GESELLSCHAFT
FÜR INFORMATIK

Gregor Engels, Regina Hebig,
Matthias Tichy (Hrsg.)

Software Engineering 2023

Fachtagung des GI-Fachbereichs Softwaretechnik

20.-24. Februar 2023
Paderborn

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-332

ISBN 978-3-88579-726-5
ISSN 1617-5468

Volume Editors
Prof. Dr. Gregor Engels
Universität Paderborn
Zukunftsmeile 2, 33102 Paderborn, Germany
engels@upb.de

Dr. Regina Hebig
Chalmers | University of Gothenburg
Teknikgatan 7, 417 56 Göteborg, Sweden
rhebig@acm.org

Prof. Dr. Matthias Tichy
Universität Ulm
Albert-Einstein-Allee 11, 89069 Ulm, Germany
matthias.tichy@uni-ulm.de

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Herzlich willkommen zur Fachtagung Software Engineering 2023 (SE23) des Fachbereichs Softwaretechnik der Gesellschaft für Informatik (GI). Die jährliche Tagung des Fachbereichs Softwaretechnik der GI hat sich als Plattform für den Austausch und die Zusammenarbeit in allen Bereichen der Softwaretechnik im deutschsprachigen Raum etabliert. Dabei spricht die Tagung sowohl Softwareentwickler*innen aus der Praxis als auch Forscher*innen aus dem akademischen Umfeld an, um einen Austausch der neuesten akademischen Erkenntnisse als auch aktueller industrieller Trends und Praktiken zu ermöglichen.

Die SE23 präsentiert im wissenschaftlichen Hauptprogramm ein "Best-of" der international in Fachzeitschriften und Konferenzen veröffentlichten Arbeiten deutschsprachiger Autor*innen. Die angenommenen wissenschaftlichen Beiträge decken dabei ein weites Spektrum des Software Engineering ab, das sich in einem vielfältigen Programm widerspiegelt. Darüber hinaus sind alle Daten und Artefakte der Arbeiten öffentlich verfügbar. Bei Fällen, in denen das nicht möglich ist, ist eine Begründung angegeben. Auf diese Weise wird ein wichtiges Prinzip der transparenten Wissenschaft berücksichtigt. Dieses Jahr gab es für das Hauptprogramm 65 Einreichungen, von denen 52 akzeptiert wurden.

Die SE23 bietet neben dem wissenschaftlichen Hauptprogramm auch noch die folgenden Tracks:

- Student Research Competition
- Industrieprogramm
- Ernst Denert Software-Engineering-Preis

Schließlich wird die Tagung durch fünf im Rahmen der SE23 organisierte Events (drei Workshops und zwei Konferenzen) ergänzt, in denen weitere Themen diskutiert werden:

- deRSE 23 – Conference for Research Software Engineering in Germany
- SEUH 2023 – Software Engineering im Unterricht der Hochschulen
- 5. Workshop für Avionics Systems- und Software-Engineering (AvioSE'23)
- 4. Workshop: Anforderungsmanagement in Enterprise Systems-Projekten (AESP'23)
- 20. Workshop Automotive Software Engineering (ASE'23)

Wir danken allen, die zum Gelingen der Tagung beigetragen haben, insbesondere den Autor*innen, den Gutachter*innen, den Keynote-Speakern, den Organisator*innen der Workshops und Tracks, den Teilnehmer*innen, den Sponsoren, der Stadt Paderborn und der GI e.V., Proceedings Chair Enes Yigitbas, Publicity Chair Kerstin Sellerberg und für die lokale Organisationsunterstützung durch Gabriele Stall, dem SICP – Software Innovation Campus Paderborn sowie allen Helfer*innen, die die Durchführung der Tagung ermöglicht haben.

Paderborn, im Februar 2023

Gregor Engels, Regina Hebig und Matthias Tichy

Sponsoren

Wir danken den folgenden Unternehmen und Institutionen für die Unterstützung der Fachtagung Software Engineering 2023:

Platin-Sponsoren

CQSE GmbH
Centa-Hafenbrädl-Straße 59
81249 München
Webseite: www.cqse.eu

dSPACE GmbH
Rathenaustraße 26
33102 Paderborn
Webseite: www.dspace.com

S&N Invent GmbH
Klingenderstraße 5
33100 Paderborn
Webseite: www.sn-invent.de

UNITY AG
Lindberghring 1
33142 Büren
Webseite: www.unity.de

Weidmüller GmbH & Co KG
Klingenbergstraße 26
32758 Detmold
Webseite: www.weidmueller.de

Gold-Sponsoren

adesso SE
Adessoplatz 1
44269 Dortmund
Webseite: www.adesso.de

Connext Communication GmbH
Balhorner Feld 11
33106 Paderborn
Webseite: www.connext.de

DATEV eG
Paumgartnerstraße 6-14
90429 Nürnberg
Webseite: www.datev.de

MaibornWolff GmbH
Theresienhöhe 13
80339 München
Webseite: www.maibornwolff.de

msg systems ag
Robert-Bürkle-Straße 1
85737 Ismaning
Webseite: www.msg.group

Porsche Digital GmbH
Grönerstraße 11/1
71636 Ludwigsburg
Webseite: www.porsche.digital

WPS – Workplace Solutions GmbH
Hans-Henny-Jahnn-Weg 29
22085 Hamburg
Webseite: www.wps.de

Silber-Sponsoren

Accso – Accelerated Solutions GmbH
Hilpertstraße 12
64295 Darmstadt
Webseite: www.accso.de

Finanz Informatik GmbH & Co. KG
Theodor-Heuss-Allee 90
60486 Frankfurt a. M.
Webseite: www.f-i.de

myconsult GmbH
Geseker Str. 5
33154 Salzkotten
Webseite: www.myconsult.de

Supporter

Universität Paderborn | SICP – Software Innovation Campus Paderborn
Zukunftsmeile 2
33102 Paderborn
Webseiten: www.upb.de, www.sicp.de

Stadt Paderborn
Am Abdinghof 11
33098 Paderborn
Webseite: www.paderborn.de

Tagungsleitung

Gesamtleitung:	Gregor Engels, Universität Paderborn
Leitung des Programmkomitees:	Regina Hebig, Chalmers | University of Gothenburg Matthias Tichy, Universität Ulm
Industrie:	Carola Lilienthal, WPS – Workplace Solutions GmbH Stefan Sauer, SICP | Universität Paderborn
Workshops:	Iris Groher, Johannes Kepler Universität Linz Thomas Vogel, Humboldt-Universität zu Berlin
Publicity:	Kerstin Sellerberg, Universität Paderborn
Proceedings:	Enes Yigitbas, Universität Paderborn
RSE Local Chair:	Daniel Röwenstrunk, Universität Paderborn
PC Young Researchers:	Rebekka Wohlrab, Chalmers | University of Gothenburg Robert Heinrich, Karlsruher Institut für Technologie

Programmkomitee

Sven Apel	Universität des Saarlandes
Eric Bodden	Universität Paderborn
Ruth Breu	Universität Innsbruck
Michael Felderer	Universität Innsbruck
Sabine Glesner	Technische Universität Berlin
Lars Grunske	Humboldt-Universität Berlin
Paula Herber	Westfälische Wilhelms-Universität Münster
Marie-Christine Jakobs	Technische Universität Darmstadt
Timo Kehrer	Universität Bern
Jil Ann-Christin Klünder	Leibniz Universität Hannover
Leen Lambers	BTU Cottbus-Senftenberg
Grischa Liebel	Reykjavik University
Michael Pradel	Universität Stuttgart
Jan Oliver Ringert	Bauhaus-Universität Weimar
Christoph Seidl	IT University of Copenhagen
Gabriele Taentzer	Philipps-Universität Marburg
Manuel Wimmer	Johannes Kepler Universität Linz
Rebekka Wohlrab	Chalmers | University of Gothenburg
Uwe Zdun	Universität Wien

Inhaltsverzeichnis

Keynotes

Wissenschaftliches Hauptprogramm

Workshops

Keynotes

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 19

Quantum Software Engineering – Quo Vadis?

Ina Schaefer[1]

Abstract: Quantentechnologien und insbesondere Quantencomputing versprechen die effiziente Lösung rechenintensiver Probleme, die auf klassischer Hardware nicht oder nicht effizient gelöst werden können. Bisher liegt ein großer Fokus in der Forschung auf der Entwicklung skalierbarer und fehlertoleranter Quantenhardware Während diese Hardware jetzt zunehmend verfügbar wird, wird diese Hardware jedoch erst effizient nutzbar, wenn auch in der Software gleichermaßen Fortschritte gemacht werden. So wie im klassischen Computing die Wertschöpfung durch Software entsteht, wird auch im Quantencomputing der wirkliche Nutzen erst durch entsprechende Software und Anwendungen entstehen. Es gibt bereits eine Reihe von Assembler- und Gatter-orientierten Sprachen, sowie Simulatoren, mit denen (einfache) Quantenalgorithmen entwickelt und erprobt werden können. Jedoch fehlt es an höheren Programmiersprachen und -werkzeugen, zugehörigen Compilern für komplexe (hybride) Quanten-Rechnerarchitekturen und Software-Entwicklungsmethoden für den Entwurf von Quantenalgorithmen, sowie an Vorgehensweisen für die Strukturierung oder die Qualitätssicherung der entwickelten Quantensoftware. In diesen Vortrag zeige ich auf, wie durch das Software Engineering die notwendigen Beiträge geleistet werden können, um die Weiterentwicklung des Quantencomputings in die breitere Anwendung und die zukünftige Wertschöpfung der Quantentechnologie zu ermöglichen.

Keywords: Quantencomputing, Software Engineering, Quantentechnologie

[1]Karlsruher Institut für Technologie (KIT), Am Fasanengarten 5, 76131 Karlsruhe, ina.schaefer@kit.edu

Diversity and Inclusion in Software Engineering

Alexander Serebrenik[1]

Abstract: Community smells are patterns indicating suboptimal organization and communication of software development teams that have been shown to be related to suboptimal organisation of the source code. Given a long standing association of women and communication mediation, we have conducted a series of studies relating gender diversity to community smells, as well as comparing the results of the data analysis with developers' perception. To get further insights in the relation between gender and community smells, we replicate our study focusing on the Brazilian software teams; indeed, culture-specific expectations on the behavior of people of different genders might have affected the perception of the importance of gender diversity and refactoring strategies when mitigating community smells. Finally, we extend the prediction model by including variables related to national diversity and see how the interplay between national diversity and gender diversity influences presence of community smells.

This talk is based on a series of papers published in 2019-2022 and co-authored with Gemma Catolino, Filomena Ferrucci, Stefano Lambiase, Tiago Massoni, Fabio Palomba, Camila Sarmento, and Damian Andrew Tamburri.

Keywords: Diversity, Inclusion, Software Engineering

[1]Eindhoven University of Technology, Groene Loper 5, 5612 AE Eindhoven, a.serebrenik@tue.nl

Software-Engineering-Fortbildung für Studierende und Industrie

Stefan Wagner[1]

Abstract: Die digitale Transformation aller Lebensbereiche und Industrien ist in vollem Gange und benötigt viele Fachkräfte. Die aktuellen Hochschulabsolvent*innen decken diesen Bedarf bei weitem nicht. Durch Umwälzungen in anderen Bereichen, wie beispielsweise der Elektrifizierung in der Mobilität, werden dagegen andere Kompetenzen nicht mehr gebraucht. Stattdessen sind Kompetenzen in Software und Künstlicher Intelligenz überall gefragt. Dafür werden maßgeschneiderte, flexible Programme zur Fortbildung im Software Engineering benötigt. Diese Keynote berichtet von zwei Beispielen, wie dies realisiert werden kann:
(1) Wir bieten eine modulare Grundausbildung Software Engineering für die Industrie an. Hier werden Mitarbeiter*innen von Unternehmen mit existierender Hochschulbildung auf den Einsatz in Softwareprojekten geschult. Dieses Programm umfasst Vorlesungen und Übungen an ein bis zwei Tagen über ein ganzes Jahr, Abschlussklausuren und zwei Projektphasen. Hier haben bereits knapp 70 Teilnehmende aus der Industrie in vier Jahrgängen das Programm durchlaufen.
(2) Studierende und Promovierende können Microdegrees an der Schnittstelle von Künstlicher Intelligenz, Software Engineering und Anwendungen erwerben. Den Rahmen dazu bietet die Artificial Intelligence Software Academy (AISA) gefördert durch das MWK Baden-Württemberg, das Forschung und Lehre an dieser Schnittstelle verbindet. Es können flexible Microdegrees erworben werden, die diese Kombination von Software Engineering und Künstlicher Intelligenz enthalten und sich insbesondere auch an Nichtinformatiker*innen richten. Damit werden diese Kompetenzen Basisfähigkeiten für alle Fachbereiche, erhöhen die Einsatzmöglichkeiten der Teilnehmenden der Fortbildung und reduzieren den Fachkräfte-Mangel. Dies unterstreicht die praktische Relevanz des Software Engineerings in der Praxis und zeigt den wichtigen Beitrag, den Universitäten hier leisten können.

Keywords: Software Engineering, Fortbildung, Industrie

[1]Universität Stuttgart, Universitätsstraße 38, 70569 Stuttgart, stefan.wagner@iste.uni-stuttgart.de

Leadership for Software Engineers in Practice: An Industrial Experience Report

Mahdi Manesh[1]

Abstract: As software plays an increasingly important role for many companies, the requirements for senior software engineers go beyond developing products and services. If they like it or not, they frequently find themselves becoming key players in organizational transformation processes. Sure enough, these playing fields can be vastly different from what engineering training commonly prepares for. Better get ready!

