recipients are those who get benefits from the system itself, while indirect recipients get benefits as follow-on effects of the effects that the direct recipients experience. Constellations of those who receive the benefits of a system vary from a few direct recipients, through many direct recipients to a mix of direct and indirect recipients. When there are only a few benefits recipients, spending time on each recipient might not represent a large cost. When the number of recipients is large, the amount of time spent on each recipients is often expected to be low. This can pose a problem when adoption does not go as expected; especially when the diversity among benefits recipients is large. That is, the cost of reaching all recipients is large when there are many, or they cannot be reached directly, such as when benefits recipients are a peripheral part of the process far away from the organization owning the new process/solution (code: *We cannot get in touch with all recipients*):

> When creating a self service solution, we cannot get in touch with all [anonymized profession], or all citizens of Norway, to make them use the new solution.

1) Smell: Insufficient contact with recipients, Questions to ask:: Do we need to, and do we have the resources to, reach the benefits recipients to ensure benefits realization?

V. Discussion

The benefits of a system impacts work and life processes and are (or should be) rooted in business, organizational and societal goals. Understanding how a system will contribute to those goals – that is, understanding the system's benefits – through shifting and evolving technological, organizational and psychosocial mechanisms is to understand a complex and opaque problem (Section II). According to one school of thought, human beings have evolved to make *good enough* (satisficing [46]) judgements on minimal cues in complex situations [47], [48]. However, due to this complexity and opacity, practitioners often lack data and/or the appropriate verbalizations to back those perceptions and judgements. As a result, actions might be taken late or not at all.

The projects smells that emerged in this study can be seen as empirically-based encapsulations of practitioner insights for ensuring better benefits realization. The smells embody good-enough actions in a complex and opaque environment. Relating again to managerial problem-solving (Section II), the three most important factors for people in identifying a problem – expertise, stance and attention management [28] – can function as impediments to problem detection. While influencing people's expertise and stance can be costly and time consuming, we believe the project smells can be a low cost solution to focusing people's attention on factors that are important for realizing the benefits of software projects. If organizations include project smells for early detection of problems with benefits realization as part of what they keep track of – and pay attention to – the threshold for problem-stimuli to actually be detected may become lower. This should put practitioners in a situation where they can react timely to problem-stimuli.

If adopted in an organization or in a project, project smells could provide important reflection points for software engineers with a legitimacy for concerns raised by those closest to were relevant observations are made. The time horizon for actions based on project smells should not be in the future, at the stratgeic level, but rather, in the short term or immediately.

Smells are not termination indicators. They are indicators that something must be done to make a situation better, but this has to happen on time. Projects who practice benefits management activities *during project execution* seem to be more successful on benefits realization, and also on other success criteria [1]. Brooks famously said: "How does a project get to be a year late? ... One day at a time" [49]. Thinking analogously for benefits realization, the need for day-to-day adjustments becomes pertinent. These day-to-day adjustments can only be done if software engineering teams and their managers understand what is going on. The project smells, we argue, helps teams identify and understand what is going on regarding benefits realization and can help practitioners to identify the right time to take action. Rather than the project smells being binary warning lights that managers should monitor at the cost of everything else, we think people should hold them in mind to guide them when talking with people or otherwise observing their projects.

There are a myriad of recommended sensible actions one can take in development initiatives with the aim to get good results, but it is hard to tell what, of all these things, to do and when to do it, before it is too late. In hindsight, there are often few surprises to what went well or wrong, but the trick is to do something before the fact, and the project smells are an empirically-based contribution to that.

We do not address the follow-up question of what to do for each project smell and who should do it. While this may seem as an omission, trying to list all the sensible ways to react to each smell quickly comes out of hand. Indeed, whereas the smells are generic, the ways to take action must be specific and depend on the details of the initiative's organization and culture. These specific details will also influence how the interaction of events which might lead to a mix of smells.

VI. Limitations

The main threats to validity to the empirical study are construct validity and external validity [50], [51].

A. Construct Validity

Construct validity for the SDI method concerns the extent to which the concepts are well-defined (accuracy) and whether they are validly founded in the data (reliability) [36], [51]. As mentioned, the concept under elaboration in this article ("characteristics of projects that affects the realization of benefits") is one of several concepts elicited in a larger study. Although the reflexive approach does not see a single reviewer at lower levels of coding as threat to validity, scholars versed in the neopositivist tradition might still consider this a threat to construct validity. The SDI method itself has safeguards to heighten validity, even when using one reviewer. Moreover, the fact that many concepts were elicited in the larger study, which demands extensive adjusting of the various concepts to gain a level of integrity and distinction for each concept relative to the other concepts, also gives credibility for accuracy and reliability for the concept under elaboration here. In this study, the concept was also refined further by both authors. In our case, construct validity justifies generalizability, roughly speaking, to situations for which the concept, including the project smells which are derived from the concept, applies.

B. External Validity

This concerns the extent to which the results obtained for the study's sample and situation hold across other samples and situations. The sample is designed, rather than random, in that the

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

projects were incentivized to perform benefits management. In the outset, this poses threats to generalizability. However, the sample is particularly relevant to the topic of interest, which increases the construct validity of the responses. This is advantageous for conceptual development, which is our aim in this study. Also, the sample is *critical* [52], in that challenges with benefits management, and project smells, that appear in the sample are arguably even more present in non-incentivized settings. On the other hand, external validity may be reduced, since our sample may be biased by special interest in the topic and that the sample is from the Norwegian public sector. To validate the concept and the smells, further studies should be conducted with other samples and in other development contexts.

VII. Conclusion and Further Research

Further studies that observe the use of the smells will tell the extent to which the proposed seven project smells for benefits realization are useful. Observational studies, in similar and related context, will hopefully lead to refinements of the smells and the identification of further smells.

Benefits realization concerns the effects of using a software product, and the idea is that the project smells will facilitate a product focus in projects. Further, it is natural to study the smells in product-centric development, where cross-functional autonomous teams are responsible for the entire lifecycle of functional areas.

The notion of code smells inspired, in form, our notion of project smells. Work on code smells has been ongoing for close to 25 years, and both manual and static approaches to identifying code smells have been applied [53]. Project smells for early detection of problems with benefits realization, as presented here, is not even at the level of the first publication on code smells [19], which have names and labels for a large set of different smells. Although labelling the identified project smells is easy enough, we propose to postpone that exercise, and rather focus on the understanding that lies behind each project smell. That way, the labels (which are less important than the understanding) can come at a later point, when the categories of smells has congealed as a result of further research.

Acknowledgments

The authors are grateful to Magne Jørgensen and Geir K. Hanssen for their contributions in designing and conducting interviews. The authors are further grateful to the interview participants for sharing their time and insights, and to Maurizio Giardini, Eckhart Köppen, Stian Støle, Lene Renneflott, Adam Lawson and Kristoffer Flottorp for their feedback and reflections on the developed concept.

References

[1] M. Jørgensen, "A survey of the characteristics of projects with success in delivering client benefits," *Information and Software Technology*, vol. 78, pp. 83–94, 2016.
[2] S. S. Tanilkan and J. E. Hannay, "Benefit considerations in project decisions," in *Proc. Int'l Conf. Product-Focused Software Process Improvement (PROFES)*, pp. 217–234, Springer, 2022.
[3] D. Baccarini, "The logical framework method for defining project success," *Project management journal*, vol. 30, no. 4, pp. 25–32, 1999.
[4] G. Bradley, *Benefit Realisation Management: A practical guide to achieving benefits through change*. Routledge, 2016.
[5] S. Jenner, *Managing Benefits: Optimizing the Return from Investments*. The Stationery Office, APMG-International, 2014.
[6] C. Lin and G. Pervan, "The practice of IS/IT benefits management in large Australian organizations," *Information & Management*, vol. 41, no. 1, pp. 13–24, 2003.
[7] T. Melton, P. Iles-Smith, and J. Yates, *Project Benefits Management: Linking projects to the Business*. Butterworth-Heinemann, 2008.
[8] M. Payne, *Benefits Management: Releasing project value into the business*. Project Manager Today, 2007.
[9] J. Thorp, *The Information Paradox: Realizing the Business Benefits of Information Technology*. McGraw-Hill, revised ed., 2007.
[10] J. Ward and E. Daniel, *Benefits Management: How to increase the business value of your IT projects*. Wiley, 2nd ed., 2012.
[11] R. Breese, S. Jenner, C. E. M. Serra, and J. Thorp, "Benefits management: Lost or found in translation," *International Journal of Project Management*, vol. 33, no. 7, pp. 1438–1451, 2015.
[12] Infrastructure and Projects Authority (UK), "Guide for effective benefits management in major projects," guidance to practitioners, Infrastructure and Projects Authority, 2017.
[13] J. E. Hannay, H. C. Benestad, and K. Strand, "Benefit points—the best part of the story," *IEEE Software*, vol. 34, no. 3, pp. 73–85, 2017.
[14] C. Larman and B. Vodde, *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Addison Wesley, 2010.
[15] D. Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs and the Enterprise*. Addison Wesley, 2011.
[16] D. Reinertsen, *Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, 2009.
[17] J. E. Hannay, H. C. Benestad, and K. Strand, "Earned business value management—see that you deliver value to your customer," *IEEE Software*, vol. 34, no. 4, pp. 58–70, 2017.
[18] M. Haaber and P. Grøhøj, "Benefit points in scrum: A design science study," tech. rep., Dept. of Computer Science, Aalborg University, 2018.
[19] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: improving the design of existing code. addison*. Wesley Longman Publishing Co., Inc., 1999.
[20] A. P. Snow and M. Keil, "The challenge of accurate software project status reporting: a two-stage model incorporating status errors and reporting bias," *IEEE Transactions on Engineering Management*, vol. 49, no. 4, pp. 491–504, 2002.
[21] S. Petter, "If you can't say something nice: Factors contributing to team member silence in distributed software project teams," in *Proceedings of the 2018 ACM SIGMIS Conference on Computers and People Research*, pp. 43–49, 2018.
[22] J. Andrea, "The case of the missing fingerprint: Solve the mystery of successful end-of-projects retrospectives," *Better Software*, pp. 30–36, February 2007.
[23] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.
[24] B. Van Oort, L. Cruz, B. Loni, and A. Van Deursen, ""Project smells" – Experiences in analysing the software quality of ML projects with mllint," in *Proc. 44th IEEE/ACM Int'l Conf. Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 211–220, 2022.
[25] U. Telemaco, T. Oliveira, P. Alencar, and D. Cowan, "A catalogue of agile smells for agility assessment," *IEEE Access*, vol. 8, pp. 79239–79259, 2020.
[26] M. Cohn, "Toward a catalog of scrum smells," 2003.
[27] E. E. Smith, "Concepts and thought," *The psychology of human thought*, vol. 147, 1988.
[28] G. Klein, R. Pliske, B. Crandall, and D. D. Woods, "Problem detection," *Cognition, Technology & Work*, vol. 7, pp. 14–28, 2005.
[29] D. A. Cowan, "Developing a process model of problem recognition," *Academy of Management Review*, vol. 11, no. 4, pp. 763–776, 1986.
[30] H. Mintzberg, D. Raisinghani, and A. Theoret, "The structure of "unstructured" decision processes," *Administrative Science Quarterly*, vol. 21, no. 2, pp. 246–275, 1976.
[31] R. Chow, K. Christoffersen, and D. D. Woods, "A model of communication in support of distributed anomaly response and replanning," *Proc. Human Factors and Ergonomics Society Annual Meeting*, vol. 44, no. 1, pp. 34–37, 2000.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