In this presentation, the speaker shares his personal view on the subject of leadership in software engineering projects, based on real-world experiences made in various industrial settings and roles (i.e., founder, researcher, manager).

The main goal of this talk is to (re)initiate a fruitful discussion about skills and know-how needed for succeeding in managerial roles in tech. Definitely, these positions are full of opportunities and critical to the success of small and large organizations alike.

Keywords: Leadership, Software Engineering

[1]Porsche Digital GmbH, Stralauer Allee 12, 10245 Berlin, mahdi.manesh@porsche.digital

Wissenschaftliches Hauptprogramm

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 29

Formal Verification of Intelligent Hybrid Systems that are modeled with Simulink and the Reinforcement Learning Toolbox

Julius Adelt[1], Timm Liebrenz[1], Paula Herber[1]

Abstract: Reinforcement Learning (RL) is a powerful technique to control intelligent hybrid systems (HS) in dynamic and uncertain environments. However, formally guaranteeing safe behavior of intelligent HS is hard because formal descriptions are often not available in industrial design processes and hard to obtain for RL. Furthermore, the intertwined discrete and continuous behavior of hybrid systems results in limited scalability of automatic verification methods, such as model checking. This makes deductive verification desirable. In this paper, we summarize our approach for deductive verification of intelligent HS with embedded RL components that are modeled with Simulink and the RL Toolbox. This paper was originally published at the Formal Methods conference 2021 (FM21) [ALH21].

Keywords: Formal Verification; Theorem Proving; Hybrid Systems; Safe Reinforcement Learning

Summary of Our Approach

Our approach for deductive verification of intelligent HS is depicted in Fig. 1 [ALH21]. We assume that a given intelligent HS is modeled in Simulink together with the RL Toolbox [Si22], that is, it contains an RL agent block. The key idea of our approach is threefold: ① We solve the problem of formally describing RL agents by capturing their safe behavior in a hybrid contract (HC), which defines safe actions under given observations. This provides a formally well-defined basis for an RL agent's behavior in an HS. ② To overcome the problem that formal descriptions of intelligent hybrid systems are often not available, we capture the semantics of intelligent Simulink models in differential dynamic logic (d$\mathcal{L}$) [Pl08]. To achieve this, we extend our Simulink2d$\mathcal{L}$ transformation [LHG21] to reinforcement learning. During transformation, the RL agent is replaced by its HC. This enables the transformation of Simulink models with embedded RL components into a formal d$\mathcal{L}$ representation. For this d$\mathcal{L}$ representation, safety properties can be verified deductively with the interactive theorem prover KeYmaera X [Fu15]. Our transformation also enables compositional verification by verifying hybrid contracts for groups of Simulink blocks (services) individually. If the verification fails, a counter-example can often be obtained, which may be used for debugging. ③ We ensure that the RL agent complies with its HC at simulation time with automatically generated runtime monitors based on [FP18].

[1] University of Münster, Einsteinstraße 62, 48149 Münster, Germany, firstname.lastname@uni-muenster.de

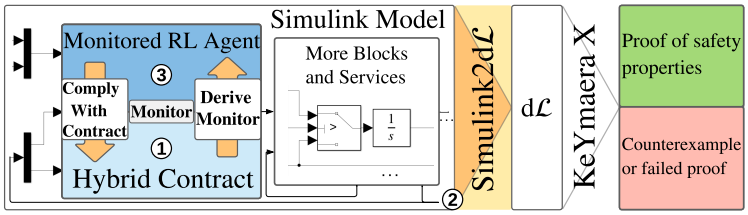


Fig. 1: Deductive Verification of Intelligent Hybrid Systems [ALH21]

To demonstrate the applicability of our approach, we use a case study of a learning-enabled autonomous robot in a factory setting. We verify collision freedom with two and with six other robots and also for a scenario with disturbed sensor data.

Data Availability

Our transformation is available on Github: `https://github.com/EmbSys-WWU/Simulink2dL`. All our proof results and Simulink models are available on our website: `https://www.uni-muenster.de/EmbSys/research/Simulink2dL.html`. The Matlab code of the monitored RL agent and the simulation experiments are not publicly available for licensing reasons.

References

[ALH21] Adelt, J.; Liebrenz, T.; Herber, P.: Formal Verification of Intelligent Hybrid Systems that are modeled with Simulink and the Reinforcement Learning Toolbox. In: Formal Methods (FM). Vol. 13047. LNCS, Springer, 2021.

[FP18] Fulton, N.; Platzer, A.: Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. AAAI Conf. on Artif. Intellig. 32/, 2018.

[Fu15] Fulton, N.; Mitsch, S.; Quesel, J.-D.; Völp, M.; Platzer, A.: KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In: International Conference on Automated Deduction. Vol. 9195. LNCS, Springer, pp. 527–538, 2015.

[LHG21] Liebrenz, T.; Herber, P.; Glesner, S.: Service-oriented decomposition and verification of hybrid system models using feature models and contracts. Science of Computer Programming 211/, p. 102694, 2021.

[Pl08] Platzer, A.: Differential Dynamic Logic for Hybrid Systems. Journal of Automated Reasoning 41/2, pp. 143–189, 2008.

[Si22] Simulink Documentation: Simulation and Model-Based Design, 2022, URL: `https://www.mathworks.com/products/simulink.html`.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 31

Peer-Reviewing and Submission Dynamics Around Top Software-Engineering Venues: A Juniors' Perspective

Rand Alchokr[1] Jacob Krüger[2] Yusra Shakeel[3] Gunter Saake [4] Thomas Leich [5]

Abstract: In this extended abstract, we summarize our paper with the homonymous title published at the International Conference on Evaluation and Assessment in Software Engineering (EASE) 2022 [Al22].

Keywords: juniors; peer review; bias; challenges; collaboration

Background: Research is an intrinsically challenging process full of obstacles. However, these obstacles may be more dominant for a specific group of researchers (such as junior researchers) compared to others. It is the responsibility of the community to pay close attention to those groups that may be struggling for unfair reasons and provide necessary support. Junior researchers are of high importance to the scientific community, and are defined as young researchers who have recently started their research career [Li19]. Despite their importance, juniors may face impediments when starting their career that hinder their activities and motivation. For instance, collaboration aspects and peer-reviewing models can play a role. Junior researchers without a high reputation (e.g., via their co-authors) may be negatively impacted by reputation biases, and thus could have even more problems with publishing and building their reputation independently. In our study, we investigate what challenges junior researchers perceive when submitting their work to software-engineering venues with a high reputation.

Objective: Only few studies have analyzed the contributions of juniors in the software-engineering community [Al21] and the challenges they face. We aimed to identify and understand what kinds of challenges junior researchers experience when aiming to publish their research and getting involved into the community. For this purpose, we conducted an online survey with a focus on two common types of challenges: peer-reviewing models (double-blind and single-blind) and collaboration.

Method: We designed and conducted an exploratory web-based survey with which we targeted active software-engineering researchers, aiming to identify the community's awareness of juniors' challenges. In general, we structured our survey around four research

[1] Otto-von-Guericke-University Magdeburg, rand.alchokr@ovgu.de
[2] Eindhoven University of Technology, The Netherlands, j.kruger@tue.nl
[3] Karlsruhe Institute of Technology, yusra.shakeel@kit.edu
[4] Otto-von-Guericke-University Magdeburg, saake@ovgu.de
[5] Harz University of Applied Sciences, tleich@hs-harz.de

questions concerning the community's opinion on the fairness of reviewing models (double-blind versus single-blind) and the importance of collaborations. We translated each of our research questions into several survey questions, which we arranged according to the survey's homogeneous flow. Mostly, we relied on close-ended questions, sometimes followed by open-ended ones. We used mailing lists for software-engineering researchers and Twitter as channels to distribute the survey. To empirically evaluate our results and answer our research questions, we used descriptive statistics and visualizations to analyze the responses.

Results: A total of 52 respondents completed our survey, with the majority having publishing experience reflected by the number of papers they have published. Regarding the academic position or role, 34 responses out of 52 stemmed from PhD students. The results indicate that the majority of our participants favors double-blind reviewing with more than half of them (67.2 %) voting in favor of it, believing that single-blind reviewing favors seniors and negatively impacts juniors. However, our participants indicate that reviewing models do not affect their submission decisions. When looking at juniors, they seem to hesitate to submit to highly prestigious venues and believe that collaborations with seniors raises their papers' chances of getting accepted. Finally, our participants agree that the chances of getting papers accepted are not equal for juniors and seniors, with a lack of experience and academic writing skills posing the strongest barriers for junior researchers. Also, our participants agree that supervision and work-group problems pose strong barriers.

Conclusion: Our findings indicate a high level of awareness inside the software-engineering community regarding the challenges junior researchers face. Our study is only a first step in accomplishing a comprehensive analysis of our community and the challenges certain groups of researchers face. Our findings can be used to define these challenges and start contributing to their solutions. Moreover, we provide insights into diversity and inclusion aspects inside the software-engineering research community.

Data Availability: We share our questionnaire and anonymized raw data in a publicly available repository.[6]

Bibliography

[Al21] Alchokr, Rand; Krüger, Jacob; Shakeel, Yusra; Saake, Gunter; Leich, Thomas: Understanding the Contributions of Junior Researchers at Software-Engineering Conferences. In: Joint Conference on Digital Libraries (JCDL). IEEE, 2021.

[Al22] Alchokr, Rand; Krüger, Jacob; Shakeel, Yusra; Saake, Gunter; Leich, Thomas: Peer-Reviewing and Submission Dynamics Around Top Software-Engineering Venues: A Juniors' Perspective. In: International Conference on Evaluation and Assessment in Software Engineering (EASE). IEEE, 2022.

[Li19] Li, Weihua; Aste, Tomaso; Caccioli, Fabio; Livan, Giacomo: Early Coauthorship with Top Scientists Predicts Success in Academic Careers. Nature Communications, 10, 2019.

[6] https://doi.org/10.5281/zenodo.6463764

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 33

Synthesizing of Process-Aware Digital Twin Cockpits from Event Logs

Dorina Bano[1], Judith Michael[2], Bernhard Rumpe[2], Simon Varga[2], Mathias Weske[1]

Abstract: In this work, we summarize our article "Process-Aware Digital Twin Cockpit Synthesis from Event Logs" published in the Journal of Computer Languages (COLA). The engineering of digital twins and their user interfaces with explicated processes, namely process-aware digital twin cockpits (PADTCs), is challenging due to the complexity of the systems and the need for information from different disciplines within the engineering process. Therefore, we have investigated how to facilitate their engineering by using already existing data, namely event logs. We present a low-code development approach that reduces the amount of hand-written code needed to derive PADTCs using process mining techniques. We describe what models could be derived from event log data, which generative steps are needed for the engineering of PADTCs, and how process mining could be incorporated into the resulting application. A PADTC prototype is created based on the MIMIC III dataset, which simulates an automated hospital transportation system. Initially, our approach requires no hand-written code and empowers the domain expert to iteratively create PADTC prototypes.

Keywords: Process-Aware Digital Twin Cockpit; Low-Code Development Approaches; Sensor Data; Event Log; Process Mining; Process-Awareness

1 Summary

Improving the engineering process of digital twins (DTs) with a higher degree of automation enables domain experts to take an active role in this process and allows for an iterative evolvement of the DT. A DT of an actual object can include process-aware DT cockpits (PADTCs) which allow for user interaction and provide means to handle processes of the actual object and its context. Full automation of the DT engineering process and its services might be a big vision. However, the automated engineering of PADTCs is the first reachable goal. Within [Ba22], we have suggested a low-code development approach for generating PADTCs from event logs. Our approach includes *four phases*.

In the *preparation phase*, we analyze the real world and extract models from data about the actual object. We extract an event log from the sensor data of an actual object. Using this event log, we perform data-to-model transformations and discover domain information in a class diagram, process models in BPMN, and roles in a tagging model.

In the *generation phase*, we apply model-to-model and model-to-code transformations to generate a PADTC. We are using the models from the preparation phases as input for

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany {dorina.bano,mathias.weske}@hpi.de
[2] Software Engineering, RWTH Aachen University, Aachen, Germany {michael,rumpe,varga}@se-rwth.de

model-to-model transformations and generate data models (as views) and GUI models. These generated models and the domain model are the generator input to synthesize the PADTC source code.

In the *adaption phase*, we can extend the generated parts by handwritten additions on models and code levels. We can add handwritten models which are used as additional input to synthesize the code of the PADTC with a code generator. Additionally, we can add handwritten code in addition to the generated code.

For *runtime of the PADTC*, we connected the PADTC to DT services to get a fully functional digital twin. For a process-aware digital twin, especially the connection to services for process discovery, process conformance checking and process prediction services are relevant. Additionally, the PADTC visualizes live data from the actual object or third-party applications and enables domain users to interact with the DT. The DT can influence the actual object via user commands from the PADTC or automatic commands from self-adaptive services.

Using this low-code development approach, we have created a PADTC prototype that simulates a hospital transportation system. For its creation, we have used event logs extracted from the MIMIC III dataset [Jo16], a dataset with health-related data of patients who stayed in critical care units. However, our approach is not limited to this domain and could be applied to engineer PADTCs for areas where process data is available, e.g., tools and machines in production, products, experiments, software systems, or organizations.