[32] R. F. Haines, “A breakdown in simultaneous information processing,” in *Presbyopia research: From molecular biology to visual adaptation*, pp. 171–175, Springer, 1991.

[33] A. Mack, “Inattentional blindness: Looking without seeing,” *Current directions in psychological science*, vol. 12, no. 5, pp. 180–184, 2003.

[34] S. S. Tanilkan, “Benefits management – a study of public sector digitalization projects.” https://tinyurl.com/PubSecBM. Accessed: 2024-04-08.

[35] J. Corbin and A. Strauss, *Basics of Qualitative Research*. Sage Publications, 2015.

[36] A. Tjora, *Kvalitative forskningsmetoder i praksis*. Gyldendal Norsk Forlag AS, 2020.

[37] M. Alvesson and K. Sköldberg, *Reflexive Methodology: New Vistas for Qualitative Research*. Sage Publications, 2018.

[38] V. Braun and V. Clarke, “One size fits all? what counts as quality practice in (reflexive) thematic analysis?,” *Qualitative research in psychology*, vol. 18, no. 3, pp. 328–352, 2021.

[39] C. O’Connor and H. Joffe, “Intercoder reliability in qualitative research: Debates and practical guidelines,” *Int’l J. Qualitative Methods*, vol. 19, 2020.

[40] K. K. Holgeid, M. Jørgensen, D. I. K. Sjøberg, and J. Krogstie, “Benefits management in software development: A systematic review of empirical studies,” *IET Software*, vol. 15, 2021.

[41] Direktoratet for Økonomistyring, “Gevinstrealisering – planlegging for å hente ut gevinster av offentlige prosjekter,” guidance to practitioners, Direktoratet for Økonomistyring, 2014.

[42] M. Saunders, P. Lewis, and A. Thornhill, *Research methods for business students*. Pearson Education, 8th ed., 2019.

[43] W. Huang, *The Management of Continuous Product Development: Empirical Research in the Online Game Industry*. Springer Nature, 2022.

[44] S. S. Tanilkan and J. E. Hannay, “Projects vs continuous product development – does it affect benefits realization?,” in *Proc. Int’l Conf. Advances and Trends in Software Engineering (SOFTENG)*, pp. 20–25, 2023.

[45] S. S. Gautam and V. Singh, “The state-of-the-art in software development effort estimation,” *J Software: Evolution and Process*, vol. 30, no. 12, 2018.

[46] H. A. Simon, *The sciences of the artificial*. MIT press, 1996.

[47] G. Gigerenzer and P. M. Todd, *Simple heuristics that make us smart*. Oxford University Press, USA, 1999.

[48] R. M. Hogarth, *Educating intuition*. University of Chicago Press, 2001.

[49] F. P. Brooks Jr, “The mythical man-month (anniversary ed.),” 1995.

[50] T. D. Cook and D. T. Campbell, *Quasi-Experimention: Design & Analysis Issues For Field Settings*. Houghton Wifflin Co., 1979.

[51] W. M. Trochim and J. P. Donnelly, *Research methods knowledge base*, vol. 2. Atomic Dog Pub. Macmillan Publishing Company, New York, 2001.

[52] R. K. Yin, *Case study research: Design and methods, volume 5 of Applied Social Research Methods Series*. Sage, 3rd ed., 2003.

[53] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, “The evolution and impact of code smells: A case study of two open source systems,” in *2009 3rd international symposium on empirical software engineering and measurement*, pp. 390–400, IEEE, 2009.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Software Bug Prediction Based on Semi-definite Logistic Regression Model

Tadashi Dohi, Jingchi Wu, and Hiroyuki Okamura

Graduate School of Advanced Science and Engineering, Hiroshima University
Higashi-Hiroshima 739-8527, Japan
email: {dohi, d220580, okamu}@hiroshima-u.ac.jp

***Abstract*—In software bug prediction to identify bug-prone modules, several machine learning techniques have been used in past. However, it has been known that almost all of them were not *explainable* and could not be applied to the program understanding, because the contributions of software metrics were unclear in such black box techniques. In this article, we aim at overcoming the problems in an explainable logistic regression model, called *multicollinearity* and *interaction*, and apply the semi-definite logistic regression model to identify software bug-prone modules. More specifically, we use three actual software development project data sets to evaluate the F-score as well as precision and recall, and compare our semi-definite logistic regression model with the classical logistic one, in terms of the predictive performance of software bug-prone modules. It is shown that our semi-definite logistic regression model involves the common logistic regression model as a special case and can improve the predictive performances on the F-score.**

Keywords-software bug prediction; bug-prone module; logistic regressions; semi-definite programming; discrimination problem; F-score.

I. Introduction

In testing and maintenance phases of software development, identification of software bug-prone modules containing bugs is crucial for both localizing software bugs on a computer program and optimizing the software test process. Since this problem is formulated as a typical discrimination problem to identify software bug-prone modules, several machine learning techniques have been used in past, where the underlying data are the binary data to denote whether each module is bug-prone (1) or not (0), and the features called software metrics to characterize the quality attributes of each module, such as the module's size and program complexity. Various discrimination and data mining techniques, including logistic regressions [2] [14], support vector machines [4], naive Bayes [15], Bayesian networks [3] [16], random forest [5], multilayer perceptron neural networks [6], convolutional neural networks [1], and spam filtering technique [13], among others [12] [17], have been directly used to identify software bug-prone modules. For the recent survey on software bug prediction, see Li et al. [11].

However, it has been known that almost all of them were not *explainable* and could not be applied to the program understanding, because the contributions of software metrics were unclear in such black box techniques. In fact, through a careful analysis on the contribution of each software metric in the bug prediction, it would be possible to improve the test efficiency by localizing software bugs from the code metrics such as the number of lines of code, cyclomatic numbers, the number of operators measured in each module development. In the view point of program understanding, it is quite important to investigate the relationship between bug-prone modules and explanatory variables (features), and to infer the presence/absence of software bugs in each module. If there is a clear causal relationship with the explanatory variables in an explainable method as logistic regression models, we may design the test cases efficiently according to the contribution of the metrics.

Unfortunately, it should be noted that the classical logistic regression model could not provide satisfactory bug-prediction results in terms of predictive performances [2] [14], compared to the typical deep machine learning techniques. Even for the explainable models, we need to carefully check not only the independence between explanatory variables but also *multicollinearity* and *interaction* in the regression-based approach. Konno et al. [8] pointed out in the problem of estimating bankruptcy probability from financial metrics in companies that the logistic regression model used conventionally deals with the metrics that have a monotonic relationship, where the bankruptcy probability increases (decreases) as the values of the financial metrics increase (decrease). In the financial bankruptcy problem, it is implicitly assumed that there is no interaction on effects of each explanatory variable as a financial metric on the bankruptcy probability, but generally, the impact of explanatory variables on the bankruptcy probability may vary depending on the size of the explanatory variable.

Although the conventional logistic regression model, being simple and low in computational cost, is frequently applied to many real-world discrimination problems by devising the selection of explanatory variables and the classification of the dependent variable groups, there are theoretical and empirical limitations on the explainable logistic regression model. Konno et al. [8] proposed a semi-definite logistic regression model and attempted to solve large-scale semi-definite logistic regression problems in real-time [10] by applying a few optimization techniques such as the cutting-plane method [7] and the two-stage method [9].

In this article, we aim at overcoming the problems in a classical logistic regression model for software bug prediction, and apply the semi-definite logistic regression model to identify software bug-prone modules. More specifically, we use three actual software development project data sets, and evaluate the predictive performance on F-score, as well as precision and recall. We compare our semi-definite logistic regression model with the classical logistic regression model

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

in the context of software bug prediction. It is shown that our semi-definite logistic regression model involves the common logistic regression model as a special case and can improve the predictive performances on the F-score in the software bug prediction.

The remaining part of this article is organized as follows. In Section II, we describe a software bug prediction problem by means of the logistic regression model. Section III formulates a semi-definite logistic regression and summarizes a variable selection method. Section IV is devoted to numerical experiments, where the underlying data sets are given and the predictive performances between two logistic regression models are compared. Finally the article is concluded with some remarks in Section V.

II. Software Bug Prediction

Suppose that there are N software modules in the module testing. Let $\boldsymbol{x_i} = (x_{i1}, x_{i2}, \ldots, x_{in})$ denote the feature vector of the i $(= 1, 2, ..., N)$-th software module, where n types of features, called software metrics, are available in the software development. Define the probability that the i-th module contains any software bug by $f(\boldsymbol{x}_i)$ as a function of the feature vector $\boldsymbol{x}_i$, where $f(\cdot)$ denotes a nonlinear function. In the nonlinear equation $y_i = f(\boldsymbol{x}_i)$ for a given y_i, it is not possible to directly observe the probability of a module containing software bugs in advance, so that the information about the presence/absence of software bugs in each module is used post-hoc. Define the binary random variable Y_i with the realization y_i in the following:

$$Y_i = \begin{cases} 1, & \text{if module } i \text{ contains software bugs,} \\ 0, & \text{if module } i \text{ does not contain software bugs.} \end{cases} \quad (1)$$

There are various methods to formulate the above discrimination problem. Among them, the logistic regression model is easy to understand in terms of the formulation and the low computation cost. In the logistic regression model, the regression function $f(\boldsymbol{x}_i)$ is given by

$$f(\boldsymbol{x}_i) = \frac{\exp(\boldsymbol{Z}_i)}{1 + \exp(\boldsymbol{Z}_i)}, \quad (2)$$

where $\exp(\boldsymbol{Z}_i) = \boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0$ denotes the random variables representing the tendency of bug presence, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_n)$ is the regression coefficient vector, β_0 is a scalar constant, and T is the transpose. Since the bug-prone probability $f(\boldsymbol{x}_i) = f(\boldsymbol{x}_i; \boldsymbol{\beta}, \beta_0)$ is given by a function of $\boldsymbol{x}_i$, it turns out that the dependent variable becomes a monotonic function with respect to each component of $\boldsymbol{x}_i$.

Once the binary data and the software metric data $(y_i, \boldsymbol{x}_i)$ $(i = 1, 2, \ldots, N)$ are given, the log likelihood function is obtained as

$$\ln L(\boldsymbol{\beta}, \beta_0) = \sum_{i=1}^{N} \Big\{ y_i \ln f(\boldsymbol{x}_i; \boldsymbol{\beta}, \beta_0) + (1 - y_i) \ln(1 - f(\boldsymbol{x}_i; \boldsymbol{\beta}, \beta_0)) \Big\}. \quad (3)$$

Then the problem is to seek the maximum likelihood estimate $(\tilde{\boldsymbol{\beta}}, \tilde{\beta}_0) = \operatorname{argmax} \ln L(\boldsymbol{\beta}, \beta_0)$.

As mentioned in Section I, it is known that the logistic regression has several limitations. First, it is assumed that the larger (smaller) the value of each explanatory variable, the larger (smaller) the predicted probability y_i becomes. Second, it is assumed that the elements of each explanatory variable are independent of each other, and there is no interaction effects of each explanatory variable on the bug-prone probability. In the actual software bug prediction, for instance, an increase in the number of comment lines on a program is implicitly assumed as one of the software metrics increases the bug-prone probability. However, this property may not always hold, because insertion of a certain type of detailed comments may increase the understandability of the program, and may reduce the bug-prone probability efficiently. Furthermore, among many software metrics, the relationship between the lines of code and the total number of operators on the program may be unlikely to be independent. This is because an increase in the lines of codes may naturally tend to increase the total number of operators.

III. Semi-definite Logistic Regression Approach

The semi-definite logistic regression model was proposed in the reference [8]. For the real symmetric matrix $\boldsymbol{B} \in \mathcal{R}^{n \times n}$, the bug-prone probability $f(\boldsymbol{x}_i) = f(\boldsymbol{x}_i; \boldsymbol{B}, \beta_0)$ $(i = 1, 2, \ldots, N)$ is also defined by Eq.(2), where

$$\begin{aligned} \boldsymbol{Z}_i &= \boldsymbol{x}_i^T \boldsymbol{B} \boldsymbol{x}_i + \boldsymbol{\beta}^T \boldsymbol{x}_i + \beta_0, && (4) \\ \boldsymbol{B} &= \boldsymbol{B}^T. && (5) \end{aligned}$$

In our semi-definite logistic regression model, it should be noted that $\mathbf{Z}_i$ is a quadratic form of $\boldsymbol{x}_i$. Hence, it is possible to incorporate non-monotonicity and interaction effects between explanatory variables.

Several optimization algorithms have already been proposed to solve the semi-definite logistic regression problems [7] [8] [9] [10]. As the problem size in dealing with the software bug prediction increases, it tends to be difficult to solve the maximum likelihood estimation problem. Fortunately, since our problem size is relatively small comparing with the financial bankruptcy problem, we can handle the maximum likelihood estimation for the semi-definite logistic regression model, by applying the quadratic programming algorithm implemented in the statistical software, R, without using the cutting-plane method [7] and the two-stage method [9]. On the other hand, it should be emphasized that all the explanatory variables may not always be useful for discriminating the software bug-prone modules. Generally, it is important to select a small number of useful explanatory features from all available ones. In this article, we use the well-known Akaike information criterion (AIC):

$$\text{AIC} = -2 \ln L(\tilde{\boldsymbol{B}}, \tilde{\beta}_0) + 2 \times (\text{number of free parameters in the model}) \quad (6)$$

TABLE I
DATA SETS.

	No. modules (N)	Bug inclusion rate	No. metrics (n)
jm1	7782	21.05%	21
pc1	705	8.65%	37
cm1	327	12.80%	37

for the maximum likelihood estimate $(\tilde{\boldsymbol{B}}, \tilde{\beta}_0)$ to determine the explanatory variables employed in the analysis. A smaller AIC indicates better model fit.

In general, there are two variable selection methods; the variable reduction method and variable increase method under the AIC criterion. We apply the variable reduction method to the semi-definite logistic regression model. The main reason why the variable increase method is not used here is that the number of arbitrary parameters in the semi-definite logistic regression model becomes large in the order of squares, and results in enormous computation cost, so our semi-definite logistic regression model penalizes by applying the variable increase method. Concretely, we estimate the parameters (regression coefficients) using all software metrics first, and then remove unnecessary feature one by one so as to minimize the AIC. We repeat this procedure again and again until the AIC value can be reduced no longer. In summary, we describe the procedure for our variable reduction method as follows.

Step 1: For $(y_i, \boldsymbol{x}_i)$ $(i = 1, 2, \ldots, m)$, where m is the number of training data, apply a semi-definite quadratic programming to the semi-definite logistic regression model, derive the maximum likelihood estimate $(\tilde{\boldsymbol{\beta}}, \tilde{\beta}_0)$, and calculate the AIC.

Step 2: Apply the variable reduction method, remove one unnecessary software metric with minimum regression coefficient and set $m - 1 \to m$.

Step 3: Go to Step 1 and calculate the AIC' with the updated m.

Step 4: If AIC $>$ AIC', then Go to Step 1, otherwise, and set $m \to m - 1$ and Stop the procedure.

IV. NUMERICAL EXPERIMENTS

A. Data Sets

Data sets used here are from NASA's software development projects, namely, jm1, pc1, and cm1. The number of modules, defect density, and the number of software metrics used in the experiments for each data set are shown in Tables I to III.[1]

B. Predictive Performances

We compare the predictive performances of four bug prediction models; the standard logistic regression model, the semi-definite logistic regression model, and respective models refined by variable reduction. Hereafter, we denote the standard logistic regression as Logit-11, the standard logistic regression with variable reduction method Logit-12, the semi-definite logistic regression model Logit-21, and the semi-definite logistic regression model with variable reduction method Logit-22. In our experiments, the data sets are randomly split into the training data and the validation data. Especially, three cases with different training data sizes; 25%, 50%, and 75% of the whole data, are considered. In each case, the remaining 75%, 50% and 25% data sets are used for validation/prediction.

[1]The data sets were reported in NASA/WVU IV &V Facility, Metrics Data Program. http://mdp.ivv.nasa.gov/.

TABLE II
SOFTWARE METRICS IN JM1.

	Software metrics
x_{i1}	LOC BLANK
x_{i2}	BRANCH COUNT
x_{i3}	LOC CODE AND COMMENT
x_{i4}	LOC COMMENTS
x_{i5}	CYCLOMATIC COMPLEXITY
x_{i6}	DESIGN COMPLEXITY
x_{i7}	ESSENTIAL COMPLEXITY
x_{i8}	LOC EXECUTABLE
x_{i9}	HALSTEAD CONTENT
x_{i10}	HALSTEAD DIFFICULTY
x_{i11}	HALSTEAD EFFORT
x_{i12}	HALSTEAD ERROR EST
x_{i13}	HALSTEAD LENGTH
x_{i14}	HALSTEAD LEVEL
x_{i15}	HALSTEAD PROG TIME
x_{i16}	HALSTEAD VOLUME
x_{i17}	NUM OPERANDS
x_{i18}	NUM OPERATORS
x_{i19}	NUM UNIQUE OPERANDS
x_{i20}	NUM UNIQUE OPERATORS
x_{i21}	LOC TOTAL
x_{i22}	DEFECTIVE

To evaluate the bug-prone probability, we apply the F-score which is a harmonic mean of *precision* and *recall*, where precision indicates a proportion of correct results in the prediction, and recall is a proportion of how much of the correct answers could be predicted. That is, we have

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}, \\ \text{Recall} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}, \end{aligned} \tag{7}$$

where True Positive is the number of data that could be correctly predicted to contain software bugs, False Positive is the number of data incorrectly predicted to contain software bugs, and False Negative is the number of data incorrectly predicted to contain no software bugs. Then F-score is defined by

$$F\text{-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{8}$$

Note that F-score is given by a real number between 0 and 1, and can be interpreted such that the higher the value of F-score the higher the predictive performance of the model.