The application has shown, that we can achieve a high degree of automation in the PADTC engineering process and reduce the amount of hand-written code needed: Developers need no hand-written code in the first place to get a deployable prototype. This can be extended by hand-written models and code to create a digital twin.

Data Availability. The original publication is accessible under `http://doi.org/10.1016/j.cola.2022.101121`, a preprint is available at `https://www.se-rwth.de/publications`. Unfortunately, we could not provide more detailed artifacts for reasons of confidentiality.

Acknowledgments. This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2023 Internet of Production - 390621612. Website: https://www.iop.rwth-aachen.de

Bibliography

[Ba22] Bano, Dorina; Michael, Judith; Rumpe, Bernhard; Varga, Simon; Weske, Matthias: Process-Aware Digital Twin Cockpit Synthesis from Event Logs. Journal of Computer Languages (COLA), 70, June 2022.

[Jo16] Johnson, Alistair EW; Pollard, Tom J; Shen, Lu; Lehman, Li-wei H; Feng, Mengling; Ghassemi, Mohammad; Moody, Benjamin; Szolovits, Peter; Celi, Leo Anthony; Mark, Roger G: MIMIC-III, a freely accessible critical care database. Scientific data, 3, 2016.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 35

Editing Support for Software Languages: Implementation Practices in Language Server Protocols (Summary)

Djonathan Barros[1], Sven Peldszus[2], Wesley K. G. Assunção[1,3], Thorsten Berger[2,4]

Abstract: We present our paper published at the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS) [Ba22a]. Effectively using software languages requires effective editing support. Modern IDEs, modeling tools, and code editors typically provide sophisticated support to create, comprehend, or modify instancesof particular languages. Unfortunately, building such editing support is challenging. While the engineering of languages is well understood and supported by modern model-driven techniques, there is a lack of engineering principles and best practices for realizing their editing support. We study practices for implementing editing support in so-called language servers—implementations of the language server protocol (LSP). LSP is a recent de facto standard to realize editing support for languages, separated from the editing tools, enhancing the reusability and quality of the editing support. Witnessing the LSP's popularity, we take this opportunity to analyze the implementations of 30 language servers. We identify concerns that developers need to take into account when developing editing support, and we synthesize implementation practices to address them, based on a systematic analysis of the servers' source code. We hope that our results shed light on an important technology for software language engineering, that facilitates language-oriented programming and systems development, including model-driven engineering.

Keywords: Language engineering; code assistance; source code editor; implementation practices

1 Summary

Software languages are paramount—not only to software engineering, but also to many other engineering disciplines that need to create models and automate tasks. Effectively using software languages requires effective editing support. Modern IDEs and modeling tools often come with sophisticated editing support to create, comprehend, and modify programs or models expressed in a certain language. Typical editing support features are code completion, syntax highlighting, error marking, formatting, and refactoring, among others.

Unfortunately, creating proper editing support for languages is difficult. While for mainstream software languages, the vendors of software engineering or modeling tools typically invest the necessary resources to realize editing support, domain-specific languages (DSLs) are often created by smaller organizations or individual developers. Sometimes, their use is limited to specific purposes or only a few projects. As such, DSLs would especially benefit from better support to realize editing support—allowing their users focus on solving real problems, instead of wasting time with learning the exact use of individual DSLs.

[1] PPGComp, Western Paraná State University, Brazil
[2] Ruhr University Bochum, Germany
[3] Johannes Kepler University, Austria
[4] Chalmers | University of Gothenburg, Sweden

The language server protocol (LSP) [Mi] is a recent initiative from 2016 to modularize editing support into the so-called language servers. The LSP addresses the problem that language-specific editing support is currently deeply integrated into single IDEs or editors, preventing its reuse or extension for different tools. The LSP aims at enabling language engineers to make their languages, be it programming languages or DSLs, available to a wide range of editing tools while requiring minimal effort for adoption and reuse. The LSP describes a common API that can be implemented and reused by different clients [Mi].

We took this opportunity and investigated the design and realization of real language servers. Our goal was to identify implementation *concerns* related to the realization of language editing support. Our working hypothesis was that engineering principles can be identified within existing language servers, paving the way to our long-term goal of establishing patterns and pattern catalogs for realizing editing support. We followed a thematic synthesis method by coding, analyzing, grouping, and reporting how editing features are implemented.

We present the first set of engineering practices, systematically identified—quantitatively and qualitatively— from a sample of 30 LSP servers. We synthesized seven core concerns and present practices for realizing language editing support. We found that a variety of features are required to provide editing support, and the concrete aspects of languages play a rather minor role in the basic editing support. Still, for advanced editing support, characteristics of the target language become more important. We hope to provide researchers with concerns related to the realization of editing support and insights on implementation practices for addressing them, to spark a discussion on best practices and eventually developing a theory and novel techniques on realizing effective editing support for languages. We hope that practitioners can use our concerns and discussed solutions as guidelines when implementing new servers or extending existing ones. For designing LSP servers, we identified design and implementation practices that we hope to extend to a reference architecture in later works.

2 Data Availability

The summarized work is available open access in the conference proceedings [Ba22a] and we provide all raw and processed data in our replication package at Zenodo [Ba22b].

Bibliography

[Ba22a] Barros, D.; Peldszus, S.; Assunção, W. K. G.; Berger, T.: Editing Support for Software Languages: Implementation Practices in Language Server Protocols. In: International Conference on Model Driven Engineering Languages and Systems (MODELS). 2022. https://doi.org/10.1145/3550355.3552452.

[Ba22b] Supplementary Material – Editing Support for Software Languages: Implementation Practices in Language Server Protocols 59, https://doi.org/10.5281/zenodo.6974153.

[Mi] Microsoft: Language Server Protocol. https://microsoft.github.io/language-server-protocol/.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 37

Component-based CEGAR - Building Software Verifiers from Off-the-Shelf Components

Dirk Beyer[1] Jan Haltermann[2] Thomas Lemberger[3] Heike Wehrheim[4]

Abstract: Software verification tools typically consist of tighly coupled components, thereby precluding the easy integration of off-the-shelf components. We propose to decompose software verification into independent subtasks, each task being implemented by an own component communicating with other components via clearly defined interfaces. We apply this idea of decomposition to one of the most frequently used techniques in software verification: CEGAR. Our decomposition, called component-based CEGAR (C-CEGAR), comprises three components: An abstract model explorer, a feasibility checker and a precision refiner. It allows employing conceptually different components for each task within one instance. Our evaluation shows that C-CEGAR has, compared to a monolithic CEGAR-implementation, a similar efficiency and that the precision in solving verification tasks even increases.

Keywords: Software engineering, Software verification, Abstraction refinement, CEGAR, Decomposition, Cooperative verification

Component-based CEGAR

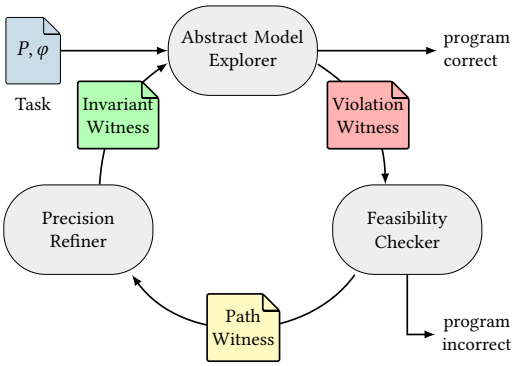


Fig. 1: Workflow of component-based CEGAR

The past years have seen enormous progress in software verification. Although there is an interest in standardizing verification artifacts (e.g., using witnesses), the verification task itself – though consisting of several subtasks – is predominantly realized using strongly cohesive software units with stateful components. This makes reusing components complicated, impacts scalability (e.g., parallelization) and hampers exchange and integration of new (off-the-shelf) components.

[1] LMU Munich, Munich, Germany, dirk.beyer@sosy-lab.org
[2] University of Oldenburg, Oldenburg, Germany, jan.haltermann@uol.de
[3] LMU Munich, Munich, Germany, thomas.lemberger@sosy.ifi.lmu.de
[4] University of Oldenburg, Oldenburg, Germany, heike.wehrheim@uol.de

We propose to *decompose* the construction of verifiers into independent components and employ *cooperative verification* to have the components solve verification tasks together. We demonstrate the feasibility of this idea by applying the decomposition to the *counterexample-guided abstraction refinement* (CEGAR) scheme. CEGAR's main concept during verification is to iteratively refine an abstract model of the system using infeasible counterexamples.

The result of the decomposition, called C-CEGAR, is depicted in Fig. 1. It comprises three components: *Abstract model explorer*, *feasibility checker* and *precision refiner*. The interfaces for the communication between them – *violation*, *path* and *invariant witnesses* – are defined using the existing and standardized witness format. The abstract model explorer builds an abstract model of the program and therein searches for property violations, given the task (program P and property φ) and an invariant witness (encoding the so-far discovered abstraction predicates). If no violation is found, the program is proven correct; otherwise, it constructs an error path leading to the violation, encoded as violation witness. The feasibility checker checks the counter-example encoded in the violation witness for feasibility. If it is feasible, the program violates the property. Otherwise, a path witness encoding the infeasible path is built and given to the precision refiner which computes additional abstraction predicates. These ensure that the infeasible counter-example is not rediscovered. To be able to employ off-the-shelf tools within C-CEGAR, we provide constructions for using a verifier as abstract model explorer or feasibility checker and an invariant generator as precision refiner [Be22].

We realized C-CEGAR with CoVeriTeam. First, we decomposed CPAchecker's CEGAR-based predicate abstraction and than employed it with FShell-witness2test and Ultimate Automizer as additional standalone-components within C-CEGAR. The evaluation on 8 347 verification tasks has shown that (1) the decomposition of an existing CEGAR implementation has (almost) no negative effects on the effectiveness, on the contrary increasing the precision through the use of more sophisticated components, and that the efficiency only decreases by a constant factor; (2) the cost of in addition using standardized formats for exchanging information leads to a reduction of the effectiveness by around 20% and (3) using different components within one C-CEGAR instance is simple and pays off.
In short: Building software verifiers from off-the-shelf components is doable and worth it.

Data Availability
Our implementation of C-CEGAR is open source and available online as part of CoVeriTeam; Our artifact (evaluated as available and reusable) containing the implementation and all experimental data is archived at Zenodo (`https://doi.org/10.5281/zenodo.5301636`).

Bibliography

[Be22] Beyer, D.; Haltermann, J.; Lemberger, T.; Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: Proc. ICSE. ACM, pp. 536–548, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 39

Classifying Edits to Variability in Source Code – Summary

Paul Maximilian Bittner[1], Christof Tinnes[2], Alexander Schultheiß[3], Sören Viegener[4], Timo Kehrer[5], Thomas Thüm[6]

Abstract: We report about recent research on edit classification in configurable software, originally published at the 30th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) 2022 [Bi22]. For highly configurable software systems, such as the Linux kernel, maintaining and evolving variability information along changes to source code poses a major challenge. While source code itself may be edited, also feature-to-code mappings may be introduced, removed, or changed. In practice, such edits are often conducted ad-hoc and without proper documentation. To support the maintenance and evolution of variability, it is desirable to understand the impact of each edit on the variability. We propose the first complete and unambiguous classification of edits to variability in source code by means of a catalog of edit classes. This catalog is based on a scheme that can be used to build classifications that are complete and unambiguous by construction. To this end, we introduce a complete and sound model for edits to variability. In about 21.5 ms per commit, we validate the correctness, relevance, and suitability of our classification by classifying each edit in 1.7 million commits in the change histories of 44 open-source software systems automatically.

Keywords: software evolution, software variability, feature traceability, software product lines, mining version histories

Summary

In configurable software systems, such as the Linux kernel, certain code should only be present in certain *variants* of the software. For instance, parts of the code base may be platform dependent or a feature should only be available to a subset of customers. Maintaining and evolving variability information along changes to source code poses a major challenge for developers. One aspect thereof is keeping track of changes to variable parts of the code base that implement different *features* or feature interactions of the configurable software. While source code itself may be edited, *feature-to-code mappings* may also be introduced, removed, or changed.

Awareness of edits to variability is crucial in the development of configurable software. For instance, edits to software product lines might introduce type errors in certain variants or alter

[1] University of Ulm, Germany, paul.bittner@uni-ulm.de
[2] Siemens AG, München, Germany, christof.tinnes@siemens.com
[3] Humboldt-University of Berlin, Germany, alexander.schultheiss@informatik.hu-berlin.de
[4] University of Ulm, Germany, soeren.viegener@uni-ulm.de
[5] University of Bern, Switzerland, timo.kehrer@unibe.ch
[6] University of Ulm, Germany, thomas.thuem@uni-ulm.de

the set of available variants in an unintended way. In clone-and-own development, where each variant of a software is developed as a separate copy of the software (e. g., using branching or forking), changes to variants have to be tracked to update other variants accordingly. Variation control systems and managed clone-and-own methods inspect edits paired with information on edited features to recover knowledge about variability incrementally or to reintegrate edits to a hidden unified code base. In practice, however, the variability of the code base is often edited ad-hoc and without proper documentation.