In order to predict the bug-proneness of each software module, it is necessary to choose a *threshold* for judgement of bug proneness. In our experiments we set five threshold values; 0.3, 0.4, 0.5, 0.6, and 0.7. If the bug-prone probability is greater than a given threshold, then the resulting software

TABLE III
SOFTWARE METRICS IN PC1 AND CM1.

	Software metrics
x_{i1}	LOC BLANK
x_{i2}	BRANCH COUNT
x_{i3}	CALL PAIRS
x_{i4}	LOC CODE AND COMMENT
x_{i5}	LOC COMMENTS
x_{i6}	CONDITION COUNT
x_{i7}	CYCLOMATIC COMPLEXITY
x_{i8}	CYCLOMATIC DENSITY
x_{i9}	DECISION COUNT
x_{i10}	DECISION DENSITY
x_{i11}	DESIGN COMPLEXITY
x_{i12}	DESIGN DENSITY
x_{i13}	EDGE COUNT
x_{i14}	ESSENTIAL COMPLEXITY
x_{i15}	ESSENTIAL DENSITY
x_{i16}	LOC EXECUTABLE
x_{i17}	PARAMETER COUNT
x_{i18}	HALSTEAD CONTENT
x_{i19}	HALSTEAD DIFFICULTY
x_{i20}	HALSTEAD EFFORT
x_{i21}	HALSTEAD ERROR EST
x_{i22}	HALSTEAD LENGTH
x_{i23}	HALSTEAD LEVEL
x_{i24}	HALSTEAD PROG TIME
x_{i25}	HALSTEAD VOLUME
x_{i26}	MAINTENANCE SEVERITY
x_{i27}	MODIFIED CONDITION COUNT
x_{i28}	MULTIPLE CONDITION COUNT
x_{i29}	NODE COUNT
x_{i30}	NORMALIZED CYLOMATIC COMPLEXITY
x_{i31}	NUM OPERANDS
x_{i32}	NUM OPERATORS
x_{i33}	NUM UNIQUE OPERANDS
x_{i34}	NUM UNIQUE OPERATORS
x_{i35}	NUMBER OF LINES
x_{i36}	PERCENT COMMENTS
x_{i37}	LOC TOTAL
x_{i38}	DEFECTIVE

module is judged to contain software bugs. The prediction results obtained in the experiments are shown in Tables VI to XII. In these tables, the largest value in each table is denoted by a double underline, and the largest value in each threshold level is singly underlined.

In the data set, jm1, our semi-definite logistic model with variable reduction could show the highest F-scores when estimating the parameters using 75% of the training data. Furthermore, in Tables IV to VI, the standard logistic regression showed the highest F-score with 25% of the training data, but our semi-definite logistic regression model with variable reduction could give the higher performances on F-score when estimating parameters using 50% and 75% of the training data.

In the data set, pc1, it is observed that our semi-definite logistic regression model with variable reduction provided the highest F-score with 75% of the training data. Also, in all cases of models in Tables VII to IX, when estimating the model parameters with 25%, 50% and 75% of the training data, the semi-definite logistic regression model with variable reduction could give the highest F-score evidently.

Finally, in the data set, m1, it can be seen that our semi-definite logistic regression model with variable reduction gave the highest F-score with 25% of the training data. In Tables X to XII, the standard logistic regression model gave the highest F-score with 50% of the training data, but our semi-definite logistic regression model with variable reduction could show the highest predictive performances when estimating parameters using 25% and 75% of the training data.

In comparison between the conventional logistic regression model and the semi-definite regression model, it is found that our novel approach could not always outperform the classical one. However, our experimental results showed that in most cases across all data sets, the semi-definite logistic regression models could exhibit higher F-score than the conventional logistic regression models. As shown in Tables IV to XII, even though the predictive performances of our semi-definite

TABLE IV
JM1 (25% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.445	0.307	0.363
	Logit-12	0.451	0.298	0.357
	Logit-21	0.305	0.296	0.291
	Logit-22	0.443	0.304	0.360
0.4	Logit-11	0.508	0.186	0.271
	Logit-12	0.514	0.176	0.261
	Logit-21	0.318	0.288	0.292
	Logit-22	0.510	0.182	0.267
0.5	Logit-11	0.550	0.111	0.184
	Logit-12	0.546	0.108	0.179
	Logit-21	0.330	0.287	0.292
	Logit-22	0.539	0.113	0.187
0.6	Logit-11	0.592	0.070	0.125
	Logit-12	0.587	0.072	0.127
	Logit-21	0.325	0.288	0.290
	Logit-22	0.605	0.075	0.132
0.7	Logit-11	0.648	0.046	0.086
	Logit-12	0.640	0.046	0.086
	Logit-21	0.317	0.286	0.285
	Logit-22	0.652	0.050	0.093

TABLE V
JM1 (50% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.453	0.309	0.367
	Logit-12	0.462	0.307	0.368
	Logit-21	0.330	0.319	0.311
	Logit-22	0.432	0.326	0.371
0.4	Logit-11	0.513	0.182	0.268
	Logit-12	0.518	0.176	0.262
	Logit-21	0.336	0.279	0.289
	Logit-22	0.509	0.183	0.268
0.5	Logit-11	0.559	0.105	0.176
	Logit-12	0.558	0.105	0.176
	Logit-21	0.325	0.288	0.281
	Logit-22	0.557	0.115	0.190
0.6	Logit-11	0.609	0.069	0.123
	Logit-12	0.610	0.066	0.119
	Logit-21	0.374	0.240	0.261
	Logit-22	0.631	0.071	0.128
0.7	Logit-11	0.685	0.043	0.082
	Logit-12	0.682	0.044	0.083
	Logit-21	0.356	0.276	0.272
	Logit-22	0.646	0.046	0.086

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE VI
JM1 (75% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.459	0.313	0.372
	Logit-12	0.457	0.310	0.369
	Logit-21	0.343	0.325	0.315
	Logit-22	0.453	0.331	0.382
0.4	Logit-11	0.514	0.179	0.265
	Logit-12	0.520	0.177	0.264
	Logit-21	0.337	0.300	0.294
	Logit-22	0.515	0.187	0.274
0.5	Logit-11	0.563	0.104	0.175
	Logit-12	0.570	0.104	0.175
	Logit-21	0.346	0.288	0.284
	Logit-22	0.572	0.103	0.175
0.6	Logit-11	0.632	0.066	0.119
	Logit-12	0.630	0.067	0.120
	Logit-21	0.364	0.294	0.284
	Logit-22	0.655	0.066	0.120
0.7	Logit-11	0.686	0.042	0.080
	Logit-12	0.691	0.043	0.081
	Logit-21	0.351	0.260	0.260
	Logit-22	0.731	0.044	0.082

TABLE VII
PC1 (25% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.226	0.348	0.269
	Logit-12	0.239	0.342	0.278
	Logit-21	0.113	0.494	0.183
	Logit-22	0.286	0.462	0.351
0.4	Logit-11	0.232	0.349	0.273
	Logit-12	0.243	0.327	0.271
	Logit-21	0.106	0.489	0.173
	Logit-22	0.293	0.320	0.298
0.5	Logit-11	0.235	0.357	0.277
	Logit-12	0.248	0.334	0.277
	Logit-21	0.110	0.477	0.178
	Logit-22	0.309	0.313	0.305
0.6	Logit-11	0.237	0.351	0.276
	Logit-12	0.246	0.320	0.268
	Logit-21	0.112	0.518	0.182
	Logit-22	0.253	0.217	0.221
0.7	Logit-11	0.232	0.336	0.269
	Logit-12	0.238	0.325	0.268
	Logit-21	0.106	0.485	0.173
	Logit-22	0.319	0.242	0.264

TABLE VIII
PC1 (50% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.317	0.355	0.327
	Logit-12	0.303	0.338	0.312
	Logit-21	0.088	0.490	0.149
	Logit-22	0.328	0.298	0.304
0.4	Logit-11	0.337	0.325	0.323
	Logit-12	0.326	0.280	0.294
	Logit-21	0.086	0.486	0.146
	Logit-22	0.350	0.234	0.270
0.5	Logit-11	0.364	0.265	0.299
	Logit-12	0.369	0.243	0.280
	Logit-21	0.088	0.500	0.149
	Logit-22	0.489	0.267	0.334
0.6	Logit-11	0.366	0.248	0.286
	Logit-12	0.374	0.220	0.266
	Logit-21	0.088	0.490	0.149
	Logit-22	0.366	0.288	0.305
0.7	Logit-11	0.391	0.198	0.253
	Logit-12	0.421	0.207	0.266
	Logit-21	0.092	0.499	0.154
	Logit-22	0.461	0.187	0.239

TABLE IX
PC1 (75% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.346	0.356	0.342
	Logit-12	0.312	0.317	0.306
	Logit-21	0.853	0.483	0.144
	Logit-22	0.391	0.349	0.357
0.4	Logit-11	0.395	0.293	0.325
	Logit-12	0.388	0.267	0.305
	Logit-21	0.091	0.501	0.153
	Logit-22	0.412	0.329	0.353
0.5	Logit-11	0.439	0.243	0.301
	Logit-12	0.404	0.196	0.260
	Logit-21	0.092	0.534	0.156
	Logit-22	0.385	0.405	0.374
0.6	Logit-11	0.435	0.211	0.272
	Logit-12	0.422	0.167	0.242
	Logit-21	0.088	0.500	0.148
	Logit-22	0.417	0.245	0.294
0.7	Logit-11	0.479	0.190	0.260
	Logit-12	0.486	0.160	0.241
	Logit-21	0.094	0.52	0.158
	Logit-22	0.521	0.250	0.301

logistic regression models without variable reduction method were rather low, applying the variable reduction could improve the predictive performances. In a few cases, it can be found that the conventional logistic regression models showed better predictive performances than the semi-definite logistic regression models. However, since the semi-definite logistic regression model includes the logistic regression model as a special case, the predictive performances of the semi-definite logistic regression models are never inferior to those of the common logistic regression models.

V. CONCLUSIONS

In this article, we have proposed a novel and explainable software bug-prediction model based on the semi-definite logistic model and compared the applicability in predicting the bug-prone module. In the past literature, almost all works have focused on only the predictive performances including F-score and attempted to apply several machine learning techniques in the software bug-prediction. However, since almost all machine learning techniques did not provide the feedback information on the dependence between the software metrics employed in the analysis and the bug-proneness, more sophisticated explainable bug-prediction methods have been demanded. In our numerical experiments, we have shown that our semi-definite logistic model could show the potential applicability in software bug prediction. By checking the regression coefficients with respect to a combination of software metrics, it would be possible to analyze the dependence in software bug-prone probability.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE X
CM1 (25% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.200	0.360	0.253
	Logit-12	0.234	0.336	0.269
	Logit-21	0.137	0.467	0.211
	Logit-22	0.167	0.227	0.187
0.4	Logit-11	0.206	0.367	0.260
	Logit-12	0.226	0.327	0.260
	Logit-21	0.133	0.466	0.205
	Logit-22	0.296	0.456	0.354
0.5	Logit-11	0.204	0.355	0.254
	Logit-12	0.223	0.314	0.253
	Logit-21	0.139	0.488	0.215
	Logit-22	0.252	0.321	0.275
0.6	Logit-11	0.211	0.352	0.260
	Logit-12	0.220	0.316	0.252
	Logit-21	0.141	0.505	0.220
	Logit-22	0.247	0.260	0.247
0.7	Logit-11	0.202	0.359	0.254
	Logit-12	0.214	0.311	0.247
	Logit-21	0.136	0.452	0.207
	Logit-22	0.254	0.220	0.225

TABLE XI
CM1 (50% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.269	0.349	0.295
	Logit-12	0.261	0.343	0.288
	Logit-21	0.135	0.479	0.209
	Logit-22	0.258	0.309	0.274
0.4	Logit-11	0.268	0.327	0.288
	Logit-12	0.273	0.306	0.280
	Logit-21	0.139	0.502	0.216
	Logit-22	0.392	0.244	0.292
0.5	Logit-11	0.264	0.307	0.276
	Logit-12	0.273	0.263	0.258
	Logit-21	0.140	0.489	0.216
	Logit-22	0.277	0.236	0.246
0.6	Logit-11	0.269	0.251	0.251
	Logit-12	0.284	0.224	0.239
	Logit-21	0.133	0.464	0.206
	Logit-22	0.315	0.250	0.265
0.7	Logit-11	0.283	0.254	0.258
	Logit-12	0.287	0.221	0.239
	Logit-21	0.133	0.471	0.206
	Logit-22	0.377	0.186	0.242

TABLE XII
CM1 (75% TRAINING DATA).

Threshold	Model Type	Precision	Recall	F-score
0.3	Logit-11	0.287	0.331	0.295
	Logit-12	0.276	0.326	0.295
	Logit-21	0.127	0.472	0.196
	Logit-22	0.312	0.396	0.337
0.4	Logit-11	0.338	0.305	0.299
	Logit-12	0.311	0.272	0.286
	Logit-21	0.136	0.534	0.213
	Logit-22	0.348	0.322	0.336
0.5	Logit-11	0.341	0.279	0.288
	Logit-12	0.301	0.188	0.235
	Logit-21	0.135	0.513	0.210
	Logit-22	0.418	0.212	0.277
0.6	Logit-11	0.358	0.200	0.239
	Logit-12	0.322	0.160	0.243
	Logit-21	0.133	0.492	0.205
	Logit-22	0.364	0.190	0.263
0.7	Logit-11	0.329	0.152	0.194
	Logit-12	0.342	0.091	0.215
	Logit-21	0.136	0.504	0.212
	Logit-22	0.347	0.123	0.220

In future, we will compare the semi-definite logistic regression approach with several deep learning methods in large-scaled experiments and explore the potential to use it in software bug prediction problems.

REFERENCES

[1] S. Balasubramaniam, and S. G. Gollagi, "Software defect prediction via optimal trained convolutional neural network," *Advances in Engineering Software*, vol. 169, p. 103138, 2022.

[2] L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Transactions on Software Engineering*, vol. 28, pp. 706-–720, 2002.

[3] G. Denaro, and M. Pezze, "An empirical evaluation of fault-proneness models" *Proceedings of The 24th International Conference on Software Engineering (ICSE-2002)*, pp. 241–251, 2002.

[4] K. O. Elish, and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, pp. 649–660, 2008.

[5] L. Guo, Y. Ma, B. Cukic, H. Singh, "Robust prediction of fault-proneness by random forests," *Proceedings The 15th International Symposium on Software Reliability Engineering (ISSRE-2004)*, pp. 417–428, 2004.

[6] C. Jin, and S. W. Jin, "Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization," *Applied Soft Computing*, vol. 35, pp. 717–725, 2015.

[7] H. Konno, N. Kawadai, and H. Yuy, "Cutting plane algorithms for nonlinear semi-definite programming problems with applications," *Journal of Global Optimization*, vol. 25, pp. 141–155, 2003.

[8] H. Konno, N. Kawadai, and D. Wu, "Estimation of failure probability using semi-definite logit model," *Computational Management Science*, vol. 1, pp. 59—73, 2003.

[9] H. Konno, N. Kawadai, and H. Shimode, "A two step algorithm for solving a large scale semi-definite logit model," *Optimization Letters*, vol. 1, pp 329–340, 2007.

[10] H. Konno, S. Kameda, and N. Kawadai, "Solving a large scale semi-definite logit model," *Computational Management Science*, vol. 7, pp. 111—120, 2010.

[11] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, 161–175, 2018.

[12] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, pp. 2—13, 2007.

[13] O. Mizuno, and T. Kikuno, "Training on errors experiment to detect fault-prone software modules by spam filter," *Proceedings of The 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE2007)*, pp. 405–414, 2007.

[14] N. Ohlsson, and H. Alberg, "Predicting fault-prone software modules in telephone switches," *IEEE Transactions on Software Engineering*, vol. 22, pp. 886–894, 1996.

[15] S. K. Pandey, R. B. Mishra, and A. K. Triphathi, "Software bug prediction prototype using Bayesian network classifier: A comprehensive model," *Procedia Computer Science*, vol.1 32, pp. 1412–1421, 2018.

[16] E. Perez-Minana, and J. J. Gras, "Improving fault prediction using Bayesian networks for the development of embedded software applications," *Software Testing, Verification and Reliability*, vol. 16, pp. 157–174, 2006.

[17] H. Tong, B. Liu, and S. Wang, "Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning," *Information and Software Technology*, vol. 96, pp. 94–111, 2018.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

On Reducibility of Developer-Written Unit Tests in C#

Arpit Christi
School of Computing
Weber State University
Ogden, UT, USA
email: arpitchristi@weber.edu

David Weber
Northrop Grumman
Roy, UT, USA
email: ein.nuff@gmail.com

***Abstract*—Test case reduction is employed to help developer isolate and locate faults in complex software systems if the failing test is complex and contains a lot of non failure-inducing elements. Reduced test still contains the exact same failure-inducing component as the original test. Thus, the smaller test assists developers by focusing their attention on the faulty aspect of the program quickly. Researchers have focused their attention on improvement of the test reduction process. The outcome of the test reduction is not studied thoroughly. We study the result of the test case reduction when algorithms like Delta Debugging are used to minimize the tests. We evaluate (1) test reduction size based on the category of a statement and (2) effect of the category of a statement on reduction. The developer-written tests are just like any other code - containing the same structures, elements, and components as the rest of the program. If we consider the program as an abstract syntax tree, our results demonstrate that (1) leaf nodes are removed in larger quantity and (2) leaf nodes have higher probability of removal.**

Keywords-program debugging; software testing; test case reduction.

I. Introduction

Program debugging is often tedious, time-consuming and challenging. Most of the developer time is spent on locating and isolating the fault. When utilizing a failed test to debug and fix a fault, developers may need to observe and transit through aspects of test/program that are non failure-inducing, resulting into developer time disuse. If the failed test is minimized while keeping the failure-inducing input, the developer may need to rummage through lesser program elements promoting optimal use of developer time and hence quicker debugging. Being orthogonal to aiding developer in debugging, test reduction is found useful in Automatic Fault Localization (FL) - a process to automatically locate the bug in a faulty program. The entire reduced test or the byproducts of test reduction are found to be useful in Automatic FL [1] [2].

Delta Debugging (DD) and Hierarchical Delta Debugging algorithms (HDD) were proposed to minimize failure-inducing tests [3] [4]. DD algorithm is optimal for test inputs that are flat structures like array, lists or sets. If you consider a test written in program like C#, in order to apply DD, one needs to consider test to be an array of lines, an array of characters or words. As it does not consider the tree like structure, interdependence between nodes and other such details, DD is not optimal for structures like HTML files and programs. HDD can process such tests better as it exploits the underlying tree structures to its advantage. Both algorithms are essentially a greedy search to systematically and incrementally find a smaller test until a minimal test is reached.