To this end, we present a complete, unambiguous, and automatic classification of edits to variability in source code. We first introduce *variation trees* as a formalization for variability in source code. We then introduce *variation diffs* as a formalization for edits to variation trees, thus describing edits to variability in source code. We prove that variation diffs are complete and sound regarding variation trees, meaning that any possible edit to a variation tree is described by a variation diff and that every variation diff represents an actual edit to variation trees. By classifying all structures within variation diffs, we are able to classify all edits to variability in source code. We present a set of edit classes which we prove to be complete and unambiguous on variation diffs.

To validate the suitability and potential for automation of our concepts and classification, we classified about 45 million edits to source code fully automatically. In effectively 70 min, we processed about 1.7 million commits from the histories of 44 open-source software systems. 99.89% of the considered commits were processed in less than a second, making our method feasible in practical scenarios, such as continuous integration. We found that 0.2% of the patches submitted by developers contain syntactically incorrect preprocessor annotations. All other edits were classified and each of our edit classes occurs in practice.

Data Availability

The original publication is publicly accessible with the DOI 10.1145/3540250.3549108. A preprint is also available online at `https://github.com/SoftVarE-Group/Papers/raw/main/2022/2022-ESECFSE-Bittner.pdf`. Our artifact is available on Github (`https://github.com/VariantSync/DiffDetective/tree/esecfse22`) and Zenodo (DOI: 10.5281/zenodo.7110095).

Bibliography

[Bi22] Bittner, Paul Maximilian; Tinnes, Christof; Schultheiß, Alexander; Viegener, Sören; Kehrer, Timo; Thüm, Thomas: Classifying Edits to Variability in Source Code. In: Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE). ACM, New York, NY, USA, pp. 196–208, November 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 41

API Deprecation: A Systematic Mapping Study

Leif Bonorden[1], Matthias Riebisch[2]

Abstract: *This extended abstract is based on a study published at the 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2022). [BR22a]*

We conducted a systematic mapping study on API deprecation including 36 primary studies. Our analysis highlights five major gaps in research: studying remote APIs, investigating a broader range of static APIs, joining views of suppliers and clients, including humans in studies, and designing with deprecation in mind.

1 Introduction

Application Programming Interfaces (APIs) are the prevalent interaction method for software modules, components, and systems. As systems and APIs evolve, an API element may be marked as deprecated, indicating that its use is disapproved or that the feature will be removed in an upcoming version. Consequently, deprecation is a means of communication between developers and, ideally, complemented by further documentation, including suggestions for the developers of the API's clients.

Recent reports by practitioners claimed that research on APIs does not reflect the diversity of APIs in practice. In particular, research on remote APIs is demanded. [Ra21] In contrast to *static APIs* (statically linked, e.g., libraries), such *remote APIs* (e.g., REST services) are accessed via means of network communication at runtime.

2 Method

We conducted a systematic mapping study on API deprecation to classify the state of the art and identify gaps in the research field.

An initial search for `api AND deprecat*` in academic databases yielded a set of 103 unique results. Application of selection criteria and subsequent snowballing led to a final set of 36 primary studies. We evaluated these studies regarding general criteria for software engineering research (beneficiaries, type of contribution, research strategies) [St20] as well as criteria specific to API deprecation (type of API, aspect of deprecation).

[1] Universität Hamburg, Germany, leif.bonorden@uni-hamburg.de
[2] Universität Hamburg, Germany, matthias.riebisch@uni-hamburg.de

3 Results

We located five major gaps in previous research on API deprecation as opportunities for future studies:

1. **Uncharted Territory:** Deprecation of remote APIs has barely been considered.
2. **Out of Focus:** Research on the deprecation of static APIs has strongly focused on the Java programming language and the Android ecosystem.
3. **Unbridged Gap:** Suppliers and clients of an API have rarely been considered jointly.
4. **Human-out-of-the-loop:** Research strategies have been focused on data and did not include human perspectives.
5. **Prevention Better Than Cure:** Investigations have not included causes or prevention of deprecation.

Data Availability

The complete data set has been made available as open data via Zenodo [BR22b]. The files include all search results:

- Included studies with their complete classification.
- Excluded studies with the decisive exclusion criteria.
- A list of studies identified through snowballing.

References

[BR22a] Bonorden, L.; Riebisch, M.: API Deprecation: A Systematic Mapping Study. In: 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2022). IEEE, Maspalomas, Spain, pp. 451–458, 2022.

[BR22b] Bonorden, L.; Riebisch, M.: API Deprecation: A Systematic Mapping Study [Data set], Zenodo, 2022, URL: https://www.doi.org/10.5281/zenodo.5650121.

[Ra21] Raatikainen, M.; Kettunen, E.; Salonen, A.; Komssi, M.; Mikkonen, T.; Lehtonen, T.: State of the Practice in Application Programming Interfaces (APIs): A Case Study. In: 15th European Conference on Software Architecture (ECSA 2021). Springer, Växjö, Sweden, pp. 191–206, 2021.

[St20] Storey, M.-A.; Ernst, N. A.; Williams, C.; Kalliamvakou, E.: The Who, What, How of Software Engineering Research: A Socio-Technical Framework. Empirical Software Engineering 25/5, pp. 4097–4129, 2020.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 43

BXtendDSL: A layered framework for bidirectional model transformations combining a declarative and an imperative language (Summary)

Thomas Buchmann[1], Matthias Bank[2], Bernhard Westfechtel[3]

Abstract: This summary is based on an article which appeared in 2022 in The Journal of Systems & Software [BBW22].
Bidirectional transformations have been studied in a wide range of application domains. In model-driven software engineering, they are required for roundtrip engineering processes. We present a pragmatic approach to engineering bidirectional model transformations that assists transformation developers by domain-specific languages, frameworks, and code generators. A thorough evaluation demonstrates conciseness, expressiveness, and scalability of our approach.

Keywords: model-driven software engineering; round-trip engineering; bidirectional transformation

1 Summary

Bidirectional transformations (bx) occur in different application domains, including e.g. databases, programming languages, and software engineering [Cz09]. Programming bidirectional transformations in a conventional programming language is both laborious and error-prone: Both transformation directions have to be programmed separately, and consistency of forward and backward transformations has to be checked by testing.

In response to these problems, a wide variety of bx approaches (functional, relational, or grammar-based) have been developed in research [Hi16]. A recurring theme driving bx research are *roundtrip properties*, also referred to as *bx laws*. Roundtrip properties are constraints on the interplay of forward and backward transformations. The goal of many bx approaches consists in the construction of bidirectional transformations that are *provably correct* with respect to roundtrip properties.

However, *empirical evaluations* [An20] demonstrate limitations of bx approaches with respect to *expressiveness*, i.e., the capability to solve a given bx problem. These limitations follow from the conditions that transformations have to satisfy in order to guarantee roundtrip

[1] Faculty of Computer Science, Deggendorf Institute of Technology, Dieter-Görlitz-Platz 1, 94469 Deggendorf, Germany thomas.buchmann@th-deg.de
[2] Chair of Applied Computer Science I, University of Bayreuth, Universitätsstraße 30, 95440 Bayreuth, Germany matthias.bank@uni-bayreuth.de
[3] Chair of Applied Computer Science I, University of Bayreuth, Universitätsstraße 30, 95440 Bayreuth, Germany bernhard.westfechtel@uni-bayreuth.de

properties. Additional shortcomings were observed with respect to *conciseness* — the ability to provide for short solutions with respect to size metrics — and *scalability* — the ability to perform transformations efficiently on large data sets.

These observations motivated the development of *BXtendDSL* [BBW22], a framework for engineering bidirectional model transformations. BXtendDSL combines domain-specific languages (DSL) for bidirectional model transformations that are located at different levels of abstractions. A declarative language serves to specify a bidirectional transformation concisely. Round-trip properties are guaranteed as long as the transformation specification conforms to well-behavedness conditions. Intentionally, the declarative language is computationally incomplete, i.e., it is usually not possible to completely specify a bidirectional transformation at the declarative level. Therefore, the declarative language provides extension points for adding imperative code, which is written in an internal DSL.

We evaluated BXtendDSL with respect to expressiveness, conciseness, and scalability and compared it against competing bx approaches with a number of benchmarks, using the Benchmarx framework [An20]. Our evaluations demonstrate expressiveness (all test cases were passed), conciseness (short solutions were provided), and scalability (the implementation is scalable to large model sizes).

Data Availability

The software and the benchmarks are publicly available; see the instructions given at the end of the journal paper [BBW22].

Bibliography

[An20] Anjorin, Anthony; Buchmann, Thomas; Westfechtel, Bernhard; Diskin, Zinovy; Ko, Hsiang-Shang; Eramo, Romina; Hinkel, Georg; Samimi-Dehkordi, Leila; Zündorf, Albert: Benchmarking bidirectional transformations: theory, implementation, application, and assessment. Software and Systems Modeling, 19(3):647–691, 2020.

[BBW22] Buchmann, Thomas; Bank, Matthias; Westfechtel, Bernhard: BXtendDSL: A layered framework for bidirectional model transformations combining a declarative and an imperative language. J. Syst. Softw., 189:111288, 2022.

[Cz09] Czarnecki, Krzysztof; Foster, J. Nathan; Hu, Zhenjiang; Lämmel, Ralf; Schürr, Andy; Terwilliger, James F.: Bidirectional Transformations: A Cross-Discipline Perspective. In (Paige, Richard F., ed.): Proceedings of the Second International Conference on Theory and Practice of Model Transformations (ICMT 2009). volume 5563 of Lecture Notes in Computer Science, Springer-Verlag, Zurich, Switzerland, pp. 260–283, June 2009.

[Hi16] Hidaka, Soichoro; Tisi, Massimo; Cabot, Jordi; Hu, Zhenjiang: Feature-Based Classification of Bidirectional Transformation Approaches. Software and Systems Modeling, 15(3):907–928, July 2016.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 45

iStar-Erweiterung für kollaborative cyber-physische Systeme

Marian Daun[1], Jennifer Brings[2], Lisa Krajinski[2], Viktoria Stenkova[2], Torsten Bandyszak[2]

Abstract: Dieser Vortrag berichtet von dem Beitrag *A GRL-compliant iStar extension for collaborative cyber-physical systems* [Da21], der im Februar 2021 in der Fachzeitschrift *Requirements Engineering* erschienen ist.

Keywords: Requirements Engineering; Collaborative Cyber-physical Systems; Goal Modeling; iStar; GRL

1 Einleitung

Kollaborative cyber-physische Systeme sind in der Lage, sich zu Systemverbünden zusammenzuschließen, in denen sie ihr Verhalten so koordinieren, dass sie Ziele erfüllen, die die einzelnen Systeme nicht erreichen können [Br20]. Die explizite Modellierung von Systemverbundzielen erlaubt unter anderem die Relationierung zu Systemkonfigurationen und bietet damit einen Mehrwert für die Entwicklung dynamischer kollaborativer Systeme, die ihr Verhalten abhängig von den mit ihnen kollaborierenden Systemen adaptieren [Br19]. Die komplexen Abhängigkeiten kollaborativer cyber-physischer Systeme sind jedoch mit existierenden Zielmodellierungsansätzen schwer zu erfassen [Da19].

2 GRL-konforme iStar-Erweiterung

Hierzu stellt der Beitrag eine Erweiterung der weit verbreiteten Zielmodellierungssprache iStar vor, die die Besonderheiten kollaborativer cyber-physischer Systeme und die Bedürfnisse ihrer Entwickler berücksichtigt. Die Erweiterung ist so konzipiert, dass sie konform zu der von der ITU (International Telecommunication Union) standardisierten Goal-oriented Requirement Language (GRL, [IT18]) ist. Neben der erhöhten industriellen Akzeptanz durch Standardisierung geht hiermit auch eine Vereinfachung der Sprache einher, da sich die GRL auf die Kernkonstrukte von iStar beschränkt. Dies führt zu einer leichteren Anwendbarkeit in der industriellen Praxis.

[1] Technische Hochschule Würzburg-Schweinfurt, Center Robotics, Schweinfurt, marian.daun@thws.de
[2] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, Essen, {jennifer.brings, lisa.krajinski, viktoria.stenkova, torsten.bandyszak}@uni-due.de

Insbesondere bietet die Erweiterung Unterstützung durch die explizite Unterscheidung zwischen den Zielen der einzelnen kollaborativen cyber-physischen Systeme und des Verbunds. Dies erlaubt es, Abhängigkeiten zwischen den einzelnen kollaborativen cyber-physischen Systemen und zwischen den einzelnen Systemen und dem Verbund darzustellen und zu untersuchen. Unter anderem können wechselseitige Abhängigkeiten zwischen den Zielen der Einzelsysteme und den Verbundzielen bestehen.

Die Vorteile der Erweiterung für die Zielmodellierung von kollaborativen cyber-physischen Systemen wurden an zwei Fallstudien aus verschiedenen Industriedomänen illustriert. Ein wesentlicher Vorteil ist die Reduzierung der Modellkomplexität durch die in der Erweiterung eingeführten Modellelemente.

3 Data Availability

Zur werkzeugtechnischen Unterstützung der entwickelten Erweiterung wurden Stencils für Microsoft Visio erstellt. Diese Stencils sind frei über die folgende Webseite verfügbar: `https://doi.org/10.6084/m9.figshare.13313093`.

Literaturverzeichnis

[Br19] Brings, Jennifer; Daun, Marian; Bandyszak, Torsten; Stricker, Vanessa; Weyer, Thorsten; Mirzaei, Elham; Neumann, Martin; Zernickel, Jan Stefan: Model-based documentation of dynamicity constraints for collaborative cyber-physical system architectures: Findings from an industrial case study. J. Syst. Archit., 97:153–167, 2019.

[Br20] Brings, Jennifer; Daun, Marian; Weyer, Thorsten; Pohl, Klaus: Goal-based configuration analysis for networks of collaborative cyber-physical systems. In (Hung, Chih-Cheng; Cerný, Tomás; Shin, Dongwan; Bechini, Alessio, Hrsg.): SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020. ACM, S. 1387–1396, 2020.

[Da19] Daun, Marian; Stenkova, Viktoria; Krajinski, Lisa; Brings, Jennifer; Bandyszak, Torsten; Weyer, Thorsten: Goal modeling for collaborative groups of cyber-physical systems with GRL: reflections on applicability and limitations based on two studies conducted in industry. In (Hung, Chih-Cheng; Papadopoulos, George A., Hrsg.): Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019. ACM, S. 1600–1609, 2019.

[Da21] Daun, Marian; Brings, Jennifer; Krajinski, Lisa; Stenkova, Viktoria; Bandyszak, Torsten: A GRL-compliant iStar extension for collaborative cyber-physical systems. Requir. Eng., 26(3):325–370, 2021.

[IT18] ITU International Telecommunication Union: Recommendation ITU-T Z.151: User Requirements Notation (URN). Bericht, 2018.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 47

Model-based Requirements Engineering in Industry: Experiences from Training and Application

Marian Daun[1], Jennifer Brings[2], Marcel Goger[3], Walter Koch[3], Thorsten Weyer[4]

Abstract: In this talk, we report on our findings from the paper *Teaching Model-Based Requirements Engineering to Industry Professionals: An Experience Report* [Da21a], which received the best paper award of the *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET)* in 2021.

Keywords: Requirements Engineering; Software Engineering Education; Industrial Training

Introduction and Summary

In several consecutive joint research projects, an extensive industry-ready methodology for model-based software engineering has been developed [Bö16, Bö21]. As part of this undertaking, we developed a model-based requirements engineering approach tailored for industry's needs. In a technology transfer project, we developed a training program for model-based requirements engineering and applied model-based requirements engineering in industrial practice [Da17, TD19]. In this talk, we report on insights from (a) teaching model-based requirements engineering and from (b) applying model-based requirements engineering in a productive development project.

Model-based Requirements Engineering. The model-based requirements engineering approach is based upon different building blocks: Properly defining and modeling the requirements engineering context [Da16], defining high-level goals of the system using a specifically designed profile of the goal-oriented requirements language to derive complex interrelations of collaborative cyber-physical systems [Da21b], subsequently refining and detailing these models with the use of scenario descriptions and later on the definition of detailed system requirements from different perspectives.

Teaching Approach. Besides teaching technicalities of the model-based approach, the course also emphasizes skills for working with requirements models. Among others, the participants learn how to create requirements models and how modeling decisions impact

[1] Technische Hochschule Würzburg-Schweinfurt, Center Robotics, Schweinfurt, marian.daun@thws.de
[2] Universität Duisburg-Essen, Essen, jennifer.brings@uni-due.de
[3] Schaeffler AG, Herzogenaurach, marcel.goger@schaeffler.com,walter.koch@schaeffler.com
[4] Technische Hochschule Mittelhessen, Gießen, thorsten.weyer@mni.thm.de

the understanding of requirements. Other topics include the evolution of requirements models due to constantly changing requirements, the refinement of requirements models into design artifacts, and validation of requirements models and their proper use to facilitate verification (cf. [DWP19]).

Findings from Industry Application. Based on our experience, we provide guidelines for educators designing requirements engineering courses for industry professionals. We found nine important aspects to consider when teaching model-based requirements engineering to industry professionals. In addition, we report on findings regarding relevance, practicality, and use of model-based requirements engineering for system development.

Data Availability

Teaching materials have been created in the context of the SPEDiT project (funded by BMBF) and are available at `https://spedit.informatik.tu-muenchen.de/results.html`.

Literaturverzeichnis

[Bö16] Böhm, W; Daun, M; Koutsoumpas, V; Vogelsang, A; Weyer, T: SPES XT Modeling Framework. In: Advanced Model-Based Engineering of Embedded Systems, S. 29–42. 2016.

[Bö21] Böhm, B; Böhm, W; Daun, M; Hayward, A; Kranz, S; Regnat, N; Schröck, S; Stierand, I; Vogelsang, A; Vollmar, J; Voss, S; Weyer, T; Wortmann, A: Engineering of Collaborative Embedded Systems. In: Model-Based Engineering of Collaborative Embedded Systems, S. 15–48. 2021.

[Da16] Daun, M; Tenbergen, B; Brings, J; Weyer, T: SPES XT Context Modeling Framework. In: Advanced Model-Based Engineering of Embedded Systems, S. 43–57. 2016.

[Da17] Daun, M; Brings, J; Aluko Obe, P; Pohl, K; Moser, S; Schumacher, H; Rieß, M: Teaching Conceptual Modeling in Online Courses: Coping with the Need for Individual Feedback to Modeling Exercises. In: IEEE Conf. Softw. Eng. Edu. and Training. S. 134–143, 2017.

[Da21a] Daun, M; Brings, J; Goger, M; Koch, W; Weyer, T: Teaching Model-Based Requirements Engineering to Industry Professionals: An Experience Report. In: 43rd IEEE/ACM Int. Conf. Softw. Eng.: Softw. Eng. Edu. and Training, ICSE (SEET) 2021. S. 40–49, 2021.

[Da21b] Daun, M; Brings, J; Krajinski, L; Stenkova, V; Bandyszak, T: A GRL-compliant iStar extension for collaborative cyber-physical systems. Requir. Eng., 26(3):325–370, 2021.

[DWP19] Daun, M; Weyer, T; Pohl, K: Improving manual reviews in function-centered engineering of embedded systems using a dedicated review model. Softw. Syst. Model., 18(6):3421–3459, 2019.

[TD19] Tenbergen, B; Daun, M: Industry Projects in Requirements Engineering Education: Application in a University Course in the US and Comparison with Germany. In: 52nd Hawaii Int. Conf. System Sciences. S. 1–10, 2019.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 49

Zuverlässigkeit studentischer Selbsteinschätzungen zur Vorhersage der Leistung im Software Engineering

Marian Daun[1], Jennifer Brings[2], Patricia Aluko Obe[2], Viktoria Stenkova[2]

Abstract: Dieser Vortrag berichtet von dem Beitrag *Reliability of Self-Rated Experience and Confidence as Predictors for Students' Performance in Software Engineering* [Da21], der in der Fachzeitschrift *Empirical Software Engineering* erschienen ist.

Keywords: Student performance; Self-rated Experience; Confidence; Model comprehension

1 Einleitung

Sowohl in der Software-Engineering-Forschung als auch in der Software-Engineering-Lehre werden Studierende oft in Gruppen eingeteilt. Dahinter steht häufig der Wunsch, entweder möglichst homogene Gruppen oder möglichst heterogene Gruppen bezogen auf die Leistung der Studierenden zu bilden – bspw. um eine faire Verteilung von Treatment und Control zu erreichen. Existierende Forschungsarbeiten haben gezeigt, dass bei Programmieraufgaben Selbsteinschätzungen der Erfahrung von Studierenden gute Indikatoren für deren Leistungen sind. Im Gegensatz dazu belegen die Ergebnisse unserer Studie, dass dies nicht auf alle Software-Engineering-Bereiche übertragbar ist.

2 Leistungsvorhersage mittels Selbsteinschätzung von Erfahrung und Konfidenz

Wir haben untersucht, inwiefern selbst eingeschätzte Erfahrung und Konfidenz zur Prognose der Leistung im Modellverständnis genutzt werden können. Basierend auf Daten aus vier Experimenten [DBW20, Da19, DWP19, DBW17] zum Thema Modellwirkung konnten wir zeigen, dass viele Studierende Schwierigkeiten bei der Selbsteinschätzung haben. In den Experimenten mussten die Studierenden entscheiden, ob ein natürlichsprachlich beschriebener Sachverhalt in einem gezeigten Modell dargestellt ist oder nicht. Bei jeder Antwort sollten die Studierenden auf einer fünfstufigen Skala angeben, wie sicher sie sich sind, dass die gegebene Antwort richtig ist. Die Ergebnisse zeigen zwar einen statistisch

[1] Technische Hochschule Würzburg-Schweinfurt, Center Robotics, Schweinfurt, marian.daun@fhws.de
[2] Universität Duisburg-Essen, Essen, {jennifer.brings, patricia.aluko-obe, viktoria.stenkova}@uni-due.de

signifikanten Unterschied in der Selbsteinschätzung zwischen richtigen und falschen Antworten, jedoch weisen die Selbsteinschätzungen zwischen richtigen und falschen Antworten auch einen starken Zusammenhang auf. D. h., Studierende, die bei richtigen Antworten eine hohe Selbstsicherheit angegeben haben, haben auch bei falschen Antworten eine hohe Selbstsicherheit angegeben. Gleiches gilt für Studierende, die sich eine niedrige Selbstsicherheit attestieren.

Nach der Beantwortung der Fragen zu den Modellen sollten die Studierenden ihre Erfahrung in Bezug auf relevante Themenbereiche (z. B. modellbasierte Entwicklung allgemein oder die jeweiligen Modellierungssprachen) auf einer fünfstufigen Skala angeben. Die Selbsteinschätzung der Erfahrung hat sich als noch schlechtere Vorhersage für die Leistung der Studierenden erwiesen. Die statistische Auswertung ergab, dass kein Zusammenhang zwischen der Leistung des Studierenden und dessen Selbsteinschätzung besteht.

Daher muss die Nutzung der Selbsteinschätzungen von Studierenden bei Forschung und Lehre im Bereich konzeptueller Modellierung hinterfragt werden.

3 Data Availability

Die Fragebögen und die erhobenen Daten können über die originalen Veröffentlichungen, bzw. im jeweiligen Online-Supplement zur Veröffentlichung eingesehen werden.

Literaturverzeichnis

[Da19] Daun, Marian; Brings, Jennifer; Krajinski, Lisa; Weyer, Thorsten: On the Benefits of using Dedicated Models in Validation Processes for Behavioral Specifications. In: 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP). S. 44–53, 2019.

[Da21] Daun, Marian; Brings, Jennifer; Aluko Obe, Patricia; Stenkova, Viktoria: Reliability of self-rated experience and confidence as predictors for students' performance in software engineering. Empir. Softw. Eng., 26(4):80, 2021.

[DBW17] Daun, Marian; Brings, Jennifer; Weyer, Thorsten: On the Impact of the Model-Based Representation of Inconsistencies to Manual Reviews - Results from a Controlled Experiment. In: Conceptual Modeling - 36th International Conference, ER. Jgg. 10650 in Lecture Notes in Computer Science. Springer, S. 466–473, 2017.

[DBW20] Daun, Marian; Brings, Jennifer; Weyer, Thorsten: Do Instance-Level Review Diagrams Support Validation Processes of Cyber-Physical System Specifications: Results from a Controlled Experiment. In: Proceedings of the International Conference on Software and System Processes. ICSSP '20, Association for Computing Machinery, New York, NY, USA, S. 11–20, 2020.

[DWP19] Daun, Marian; Weyer, Thorsten; Pohl, Klaus: Improving manual reviews in function-centered engineering of embedded systems using a dedicated review model. Softw. Syst. Model., 18(6):3421–3459, 2019.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 51

It's Your Loss: Classifying Information Loss During Variability Model Roundtrip Transformations

Kevin Feichtinger[1], Chico Sundermann[2], Thomas Thüm[3], Rick Rabiser[4]

Abstract: This is a summary of a paper (with the same title) originally published at the 26th ACM International Systems and Software Product Line Conference (SPLC) in 2022 discussing the information loss occurring when transforming variability models.

Keywords: Software product lines; variability modeling; variability model transformations; information loss.

1 Summary

Diverse variability modeling approaches have been developed to explicitly capture the commonalities and variability of a set of software systems. Since variability modeling approaches differ especially in terms of scope and expressiveness, it is difficult to assess their properties and find the right approach for a specific use case. Transforming variability models, i.e., of one type to another, would help to better understand and compare existing approaches and would also enable users to switch between approaches. Unfortunately, due to differences in scope and expressiveness, it is difficult to implement transformations without information loss. We analyzed concrete variability modeling approaches, presented a mapping of key concepts between them, and identified and classified the information lost in one-way and round-trip transformations [Fe22]. We evaluated the applicability of our information loss classes by transforming different models of varying sizes and complexity using an existing implementation of transformations[5]. Our information loss classes contribute to a better understanding of different variability modeling approaches, simplify the comparability, and allow users to grasp the impact of transformations.

2 Information Loss Classes

We identified *Conceptual Losses*, i.e., regarding *Configurability* and *Semantics*, as well as *Structural Losses* as distinct information loss classes. *No Information Loss* classifies fully supported transformations [Fe22].

[1] LIT CPS Lab, Johannes Kepler University Linz, Linz, Austria kevin.feichtinger@jku.at
[2] University of Ulm, Ulm, Germany chico.sundermann@uni-ulm.de
[3] University of Ulm, Ulm, Germany thomas.thuem@uni-ulm.de
[4] CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz, Linz, Austria rick.rabiser@jku.at
[5] `https://github.com/SECPS/TraVarT`

No Information Loss: An entity or a relationship, e.g., an optional feature, can be transformed into another entity or relationship of the target notation with the same semantics, e.g., a Boolean decision, and, hence, can be restored during the round-trip at its full capacity.