Many recent algorithms and implementations to minimize failing tests still rely on the DD and HDD algorithms as the foundation [5]–[11]. These tools and techniques mainly attempt to improve the test reduction process to efficiently and accurately reduce tests. Though test reduction process have been studied for a while, (1) the outcome of the test reduction and (2) the entities that were reduced as part of the reduction process have not been studied thoroughly.

Based on the type of test (program, html, xml, text files etc.), the reduction outcome and the reduced entities can be different. If we only consider tests written as a program in a particular programming language, we can define and study the reduction outcome and the reduced entities by considering the outcome as a reduced program and the entities as programming components like - program statements, program lines, nodes of Abstract Syntax Tree (AST) of the program. For this work, we consider reduced entities as nodes of AST. An example of the test is in Figure 1 and the corresponding AST is in Figure 4. We further categorize each statement node of the AST into non-tree statements and tree statements as explained in detail in Section IV.

We focus on test reduction for tests written in C# programming language. We study the reduction outcome and the reduced entities for 30 real world bugs in 5 open source C# projects. Based on our study, we provide the following insights into test reduction for C# tests.

1) The number of non-tree statements reduced are significantly larger than the number of tree statements reduced.
2) The chance of a non-tree statement removal is slightly higher than the chance of a tree statement removal.

The ReduSharptor tool that we used for test reduction is publicly available on GitHub [12].

The rest of the paper is organized as follows. In Section II, we discuss the related work. In section III, we discuss the background for our work and motivate the need for the study. In Section IV, we discuss the terminology and definitions based on the outcome of the reduction process to determine catogory of program statements. Section V depicts the test subjects, the experiments and the results. We mention how we mitigate the threats to validity in Section VI. Finally, Section VII concludes the paper by discussing the results and the future direction.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

II. Related Work

DD finds minimal failure inducing input by employing a greedy search that removes components that are unnecessary for triggering the bug [3]. HDD improves on DD for hierarchical inputs like xml, html, and programs. HDD achieves the improvement by considering the AST representation of hierarchical inputs [4].

Researchers proposed many recent algorithms, tools and techniques. They (1) improve over the original DD and HDD algorithms or (2) retrofit DD/HDD implementation for specific situation or programming languages. CReduce, Generalized Tree Reduction (GTR), Picireny, Perses, DDSET, Observational Based Slicing (ORBS), Reduktor, ProbDD, and ReduSharptor are a few such attempts [5]–[11], [13]–[15].

Christi et al. combined HDD with statement deletion mutation to propose Test-Based Software Minimization (TBSM) that reduces programs instead of tests to build a minimal resource adaptive software while sacrificing low-priority but resource-consuming functionality [16]. To improve the performance of TBSM, they study the outcome of the program reduction and the entities that were reduced. Based on that, they proposed multiple heuristics to improve the performance of TBMS [17].

Perses reduces the removal entities by only considering syntactically valid variants [6]. Wang et al. propose probabilistic delta debugging approach that uses AST, historic test results and syntactic relationships to assign probability to each element for removal [10] [11]. The approach showed significant improvement in performance because it reduces the number of entities under consideration for removal. We study the reduction process only in terms of the location of an entity within the program. Also, we use a different programming language and a different dataset to study the reduction process.

```
[Fact]
public void Foo(Test)
{
1  Math m = new Math();
2  int sum1 = m.Add(3,4)
   // Assumption: Add method is written in a
   // peculiar way and cannot add 3 and 4
   // correctly.
3  Assert.Equal(sum1,7); //suppose sum1 is 8,
    hence the test is failing here.
4   if(true){
5       int sum2 = m.Add(-2,-3)
6      Assert.Equal(sum2,-5); // This assert
    passes.
7     }
}
```

Figure 1. `original` test, Line 3 is the failing statement.

```
[Fact]
public void Foo(Test) //The minimal reduced
    test
{
1  Math m = new Math();
2  int sum1 = m.Add(3,4)
   // Same assumption as the original test.
3  Assert.Equal(sum1,7); //suppose sum1 is 8,
    hence the test is failing here.

}
```

Figure 2. `minimal` test, All statements from line 4 in Figure 1 are removed.

III. Background and Motivation

If we can categorize test program statements into distinct categories and establish an empirical link between the category of a statement and its probability of removal, we can preemptively choose to process or ignore certain types of statements in the test reduction process. To this end, the outcome of test reduction and reduced entities need to be studied further. Studying the relationship in detail can help to propose efficient approaches like perses and probabilistic delta debugging [6], [11]. It may help propose heuristics as proposed by Christi et al. to reduce the search space for the reduction [17]. So far, such categorization is not clearly established.

We will only consider tests written in C# programming language. When DD, HDD or any other techniques are applied on a test for test reduction, it produces a minimal test.

The test can be reduced at a different granularity. For example, a test can be considered as a series of characters and one or more characters can be reduced at a time to produce minimal test. If we consider HDD, reduction granularity can be a node of the AST of the program. Each test method is composed of program statements that are defined as *StatementNode*. In C# the *StatementNode* is implemented by Roslyn compiler as *StatementSytax* class or any other *StatementNode* that is derived from *StatementSyntax* class [18]. The statement node can be further decomposed for processing. Multiple previous works suggest that reduction is useful and meaningful at a statement level [9] [16]. Hence, we consider program statement or *StatementNode* as a unit of reduction. Consider a simple test as shown in Figure 1. The corresponding AST is shown in Figure 4. The dotted lines mean that the nodes can be further decomposed into non-statement nodes. But we avoid such decomposition to only consider statement level nodes. The reduced test is shown in Figure 2 and the corresponding AST is shown in Figure 5

When we compare the ASTs in Figure 4 and Figure 5, we note that two leaf statements are reduced, and one non-leaf statement is reduced (the if statement). The reduction in terms of program components is shown in Figiure 3. We want to experiment with real-world tests to study the reduction outcome and the reduced entities to establish categorization of statements and the probability of removal for each category.

IV. Terminology and definitions based on Reduction process and the outcome

We use the following terminology and definitions for the rest of the discussion.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

```
if(true){
5       int sum2 = m.Add(-2,-3)
6      Assert.Equal(sum2,-5); // This assert
   passes.
7    }
```

Figure 3. `reduced-statements`. Line 4-7 in test in Figure 1 consist of reduced statements

If the test is in a programming language like C#, we can define the following.

1) *original-test*: The test without any reduction. We consider the test to be an AST with a set of *StatementNode*s. Figure 1 shows the original test and Figure 4 shows the corresponding AST.
2) *minimal-test*: The remaining test after a test is reduced. If the *original-test* is reducible, *minimal-test* has one or more statements removed. In Figure 2, the statements from line 4 onwards in *original-test* are removed. The AST in Figure 5 depicts the AST for *minimal-test*.
3) *reduced-entities*: The program statements that are reduced during the reduction process. If original test was T and the minimal test was T' then *reduced-entities* are T - T'. *Reduced-entities* are a set of *StatementNode* or statements in our case. The *reduced-entities* are showin in Figure 3. If you compare Figure 4 and Figure 5, the *reducued-entities* are the removed subtree.
4) *TreeNode*: A *TreeNode* is a *StatementNode* that has at least one subtree that consists of one or more *StatementNode*. For example, in Figure 4, *IfStmt* is the *TreeNode*, because it contains two statements that are *TreeNode*. (1) *int sum2 = m.Add(-2,-3)* and (2) *Assert.Equal(sum2,-5)*. (Note: we are ignoring *BlockStmt*, that is explained later). In our experiment subjects, we found conditional statements, loop statements and action statements as the majority tree nodes. As we only consider statement nodes, *Treenode* can also be referred as *TreeStmt*.
5) *NonTreeNode*: A *NonTreeNode* is a *StatementNode* that does not have any subtree that consists of *StatementNode*. In Figure 4 *Math m = new Math(), int sum1 = m.Add(3,4)* etc. are *NonTreeNode*. *NonTreeNode* can also be referred as *NonTreeStmt*.

Each *BlockStmt* consists of one or more *StatementNode*s. Removing the *BlockStmt* can disturb the tree structure such that Roslyn compiler may not create syntactially correct variants [18]. Hence, *BlockStmt* is never considered during reduction. Only the statements that are below *BlockStmt* are considered for reduction as it was done with *ReduSharptor*.

We categorize the statements of a C# tests into the categories based on its location in the program: *TreeStmt* and *NonTreeStmt*. We wan to study how the quantity of removal and the probability of removal are dependent on this categorization.

V. Experiments

We want to study both *minimal-test* and *reduced-entities* to understand the effect of statement category on removal process and reduction outcome.

For that we use the same subjects and procedure used in the previous research by Weber et al [9] to evaluate a test-reduction tool *ReduSharptor*.

A. Subjects

Weber et al. used 30 real-world failing tests across five open source C# projects as the subjects. Four out of these five open source projects are under active development. Each subject was selected such that it has one or more *reduced-entities*. If the *original-test* is already minimal and cannot be reduced any further, *original-test* and *minimal-test* are the same. Hence, comparison and further evaluations are not meaningful. Accuracy of our analysis depends on the accuracy of *ReduSharptor* - *ReduSharptor* has high precision (96.58%) and high recall (96.45%). The projects that were used as subjects are enumerated in in Table 1 in the work of Weber et al. [9].

B. Process and Measurement

We apply *ReduSharptor* on each failing test, reduced the failing test and generated failure-inducing *minimal-test*. We compare failing *original-test* with the failing *minimal-test*. We collect the following information.

1) *absolute-reduction-size* (ARS): The number of statements that are reduced as part of reduction process. This is essentially the size of *reduced-entities* in terms of statements. In the example in section IV, *absolute-reduction-size* is three statements - the IfStmt and two other statements at the leaf of the IfStmt subtree.
2) *percentage-reduction-size* (PRS): The percentage of total statements reduced. In the example, the percentage is 50% - total statements are 6 and reduced statements are 3.
3) *absolutte-TreeStmt-reduction-size* (ATRS): The number of *TreeNode*s reduced. In the example, 1 *TreeNode* is reduced - the *ItStmt*.
4) *percentage-TreeStmt-reduction-size* (PTRS): The percentage of *TreeNode*s reduced. In the example, the PRTS is 16.67%.
5) *absolutte-NonTreeStmt-reduction-size* (ANTRS): The number of *NonTreeNode*s reduced. In the example, 2 *NonTreeNode*s are reduced.
6) *percentage-NonTreeStmt-reduction-size* (PNTRS): The percentage of *NonTreeNode*s reduced. In the example, PNTRS is 33.33%.

C. Results

Across 30 failing tests, we process 759 total statements. The results of reductions are shown in Table I. We show total number of statements for each test, the number of *NonTreeStmt*s, the number of *TreeStmts*s, ARS, PRS, ANTRS, PNTRS, ATRS, and PRS. On average, we processed 25.3

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

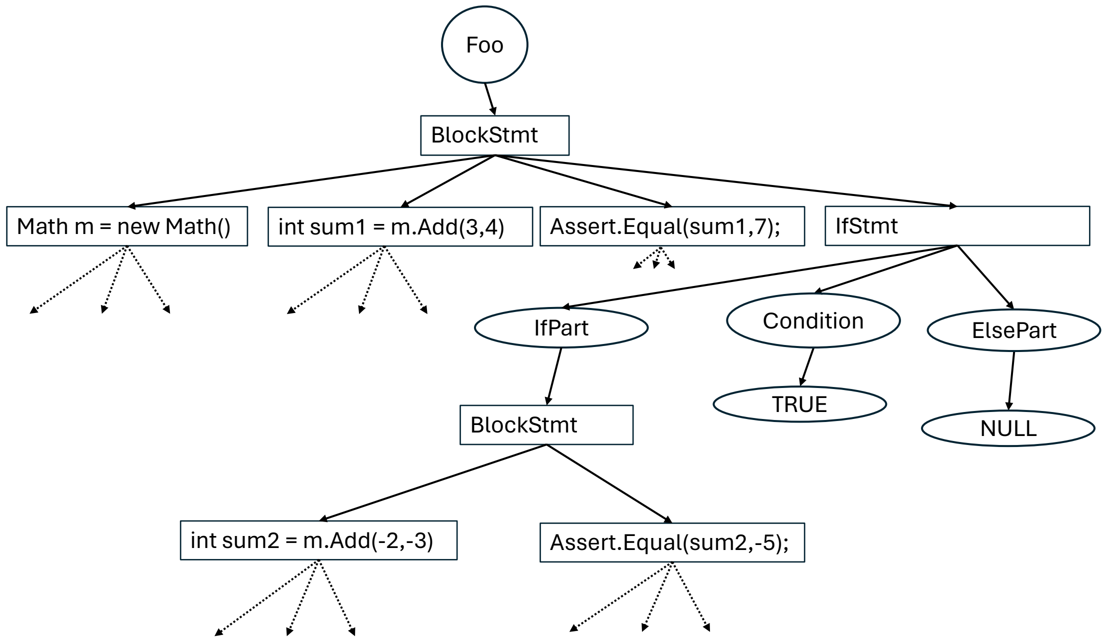

Figure 4. AST of code in Figure 1

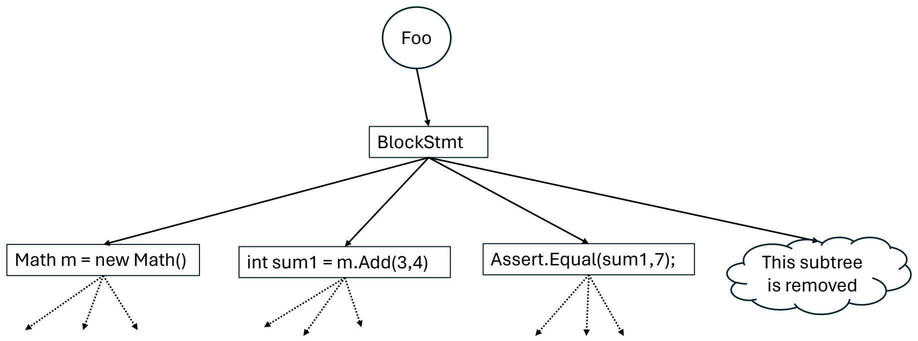

Figure 5. AST of code in Figure 2

statements per test that include 24.4 *NonTreeStmt*s and 0.9 *TreeStmt*s. We reduced on average 19.93 statements or 71.87% per failing test. For *NonTreeStmt*s, the average reduction was 19.56 or 70.44%. The same numbers for *TreeStmt*s are 0.433 and 1.43%.

The most important entities are the PRS, PNTRS and PTRS. Consider two tests - one contains 100 statements and another 10 statements. If the 20 statements are reduced in the first test and 4 statements are reduced in the the second test, the ARS, ANTRS, and ATRS numbers can be misleading. For the first test 20% statements are reduced and for the second test 40% statements are reduced. Reduction is significant for the second test.

More than half of the PTRS values are 0 and the data is not normally distributed violating the t-test assumptions. We confirm this using Shapiro-Wilk test for normality [19]. Hence, we perform paired Wilcoxon signed rank test on PNTRS and PTRS that has $V = 465$ and p-value = $1.825e - 06$ ($p << 0.05$) [20]. Wilcoxon test suggests significant difference between PNTRS and PTRS. To exactly understand the difference, we draw PNTRS vs PTRS boxplot in Figure 6. We can conclude that *NonTreeStmt*s are reduced in large numbers compared to *TreeStmt*s (approximately 50 times).

We also want to know the probability of removal of a randomly chosen statement based on its category - *TreeStmt* or *NonTreeStmt*. From the results above, we may think that if a randomly chosen statement is *NonTreeStmt*, it has more chances of removal. That may be misleading. Consider the *TestObserve* test in Table I. The test has three *TreeStmt*s and all of them are being removed resulting into 100% removal

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

of *TreeStmt*s. For the same test, out of 101 *NonTreeStmt*s 98 are removed resulting into 97% removal. For the test *TreeStmt*s have higher probability of removal. To consider this, we define two new terms: (1) ***PrNTRS*** - probability of removal of *NonTreeStmt*s defined as number of *NonTreeStmt*s removed over total *NonTreeStmts* in the test, $(ANTRS \div \#NTN) * 100$ as per Table I (2) ***PrTRS*** - probability of removal of *TreeStmt*s defined same as above but for *TreeStmt*s, $(ATRS \div \#TN) * 100$. For a certain rows in Table I, $\#TN$ is 0 and hence *PrTRS* is undefined (divide by 0). We cannot use such tests for evaluation. The results excluding the tests that have undefined *PrNTRS* are shown in Table II.

Both *PrNTRS* and *PrTRS* are not normally distributed (Shapiro-Wilk test). So, we use paired Wilcoxon signed rank test that has $V = 99$ and $p = 0.02877$ $(p < 0.05)$. So, *PrNTRS* and *PrTRS* are different. To find the difference, we again draw the values using boxplot shown in figure 7. We conclude that the probability of removal of a non-tree statement is slightly higher (approximately 1.7 times) than that of a tree statement based on the boxplot.

D. Discussion on Possible Limitations

Our experiments and results depend on how the existing tests are written for open-source C# projects. We notice that the tree-statements per test is 0.9, which is very low. For a tree statement to be reduced, the test must contain at least one or more tree statements. The PTRS entity is dependent on the availability of tree statements.

If a failed assert is at the beginning or in the middle of a test, the entities after the failed assert will always be removed. If we move that failed assert even at the end of the test, those entities will still be removed. DD and HDD algorithms guarantee 1-minimality. 1-minimality ensures that the test reduction and our analysis are not dependent on the location of the failed assert.

VI. Threats to Validity

Now, we discuss threats to validity and the steps we take to mitigate them.

A. Construct Validity

Do our results truly compare non-tree statements and tree-statements for test reduction? The *ReduSharptor* tool used for experiments has high precision and high recall. It is easy to identify tree statements and non-tree statements in a test.

B. Internal Validity

Do we mitigate bias during experiments? All the projects and tests are part of open-source projects available online. Bugs were randomly sampled such that it has one or more *reduced-entities*.

C. External Validity

Do our results generalize? We only performed experiments on C# projects and tests. As tests in other programming languages have similar structures and program statement types, we expect the results to generalize.

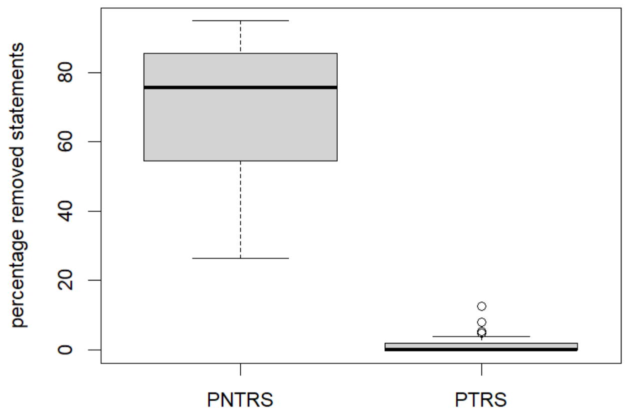

Figure 6. comparison between PNTRS vs PTRS

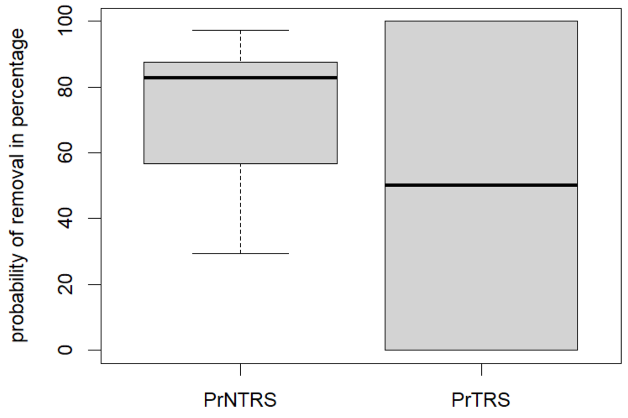

Figure 7. comparison between PrNTRS vs PrTRS

VII. Conclusion and Future Work

Based on our results, our contribution is as follows. We also plan to extend our work to further investigate the relationship between test reduction outcome and possible categorization of statements.