Structural Loss: The configuration space of both, the original and transformed model, remains identical, though they are differing in structure. Structure changes because an entity or a relationship, e.g., a conjunction constraint, may not be transformed into an identical entity or relationship, e.g., an implies constraint. This in turn also causes additional entities and relationships in the round-trip model.

Semantic Loss: An entity or relationship could not be transformed into the target language because it is not supported in the target approach, e.g., commonalities in feature models are out of scope for decision models. The configuration space of the original and the transformed model still remains identical.

Configurability Loss: An entity or relationship could not be transformed into the target language because it is not supported in the target approach, e.g., numerical or textual values of decision models cannot be represented as features in feature models. This changes the configuration space of the resulting (target/round-trip) model.

3 Data Availability

We investigated and evaluated the applicability of our information loss classes [Fe22] using a set of existing variability models of different types[6,7] . Specifically, we transformed 31 feature models, 6 DOPLER decision models, and 3 OVM models into each other.

4 Conclusion

Our evaluation [Fe22] showed that the defined information loss classes can be found in transformed variability models. The classes indicate how information is lost during one-way transformations and round-trip transformations and can therefore help modelers to investigate transformed variability models, increase their awareness of information loss and guide them when developing new transformations.

Literaturverzeichnis

[Fe22] Feichtinger, Kevin; Sundermann, Chico; Thüm, Thomas; Rabiser, Rick: It's Your Loss: Classifying Information Loss during Variability Model Roundtrip Transformations. In: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A. SPLC '22, ACM, New York, NY, USA, S. 67–78, 2022.

[6] https://github.com/Universal-Variability-Language/uvl-models

[7] https://github.com/FeatureIDE/FeatureIDE/tree/v3.8.1/featuremodels

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 53

Explainable Online Reinforcement Learning for Adaptive Systems

Felix Feit[1], Andreas Metzger[2], Klaus Pohl[3]

Abstract: This talk presents our work on explainable online reinforcement learning for self-adaptive systems published at the 3rd IEEE Intl. Conf. on Autonomic Computing and Self-Organizing Systems.

Keywords: Adaptation; Reinforcement Learning; Explainability; Interpretability

1 Presentation Summary

An adaptive system can automatically maintain its requirements in the presence of dynamic environment changes. Developing an adaptive system is difficult due to design time uncertainty, because how the environment will change at runtime and what precise effects adaptations will have on the running system are typically unknown at design time [We20].

Online reinforcement learning, i.e., employing reinforcement learning (RL) at runtime, is an emerging approach to realize self-adaptive systems in the presence of design time uncertainty. Online RL learns via actual operational data and thereby leverages feedback only available at runtime [Me22].

Deep RL algorithms represent the learned knowledge as a neural network. Compared with classical RL algorithms, Deep RL algorithms offer important benefits for adaptive systems. Deep RL can generalize over unseen inputs, it can handle continuous environment states and adaptation actions, and it can readily capture concept and data drifts [PMP20]. Yet, a fundamental problem of Deep RL is that learned knowledge is not represented explicitly. For a human, it is practically impossible to relate neural network parameters to concrete RL decisions. Figure 1 illustrates this problem by comparing how knowledge is represented.

Understanding RL decisions is key to (1) increase trust, and (2) facilitate debugging. Debugging is especially relevant for adaptive systems, because the reward function, which quantifies the feedback to the RL algorithm, must explicitly defined by developers, thus introducing a source for human error.

We introduce XRL-DINE to make Deep RL decisions for self-adaptive systems explainable [FMP22]. XRL-DINE enhances and combines explainable RL techniques from machine

[1] paluno, University of Duisburg-Essen, Essen, Germany, f.m.feit@gmail.com
[2] paluno, University of Duisburg-Essen, Essen, Germany, andreas.metzger@paluno.uni-due.de
[3] paluno, University of Duisburg-Essen, Essen, Germany, klaus.pohl@paluno.uni-due.de

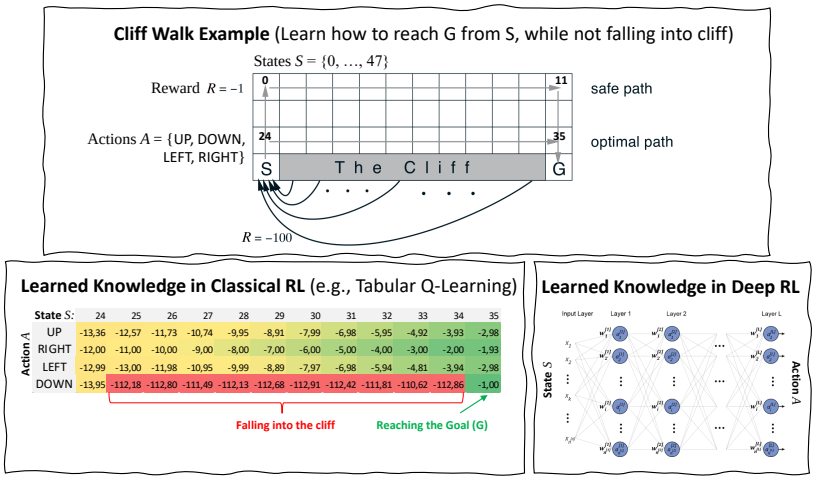


Abb. 1: Illustration of how learned knowledge is represented for Cliff Walk example from [SB18].

learning research. We present a proof-of-concept implementation of XRL-DINE, as well as qualitative and quantitative results that demonstrate the usefulness of XRL-DINE.

Data Availability. Source code and experimental data are available from `https://git.uni-due.de/rl4sas/xrl-dine`. The submission version of the original paper is available from `https://arxiv.org/abs/2210.05931`.

Literaturverzeichnis

[FMP22] Feit, Felix; Metzger, Andreas; Pohl, Klaus: Explaining Online Reinforcement Learning Decisions of Self-Adaptive Systems. In (Di Nitto, Elisabetta; Gerostathopoulos, Ilias; Bellman, Kirstie; Tomforde, Sven, Hrsg.): IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2022, Virtual, September 19-23, 2022. IEEE, S. 51–60, 2022.

[Me22] Metzger, Andreas; Quinton, Clément; Mann, Zoltán Ádám; Baresi, Luciano; Pohl, Klaus: Realizing Self-Adaptive Systems via Online Reinforcement Learning and Feature-Model-guided Exploration. Computing, 2022.

[PMP20] Palm, Alexander; Metzger, Andreas; Pohl, Klaus: Online Reinforcement Learning for Self-adaptive Information Systems. In (Dustdar, Schahram; Yu, Eric; Salinesi, Camille; Rieu, Dominique; Pant, Vik, Hrsg.): 32nd International Conference on Advanced Information Systems Engineering, CAiSE 2020, Grenoble, France, June 8-12, 2020. Jgg. 12127 in LNCS. Springer, S. 169–184, 2020.

[SB18] Sutton, Richard S; Barto, Andrew G: Reinforcement learning: An introduction. MIT press, 2018.

[We20] Weyns, Danny: An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective. John Wiley & Sons, 2020.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 55

Precomputing Reconfiguration Strategies based on Stochastic Timed Game Automata

Hendrik Göttmann[1], Birte Caesar[2], Lasse Beers[3], Malte Lochau[4], Andy Schürr[5], Alexander Fay[6]

Abstract: We summarize our paper *Precomputing Reconfiguration Strategies based on Stochastic Timed Game Automata* which has been published in the proceedings of the *25th International Conference on Model Driven Engineering Languages and Systems (MODELS 2022)*.

Keywords: Stochastic Timed Game Automata; Proactive Self-Adaptation; Strategy Synthesis; Statistical Model-Checking

1 Summary

Many recent application domains require software-intense systems with reconfiguration capabilities to be (self-)adaptable to changing contextual situations. As an example, in today's aircraft manufacturing assembly lines, human engineers are supported by mobile robots, offering production capabilities (e.g., mountable toolsets) on demand. Depending on the manufacturing context, requirements on capabilities may change (e.g., from riveting at construction site A to welding at site B) thus requiring robots to repeatedly reconfigure themselves to new contextual situations. Knowledge about occurrences of these contextual situations is only partially available at design time as this information only becomes apparent at runtime. Hence, entirely pre-planning these reconfiguration decisions is infeasible due to the large state space and the high degree of uncertainty about the expected runtime behavior. In contrast, making these highly-complex reconfiguration decisions at runtime may benefit from perfect context knowledge. However, this excludes all non-trivial decision algorithms except greedy-based heuristics as both computing resources and time for decision-making are usually limited at runtime. Furthermore, in addition to functional properties also non-functional properties such as real-time constraints have to be considered, too. For instance, the robot movement, toolset reconfiguration and computing the reconfiguration decision itself each require a certain amount of time ultimately influencing the throughput of the overall production process. Again, uncertainty is an omnipresent issue.

[1] Technical University of Darmstadt, RT Systems Lab, Darmstadt, DE hendrik.goettmann@es.tu-darmstadt.de
[2] Helmut Schmidt University, Institute of Automation Technology, Hamburg, DE birte.caesar@hsu-hh.de
[3] Helmut Schmidt University, Institute of Automation Technology, Hamburg, DE lasse.beers@hsu-hh.de
[4] University of Siegen, Model-based Engineering Group, Siegen, DE malte.lochau@uni-siegen.de
[5] Technical University of Darmstadt, RT Systems Lab, Darmstadt, DE andy.schuerr@es.tu-darmstadt.de
[6] Helmut Schmidt University, Institute of Automation Technology, Hamburg, DE alexander.fay@hsu-hh.de

To cope with these challenges, we rely on concepts from game theory. Both the system and the context act as opponents in a two-player game derived from our reconfiguration model. Our reconfiguration model combines (context) feature models describing the configuration space with a domain specific constraint language providing means to define real-time constraints on reconfigurations (i.e., changing from one configuration of the feature model to another) which cannot be encoded into plain feature models. Additionally, the constraint language supports the specification of stochastic delays given as probability distributions instead of exact values for time bounds as those are unknown at design time. From this reconfiguration specification (i.e., feature model and real-time constraints) we automatically construct a stochastic timed game automaton exactly representing the specified behavior. Based on this game-theoretic reconfiguration model, we are able to precompute winning strategies by means of UPPAAL STRATEGO which enable the system player to make fast look-ups at runtime for presumably best-fitting reconfiguration decisions satisfying the context player. Statistical model-checking further enables us to optimize the strategy w.r.t. non-functional properties like real-time behavior. To summarize, our approach [Gö22] makes the following contributions:

- integrated modeling of real-time reconfigurations comprising uncertain context behaviors based on context feature models,
- application of game theory to synthesize both safe and optimized reconfiguration strategies by means of UPPAAL STRATEGO, and
- investigation of efficiency/effectiveness trade-offs by considering a real-world example of a reconfigurable robot support system for the construction of aircraft fuselages.

2 Data Availability

To foster reproducibility, we provide the tool implementation and the evaluation data of our real-world example online[7].

3 Acknowledgements

This work has been funded by the German Research Foundation (DFG) as part of project A4 within the Collaborative Research Center (CRC) 1053 MAKI.

Bibliography

[Gö22] Göttmann, Hendrik; Caesar, Birte; Beers, Lasse; Lochau, Malte; Schürr, Andy; Fay, Alexander: Precomputing Reconfiguration Strategies based on Stochastic Timed Game Automata. In (Syriani, Eugene; Sahraoui, Houari A.; Bencomo, Nelly; Wimmer, Manuel, eds): Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022. ACM, pp. 31–42, 2022.

[7] https://doi.org/10.5281/zenodo.6962663

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 57

Requirements document relations: A reuse perspective on traceability through standards

Katharina Großer[1], Volker Riediger[1], Jan Jürjens[1,2]

Abstract: Our publication in the *Software and Systems Modeling* Journal 2022 started by observing that relations of views, like requirements documents, are scarcely considered in requirements traceability, despite being a key factor in requirements reuse. Explicit formalized document relations can facilitate review activities to improve consistency and completeness. This is relevant for projects, e.g., in the aerospace sector, with challenges related to complex document dependencies: 1. Several contractors contribute. 2. Requirements from standards are applied in several projects. 3. Reuse of requirements from previous phases. We exploit the concept of "layered traceability", explicitly considering documents as views on sets of requirements with relations between these different representations. Different types of relations and their dependencies are investigated with a special focus on requirement reuse through standards and findings formalized in an Object-Role Modelling (ORM) conceptual model. Automated analyses of requirement graphs based on this model are able to reveal inconsistencies in document integration. We show such queries in Neo4J/Cypher for the EagleEye case study. This work is a step toward better support to handle highly complex requirement document dependencies in large projects with a special focus on requirements reuse and to enable automated quality checks on dependent documents to facilitate requirements reviews.