A. Contribution

Very few DD and HDD implementations assign priority to an entity based on the category of the entity. DD and HDD work on wide range of inputs and hence defining a generic category is difficult. If we only consider tests written as programs, we can come up with a broad generic category. Based on the categorization, we study the effect of a statement

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

TABLE I
TEST, PROJECT, TOTAL STMTS, *#NTN* - NUMBER OF *NonTreeNodes*, *#TN* - NUMBER OF *TreeNodes*, ARS, PRS, ANTRS, PNTRS, ATRS, PTRS

Test	Project	Stmts	*#NTN*	*#TN*	ARS	PRS	ANTRS	PNTRS	ATRS	PTRS
ListCombineTest	language-ext	10	10	0	6	60.00%	6	60.00%	0	0%
EqualsTest	language-ext	7	7	0	6	85.71%	6	85.71%	0	0%
ReverseListTest3	language-ext	5	5	0	2	40.00%	2	40.00%	0	0%
WriterTest	language-ext	17	15	2	8	47.06%	8	47.06%	0	0%
Existential	language-ext	14	14	0	11	78.57%	11	78.57%	0	0%
TestMore	language-ext	55	55	0	47	85.45%	47	85.45%	0	0%
CreatedBranchIsOk	Umbrraco-C..	54	54	0	39	72%	39	72%	0	0%
CanCheckIfUserHasAccessToLanguage	Umbrraco-C..	19	17	2	6	31.58%	5	26.32%	1	5.26%
Can _Unpublish_ContentVariation	Umbrraco-C..	28	28	0	25	89.29%	25	89.29%	0	0%
EnumMap	Umbrraco-C..	11	11	0	6	54.55%	6	54.55%	0	0%
InheritedMap	Umbrraco-C..	17	17	0	11	64.71%	11	64.71%	0	0%
Get_All_Blueprints	Umbrraco-C..	25	23	2	22	88.00%	20	80.00%	2	8.00%
ShouldStart	Fleck	7	5	2	3	42.86%	3	42.86%	0	0%
ShouldSupportDualStackListenWhenServerV..	Fleck	4	3	1	3	75.00%	3	75.00%	0	0%
ShouldRespondToCompleteRequestCorrectly	Fleck	15	15	0	11	73.33%	11	73.33%	0	0%
ConcurrentBeginWrites	Fleck	21	21	0	16	76.19%	16	76.19%	0	0%
ConcurrentBeginWritesFirstEndWriteFails	Fleck	27	26	1	22	81.48%	21	77.78%	1	3.70%
HeadersShouldBeCaseInsensitive	Fleck	7	7	0	5	71.43%	5	71.43%	0	0%
TestNullability	BizHawk	15	15	0	13	86.67%	13	86.67%	0	0%
TestCheatcodeParsing	BizHawk	8	7	1	7	87.50%	6	75.00%	1	12.50%
SaveCreateBufferRoundTrip	BizHawk	31	29	2	24	77.42%	24	77.42%	0	0%
TestCRC32Stability	BizHawk	27	25	2	13	48.15%	13	48.15%	0	0%
TestSHA1LessSimple	BizHawk	14	14	0	7	50.00%	7	50.00%	0	0%
TestRemovePrefix	BizHawk	14	14	0	13	92.86%	13	92.86%	0	0%
TestActionModificationPickup1	Skclusive.Mob..	23	21	2	9	39.13%	9	39.13%	0	0%
TestObservableAutoRun	Skclusive.Mob..	26	25	1	23	88.46%	22	84.62%	1	3.85%
TestMapCrud	Skclusive.Mob..	39	38	1	37	94.87%	37	94.87%	0	0%
TestObserver	Skclusive.Mob..	104	101	3	101	97.12%	98	94.23%	3	2.88%
TestObserveValue	Skclusive.Mob..	62	59	3	58	93.55%	56	88.71%	3	4.84%
TestTypeDefProxy	Skclusive.Mob..	53	51	2	44	83.02%	43	81.13%	1	1.89%
Mean		**25.3**	**24.4**	**0.9**	**19.93**	**71.87%**	**19.56**	**70.44%**	**0.433**	**1.43%**

TABLE II
TEST, *PrNTRS* VS *PrTRS* FOR INDIVIDUAL TEST. TESTS WITH UNDEFINED *PrTRS* ARE NOT INCLUDED.

Test	PrNTRS	PrTRS
WriterTest	53.33%	0.00%
CanCheckIfUserHasAccessToLanguage	19.41%	50%
Get_All_Blueprints	86.95%	100%
ShouldStart	60.00%	0.00%
ShouldSupportDualStackListenWhenServerV4All	75.00%	0.00%
ConcurrentBeginWritesFirstEndWriteFails	80.76%	100.00%
TestCheatcodeParsing	85.71%	50.00%
SaveCreateBufferRoundTrip	82.75%	0.00%
TestCRC32Stability	52.00%	0.00%
TestActionModificationPickup1	42.87%	0.00%
TestObservableAutoRun	88.00%	100.00%
TestMapCurd	97.36%	0.00%
TestObserver	97.02%	100.00%
TestObserveValue	93.22%	100.00%
TestTypeDefProxy	81.31%	50.00%

category on the reduction outcome and on the removal process. We conclude that the location of a program within the AST has an effect on the reduction outcome and the removal process. The non-tree statements (leaf nodes) will be removed in larger numbers and they will have slightly higher chance of removal.
.

B. Future Work

Our work focuses on C# tests. A very obvious extension would be to verify the results on tests written in other programming languages like Java, Python, and etc. We expect similar results for other programming languages also. Currently we categorize test statements into two categories based on the location within the AST. One area of extension would be to use other kind of categories. For example, types of the statements like declaration statement, method call statement, if statement, loop statement, try-catch statement etc. A broader extension would be to derive generic categories for non-program test inputs like html, xml, and text files.

REFERENCES

[1] A. Christi, M. L. Olson, M. A. Alipour, and A. Groce, “Reduce before you localize: Delta-debugging and spectrum-based fault localization,” in *2018 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Memphis, TN, USA, October 15-18, 2018*, 2018, pp. 184–191.

[2] D. Vince, R. Hodován, and Á. Kiss, “Reduction-assisted fault localization: Don’t throw away the by-products!” in *ICSOFT*, 2021, pp. 196–206.

[3] A. Zeller and R. Hildebrandt, “Simplifying and isolating failure-inducing input,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 183–200, Feb. 2002.

[4] G. Misherghi and Z. Su, “HDD: Hierarchical delta debugging,” in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE ’06, 2006, pp. 142–151.

[5] R. Hodován and A. Kiss, “Modernizing hierarchical delta debugging,” in *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation*, ser. A-TEST 2016. ACM, 2016, pp. 31–37.

[6] C. Sun, Y. Li, Q. Zhang, T. Gu, and Z. Su, “Perses: Syntax-guided program reduction,” in *Proceedings of the 40th International Conference on Software Engineering*. Association for Computing Machinery, 2018, p. 361–371.

[7] R. Gopinath, A. Kampmann, N. Havrikov, E. O. Soremekun, and A. Zeller, “Abstracting failure-inducing inputs,” in *29th ACM SIGSOFT international symposium on software testing and analysis*, 2020, pp. 237–248.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

[8] D. Stepanov, M. Akhin, and M. Belyaev, "Reduktor: How we stopped worrying about bugs in kotlin compiler," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 317–326.

[9] D. Weber and A. Christi, "Redusharptor: A tool to simplify developer-written c# unit tests," *International Journal of Software Engineering & Applications*, vol. 14, pp. 29–40, 09 2023.

[10] G. Wang, R. Shen, J. Chen, Y. Xiong, and L. Zhang, "Probabilistic delta debugging," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 881–892.

[11] G. Wang *et al.*, "A probabilistic delta debugging approach for abstract syntax trees," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 763–773.

[12] "ReduSharptor Tool," https://github.com/amchristi/ReduSharptor, accessed: 2024-04-24.

[13] J. Regehr *et al.*, "Test-case reduction for c compiler bugs," in *33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '12, pp. 335–346.

[14] S. Herfert, J. Patra, and M. Pradel, "Automatically reducing tree-structured test inputs," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2017, 2017, pp. 861–871.

[15] D. Binkley *et al.*, "Orbs: Language-independent program slicing," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 109–120.

[16] A. Christi, A. Groce, and R. Gopinath, "Resource adaptation via test-based software minimization," in *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE, 2017, pp. 61–70.

[17] A. Christi and A. Groce, "Target selection for test-based resource adaptation," in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, July 2018, pp. 458–469.

[18] "Roslyn Compiler Documentation," https://learn.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/, accessed: 2024-03-03.

[19] "Descriptive Statistics and Normality Tests for Statistical Data," https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6350423/, accessed: 2024-03-07.

[20] "Wilcoxon Signed Ranked Test," https://www.sciencedirect.com/topics/medicine-and-dentistry/wilcoxon-signed-ranks-test, accessed: 2024-03-07.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org