Keywords: Requirement Document Relations; Requirement Reuse; Standards; Space Engineering Requirements; ECSS; Traceability

1 Summary

Relations between requirements have been described intensively in past research and are part of nearly every requirements engineering approach, although in practice, there is little support of specific relationships in common requirements engineering tools [GC17], even though there is a need for high-end traceability, e.g., in highly standardized and safety critical embedded systems domains, such as automotive, defense, or aerospace. There exist different relations on different levels of representation granularity, which interact for composite artifacts, such as documents [GF95]. Yet, relations between requirements documents, are rarely considered. This is remarkable, as they are a key factor in requirement reuse [Cas+10]. Re-occurring structures, e.g., specific sections or information on project characteristics, provide *abstraction*. Standards or reference specifications are designed to be reused, making this explicit. Yet, there is a lack of structured processes for *integration*. The goal of our paper [GRJ22] is to handle highly complex requirement document dependencies

[1] University of Koblenz, Universitätsstr. 1, 56070 Koblenz, Germany, {grosser|riediger|juerjens}@uni-koblenz.de
[2] Fraunhofer ISST, Emil-Figge-Straße 91, 44227 Dortmund, Germany

better in large projects to facilitate reviews and structured reuse by enabling automated quality checks on these dependent documents.

To achieve this, we collected knowledge from expert interviews, requirements engineering literature and guidelines, as well as analysis of specification documents from different space engineering projects. Resulting from an ontological analysis, we define a conceptual model. It provides a reference frame to define dependent relations on different layers, following the idea of "layered traceability" [GF95], with a special focus on its applicability to different document reuse scenarios observed in space engineering, in particular the application of standards [ECSSS00]. Documents are considered as *views* on sets of requirements with traceability relations on their own. Results are formalized as a conceptual model using the *Object-Role Modelling* (ORM) notation. The *EagleEye* case study, a virtual earth observation mission, is used to implement a proof of concept in *Neo4J* with *Cypher* queries that detect completeness and consistency issues. Future work should focus on implementing a practical tool to validate the framework in practice and enable semantic interoperability between different tools used through the overall system development and operations life-cycle and support advanced requirements reuse.

2 Data Availability

The EagleEye Neo4J trace graph and the Cypher queries presented in the paper are available under `https://uni-ko-ld.de/requirementrelations`.

Bibliography

[Cas+10] Verónica Castañeda et al. "The use of ontologies in requirements engineering". In: *Global journal of researches in engineering* 10.6 (Nov. 2010), pp. 2–8.

[ECSSS00] ECSS Secretariat. *ECSS system - Description, implementation and general requirements*. ECSS-S-ST-00C (ECSS). July 31, 2008.

[GC17] Hélène Gaspard-Boulinc and Stéphane Conversy. "Usability Insights for Requirements Engineering Tools: A User Study with Practitioners in Aeronautics". In: *25th IEEE International Requirements Engineering Conference (RE'17)*. 2017, pp. 223–232.

[GF95] Orlena C. Z. Gotel and Anthony C. W. Finkelstein. "Contribution structures [Requirements artifacts]". In: *2nd IEEE International Symposium on Requirements Engineering*. 1995, pp. 100–107.

[GRJ22] Katharina Großer, Volker Riediger, and Jan Jürjens. "Requirements document relations". In: *Software and Systems Modeling* 21 (6 2022). Theme Section Paper, pp. 1–37. DOI: 10.1007/s10270-021-00958-y.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 59

Benchmarking Scalability of Cloud-Native Applications

Sören Henning[1] Wilhelm Hasselbring[2]

Abstract: This contribution has been published in the journal *Empirical Software Engineering* (Springer Nature) in 2022 [HH22], https://doi.org/10.1007/s10664-022-10162-1.

Keywords: Scalability; Benchmarking; Performance engineering; Cloud-Native

1 Introduction

Software architectures [Ha18] significantly influence the quality characteristics of the resulting software systems. Scalability is such a quality characteristic that is in particular relevant for data stream processing systems [HH20; HH21b; HHM19]. In the context of analyzing IoT sensor data [HWD21], we study scalable architectures for power consumption monitoring [He21]. Cloud-native applications constitute a recent trend for designing large-scale software systems, with a focus on *scalability* (https://www.cncf.io/).

In this paper, we present the Theodolite benchmarking method, allowing researchers and practitioners to conduct empirical scalability evaluations of cloud-native applications, frameworks, and deployment options. Although scalability is often mentioned as a key driver for adopting cloud-native architectures and microservices [KH19], we found that research is lacking a commonly accepted method to empirically assess and compare the *scalability* of cloud-native applications. In empirical software engineering, benchmarks are used as a measuring instrument for comparing different technologies or configurations [Ha21]. Thus, we designed the Theodolite method for benchmarking scalability of cloud-native applications [HH22]. Our benchmarking method consists of scalability metrics [HH21a], measurement methods, and an architecture for a scalability benchmarking framework, particularly suited for cloud-native applications. To balance usability and reproducibility, our benchmarking method provides configuration options, controlling the trade-off between overall execution time and statistical grounding. We performed an extensive experimental evaluation of our method's configuration options for data stream processing applications. We find that, independent of the cloud platform, it only takes a few repetitions (≤ 5) and short execution times (≤ 5 minutes) to assess whether SLOs are achieved. Combined with our findings from evaluating different search strategies, we conclude that our method allows to benchmark scalability in reasonable time.

[1] Kiel University, Software Engineering Group, 24098 Kiel, Germany soeren.henning@email.uni-kiel.de
[2] Kiel University, Software Engineering Group, 24098 Kiel, Germany hasselbring@email.uni-kiel.de

Data Availability A replication package is available at Zenodo (`https://doi.org/10.5281/zenodo.5596982`). The source code is available at GitHub (`https://github.com/cau-se/theodolite`) and the software documentation at `https://www.theodolite.rocks/`.

Literatur

[Ha18] Hasselbring, W.: Software Architecture: Past, Present, Future. In: The Essence of Software Engineering. Springer, S. 169–184, 2018.

[Ha21] Hasselbring, W.: Benchmarking as Empirical Standard in Software Engineering Research. In: International Conference on Evaluation and Assessment in Software Engineering (EASE 2021). ACM, S. 365–372, Juni 2021.

[He21] Henning, S.; Hasselbring, W.; Burmester, H.; Möbius, A.; Wojcieszak, M.: Goals and measures for analyzing power consumption data in manufacturing enterprises. Journal of Data, Information and Management 3/1, S. 65–82, 2021.

[HH20] Henning, S.; Hasselbring, W.: Scalable and Reliable Multi-Dimensional Sensor Data Aggregation in Data-Streaming Architectures. Data-Enabled Discovery and Applications 4/1, S. 1–12, 2020.

[HH21a] Henning, S.; Hasselbring, W.: How to Measure Scalability of Distributed Stream Processing Engines? In: Companion of the ACM/SPEC International Conference on Performance Engineering. ACM, S. 85–88, Apr. 2021.

[HH21b] Henning, S.; Hasselbring, W.: Theodolite: Scalability Benchmarking of Distributed Stream Processing Engines in Microservice Architectures. Big Data Research 25/100209, S. 1–17, Juli 2021.

[HH22] Henning, S.; Hasselbring, W.: A Configurable Method for Benchmarking Scalability of Cloud-Native Applications. Empirical Software Engineering 27/143, S. 1–42, 2022, URL: `https://doi.org/10.1007/s10664-022-10162-1`.

[HHM19] Henning, S.; Hasselbring, W.; Möbius, A.: A Scalable Architecture for Power Consumption Monitoring in Industrial Production Environments. In: 2019 IEEE International Conference on Fog Computing (ICFC). IEEE, Prague, Czech Republic, S. 124–133, Juni 2019.

[HWD21] Hasselbring, W.; Wojcieszak, M.; Dustdar, S.: Control Flow Versus Data Flow in Distributed Systems Integration: Revival of Flow-Based Programming for the Industrial Internet of Things. IEEE Internet Computing 25/4, S. 5–12, 2021.

[KH19] Knoche, H.; Hasselbring, W.: Drivers and Barriers for Microservice Adoption – A Survey among Professionals in Germany. Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modeling 14/1, S. 1–35, 2019.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 61

Smoke testing for machine learning: simple tests to discover severe bugs

Steffen Herbold[1], Tobias Haar[2]

Abstract: We summarize the article *Smoke testing for machine learning: simple tests to discover severe bugs* [HH22], which was published in Empirical Software Engineering in 2022.

Keywords: Machine learning; Classification; Software testing; Smoke testing; Combinatorial testing; Equivalence classes; Boundary-value analysis

1 Overview

The article "Smoke testing for machine learning: simple tests to discover severe bugs" was published in Empirical Software Engineering in 2022. Machine learning is nowadays a standard technique for data analysis within software applications. Software engineers need quality assurance techniques that are suitable for these new kinds of systems. Within this article, we discuss the question whether standard software testing techniques that have been part of textbooks since decades are also useful for the testing of machine learning software. Concretely, we try to determine generic and simple smoke tests that can be used to assert that basic functions can be executed without crashing.

2 Results

We found that we can derive such tests using techniques similar to equivalence classes and boundary value analysis. Moreover, we found that these concepts can also be applied to hyperparameters, to further improve the quality of the smoke tests. Even though our approach is almost trivial, we were able to find bugs in all three machine learning libraries that we tested and severe bugs in two of the three libraries. This demonstrates that common software testing techniques are still valid in the age of machine learning and that considerations how they can be adapted to this new context can help to find and prevent severe bugs, evenin mature machine learning libraries.

[1] Universität Passau, Fakultät für Informatik und Mathematik, Dr.-Hans-Kapfinger-Straße 30, 94032 Passau, Deutschland steffen.herbold@uni-passau.de
[2] Universität Göttingen, Deutschland

3 Data Availability

The data and all analysis scripts are available online [He21].

Literatur

[He21] Herbold, S.: sherbold/replication-kit-2020-smoke-testing: v1.0.0, Version v1.0.0, Dez. 2021, URL: https://doi.org/10.5281/zenodo.5752045.

[HH22] Herbold, S.; Haar, T.: Smoke testing for machine learning: simple tests to discover severe bugs. Empirical Software Engineering 27/2, Jan. 2022, URL: https://doi.org/10.1007/s10664-021-10073-7.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 63

Problems with with SZZ and Features: An empirical assessment of the state of practice of defect prediction data collection

Steffen Herbold[1] Alexander Trautsch[2] Fabian Trautsch[3] Benjamin Ledel[4]

Abstract: We summarize the article *Problems with with SZZ and Features: An empirical assessment of the state of practice of defect prediction data collection* [He22], which was published in Empirical Software Engineering in 2022.

Keywords: SZZ; Bug fix labeling; Bug inducing changes; Defect prediction data; Data set

1 Overview

The article "Problems with with SZZ and Features: An empirical assessment of the state of practice of defect prediction data collection" was published in the Empirical Software Engineering in 2022. The SZZ algorithm is the de facto standard for labeling bug fixing commits and finding inducing changes for defect prediction data. Recent research uncovered potential problems in different parts of the SZZ algorithm. Most defect prediction data sets provide only static code metrics as features, while research indicates that other features are also important. We provide an empirical analysis of the defect labels created with the SZZ algorithm and the impact of commonly used features on results. We used a combination of manual validation and adopted or improved heuristics for the collection of defect data. We conducted an empirical study on 398 releases of 38 Apache projects.

2 Results

We found that only half of the bug fixing commits determined by SZZ are actually bug fixing. If a six-month time frame is used in combination with SZZ to determine which bugs affect a release, one file is incorrectly labeled as defective for every file that is correctly labeled as defective. In addition, two defective files are missed. We also explored the impact

[1] Universität Passau, Fakultät für Informatik und Mathematik, Dr.-Hans-Kapfinger-Straße 30, 94032 Passau, Deutschland steffen.herbold@uni-passau.de
[2] Universität Passau, Fakultät für Informatik und Mathematik, Dr.-Hans-Kapfinger-Straße 30, 94032 Passau, Deutschland alexander.trautsch@uni-passau.de
[3] Universität Göttingen, Deutschland
[4] Universität Göttingen, Deutschland

of the relatively small set of features that are available in most defect prediction data sets, as there are multiple publications that indicate that, e.g., churn related features are important for defect prediction. We found that the difference of using more features is not significant.

3 Data Availability

The data and all analysis scripts are available online [He21].

Literatur

[He21] Herbold, S.; Trautsch, A.; Trautsch, F.; Ledel, B.: Replication kit for: Problems with SZZ and Features: An empirical study of the state of practice of defect prediction data collection, Zenodo, Nov. 2021, URL: https://doi.org/10.5281/zenodo.5675024.

[He22] Herbold, S.; Trautsch, A.; Trautsch, F.; Ledel, B.: Problems with SZZ and features: An empirical study of the state of practice of defect prediction data collection. Empirical Software Engineering 27/2, Jan. 2022, URL: https://doi.org/10.1007/s10664-021-10092-4.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 65

Community Expectations for Research Artifacts and Evaluation Processes

Ben Hermann[1], Stefan Winter[2], Janet Siegmund[3]

Abstract: Artifact evaluation has been introduced into the software engineering and programming languages research community with a pilot at ESEC/FSE 2011 and has since then enjoyed a healthy adoption throughout the conference landscape. We conducted a survey including all members of artifact evaluation committees of major conferences in the software engineering and programming language field from 2011 to 2019 and compared the answers to expectations set by calls for artifacts and reviewing guidelines. While we find that some expectations exceed the ones expressed in calls and reviewing guidelines, there is no consensus on a quality threshold for artifacts in general. We observe very specific quality expectations for specific artifact types for review and later usage, but also a lack of their communication in calls. We also find problematic inconsistencies in the terminology used to express artifact evaluation's most important purpose. We derive several actionable suggestions which can help to mature artifact evaluation in the inspected community and also to aid its introduction into other communities in computer science.

Keywords: Research Artifacts; Artifact Evaluation; Replicability; Reproducibility; Study

1 Summary

In this paper, we present a study on the expectations of the community toward research artifacts and their evaluation processes which was originally presented at ESEC/FSE 2020 and has received the ACM SIGSOFT Distinguished Paper Award [HWS20]. Since the publication of the original study it had impact on numerous artifact evaluation tracks and inspired a quantitative study on artifact quality and visibility published at ESEC/FSE 2022 [Wi22].

In 2016, a replicability crisis became public, when more than 1500 researchers revealed having trouble replicating previous research results. This replicability crisis also reached the software engineering community, as it has embraced the importance of replication for knowledge building. To improve the situation of missing or unusable research artifacts, artifact evaluation has become a regular process for scientific conferences in computer

[1] Technische Universität Dortmund, Fakultät für Informatik, Otto-Hahn-Straße 14, 44227 Dortmund, Deutschland ben.hermann@cs.tu-dortmund.de
[2] Ludwig-Maximilians-Universität München, Fakultät für Informatik, Oettingenstraße 67, 80538 München, Deutschland sw@stefan-winter.net
[3] Technische Universität Chemnitz, Fakultät für Informatik, Straße der Nationen 62, 09111 Chemnitz, Detuschland janet.siegmund@informatik.tu-chemnitz.de

science. Since the first piloting of the process at ESEC/FSE 2011, many other conferences have included artifact evaluations as an additional step.

The overarching goal of our work is to enable an assessment of the efficacy of artifact evaluations and to identify possible improvements for these processes. As a first step towards a systematic assessment of artifact evaluation processes, the objective of our work is to assess their current perception in the AE-pioneering communities. We have conducted a survey among researchers who have served on artifact evaluation committees (AECs). To this end, we have contacted all members of AECs, including the respective chairs, for all artifact evaluations conducted at software engineering and programming language conferences between 2011 and 2019.

We found that the perceived purpose of artifact evaluation is to foster replicability and reusability at the same time. While we could observe several quality criteria to be expected from artifacts, we found no clear consensus on them. Moreover, the expressed expectations were largely not represented in the calls for artifacts. This makes it hard to define a quality standard. The results of our study show that the lack of such quality standards leaves reviewers without guidance how to decide on artifact acceptance or rejection. Moreover, it creates an ambiguity for readers how to interpret the badges awarded to papers after AE.

In summary, we make the following contributions: (1) We provide an overview of the current perception and practice of artifact evaluation and the expectations toward artifacts and the process. (2) Based on community inputs, we present suggestions for future development and improvement of artifact evaluations. (3) We published a research artifact for replication, further analysis, and extension by the community.

2 Data Availability

The original publication is accessible under the DOI `10.1145/3368089.3409767` [HWS20]. Our artifact is available on Github (`https://github.com/bhermann/artifact-survey`) and Zenodo (DOI: `10.5281/zenodo.3951724`).

Bibliography

[HWS20] Hermann, Ben; Winter, Stefan; Siegmund, Janet: Community Expectations for Research Artifacts and Evaluation Processes. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA, p. 469–480, 2020.

[Wi22] Winter, Stefan; Timperley, Christopher S.; Hermann, Ben; Cito, Jürgen; Bell, Jonathan; Hilton, Michael; Beyer, Dirk: A Retrospective Study of One Decade of Artifact Evaluations. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2022, Association for Computing Machinery, New York, NY, USA, p. 145–156, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 67

On the Subjectivity of Emotions in Software Projects: How Reliable are Pre-Labeled Data Sets for Sentiment Analysis? (Summary)

Marc Herrmann[1], Martin Obaidi[2], Larissa Chazette[3], Jil Klünder[4]

Abstract:

Social aspects (e.g., the sentiment of developers) are important for software development. In order to automatically analyze sentiments, sentiment analysis tools use machine learning methods that require data sets labeled according to emotion or polarity. As these labeled data sets strongly influence the tools' accuracy, we investigate whether the labels match developers' perceptions. For this purpose, we conducted an international survey with 94 participants who labeled 100 statements. We compare the median as well as every single participant's perception with the labels. The results show that the median perception of all participants coincides with the predefined labels for 62.5% of the statements, and that the difference between the single participant's ratings and the labels is even worse.

This summary refers to the paper with the title "On the subjectivity of emotions in software projects: How reliable are pre-labeled data sets for sentiment analysis?" [He22b]. It was published in the Journal of Systems and Software (JSS) in 2022 peer-reviewed.

Keywords: Sentiment analysis; software projects; polarity; development team; communication

1 Introduction

Sentiment analysis can be used to investigate the emotions of a team by analyzing whether text messages convey a positive, negative, or neutral feeling. To automize this kind of analysis, an increasing number of sentiment analysis tools has been developed in the last years [Ob22]. These tools analyze text and provide the polarity of the text. Many of these tools are based on machine learning methods that require a training data set. Such data sets were crawled by researchers from platforms such as GitHub or Stack Overflow and manually labeled with an emotion or a polarity. However, considering the diversity of software teams, it seems unlikely that such a gold standard data set reflects all these differences. To analyze

[1] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, Deutschland
marc.herrmann@stud.uni-hannover.de
[2] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, Deutschland
martin.obaidi@inf.uni-hannover.de
[3] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, Deutschland
larissa.chazette@inf.uni-hannover.de
[4] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, Deutschland
jil.kluender@inf.uni-hannover.de

this more specifically, we conducted an international study to investigate the median and individual perceptions of developers on statements of such data sets.

2 Results

We analyzed 94 data points by (1) comparing the median labels of all participants with the predefined labels, and by (2) calculating Cohen's κ between every single participant and the predefined labels as a measure for agreement. These analyses show a remarkable difference between the median and the predefined labels: In 62.5% only, the median label coincide with the predefined labels. Even more, we observe a wide variety in the agreement when comparing the single participant's perception with the predefined labels. A few participants achieve a very high agreement, but others achieve a substantial disagreement.

3 Conclusion

Our results show that both a large number of developers individually and in median often do not match the labels of the data sets. However, these data sets are the training basis for the sentiment analysis tools. This said, it is questionable whether tools trained on such data sets should be used to analyze what is going on in a software team. As long as the predefined labels often do not match the perception of single persons, it is unlikely that the automated analysis reflects the perception of such person. Indeed, tools should be adapted to each individual team, taking into account the different perceptions of the team members.

Data Availability

This paper is based on the data set of the SentiSurvey. The raw data is online available [He22a].

Bibliography

[He22a] Herrmann, Marc; Obaidi, Martin; Chazette, Larissa; Klünder, Jil: Dataset: SentiSurvey for Sentiment Analysis in Software Projects. Zenodo, 2022.

[He22b] Herrmann, Marc; Obaidi, Martin; Chazette, Larissa; Klünder, Jil: On the subjectivity of emotions in software projects: How reliable are pre-labeled data sets for sentiment analysis? Journal of Systems and Software, 193:111448, 2022.

[Ob22] Obaidi, Martin; Nagel, Lukas; Specht, Alexander; Klünder, Jil: Sentiment analysis tools in software engineering: A systematic mapping study. Information and Software Technology, 151:107018, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 69

Early Timing Analysis based on Scenario Requirements and Platform Models (Extended Abstract)

Jörg Holtmann[1], Julien Deantoni[2], Markus Fockel[3]

Abstract: This extended abstract summarizes our article [HDF22], published in the Journal of Software and Systems Modeling and presented as Journal First paper at MODELS'22.

Keywords: scenario-based requirements; platform modeling; real-time systems; timing analysis

Distributed, software-intensive systems must fulfill communication requirements under hard real-time constraints. The requirements have to be documented and validated carefully using a systematic requirements engineering (RE) approach, for example, by applying scenario-based requirements notations. The resources of the execution platforms and their properties induce effects on the timing behavior, which may lead to violations of the real-time requirements.

Such violations of the real-time requirements can occur for various reasons: The ECUs executing the software have restricted resources that increase execution times; the buses and wireless communication media have restricted resources increasing transmission times; the preemption induced by scheduling policies increase response times, et cetera. More generally, the various properties of the particular resources of the execution platform (*resource properties*) impact the timing behavior by inducing *timing effects* (i.e., delays) during the provision of the actual functionality.

Nowadays, the platform resource properties and their induced timing effects are verified against the real-time requirements by means of timing analysis techniques mostly implemented in commercial-off-the-shelf tools. However, such timing analyses are conducted in late development phases since they rely on artifacts produced during these phases.

For enabling timing analyses already in the early RE phase, related work provides means to specify and analyze timed behavioral models (typically relying on scenario- or automata-based notations), thereby abstracting from the final platform-specific artifacts. However, such approaches typically require to reenact and pre-calculate the timing effects induced by the resource properties and to specify them as part of the timed behavioral models.

Thus, in order to enable early timing analyses already during RE and to relieve the timing analysts from the burden to pre-calculate and to specify the timing effects induced by the

[1] Chalmers | University of Gothenburg, Interaction Design & Software Engineering, Sweden jorg.holtmann@gu.se
[2] Universite Cote d'Azur, I3S/INRIA Kairos, France julien.deantoni@univ-cotedazur.fr
[3] Fraunhofer IEM, Safe and Secure IoT Systems, Germany markus.fockel@iem.fraunhofer.de

platform resource properties as part of behavioral models, we extend a scenario-based requirements notation with allocation means to platform models and define operational semantics for the purpose of simulation-based, platform-aware timing analyses. Figure 1 sketches an overview of the approach and of the contributions.

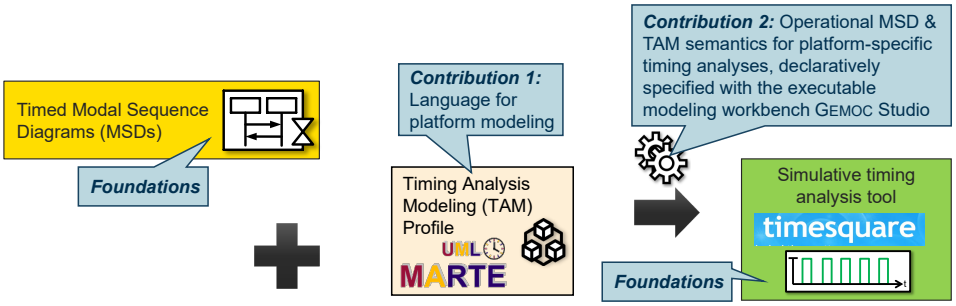


Fig. 1: Overview of the Approach and Contributions

Since the targeted real-time software-intensive systems strongly rely on message-based communication, we base the real-time requirements on our timed and component-based dialect of the scenario-based notation of Modal Sequence Diagrams (MSDs) (cf. left-hand side of Figure 1). Like the related work, the modeling and analysis means provided by our dialect enable specifying and validating real-time requirements but incorporate platform-specific aspects only insufficiently. To provide both an abstract specification of the execution platform with its particular resource properties and the allocation of MSD specifications to the execution platform, we furthermore extend platform modeling concepts of the real-time modeling UML profile MARTE (cf. middle of Figure 1). Based on the modeling languages mentioned above, we introduce as main contribution a new operational semantics for platform-aware MSDs dedicated to timing analyses (cf. arrow in the middle of Figure 1). This semantics encompasses an extended MSD message event handling semantics and particularly encapsulates the computation of the resource properties into platform-induced timing effects. To operationalize the semantics, we apply our workbench GEMOC Studio for the declarative specification of executable modeling languages. The overall approach then enables verifying the timing effects w.r.t. the real-time requirements specified by timed MSDs in platform-aware timing analyses through applying simulation and model checking in our tool suite TIMESQUARE (cf. right-hand side of Figure 1).

Data Availability The supplementary material encompasses the tooling for the actual approach, the tooling for parts of the evaluation, and evaluation data that is more detailed than in the published article. We published it at Zenodo (cf. reference in the article).

Bibliography

[HDF22] Holtmann, Jörg; Deantoni, Julien; Fockel, Markus: Early Timing Analysis based on Scenario Requirements and Platform Models. Software and Systems Modeling (Theme Section on Model-Driven Requirements Engineering), 21(6):2171–2211, 2022. J1st @ MODELS'22.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 71

A systematic literature review on counterexample explanation – Summary

Arut Prakash Kaleeswaran[1], Arne Nordmann[1], Thomas Vogel[2], Lars Grunske[2]

Abstract: In this extended abstract, we summarize our systematic literature review on counterexample explanation published in the journal Information and Software Technology (IST) in 2022 [Ka22].

Keywords: Model checking; Counterexample explanation; Safety

Summary. In order to assure the safety of the systems, automotive systems are often created in accordance with standards like ISO 26262. These standards use time-consuming and error-prone manual safety analysis techniques, such as Failure Mode and Effect Analysis, Fault Tree Analysis, and Hazard and Operability, to undertake safety analysis. To overcome these challenges, formal methods is a potential option, e.g., model checking, an automated verification method. Model checkers take the system model and specifications as input and verify whether the specifications are satisfied by the system model. If not, then a counterexample is provided by the model checker that illustrates the violation with the execution path and system states. Nevertheless, comprehending the counterexample is challenging because it is cryptic, relevant erroneous states and variables are not highlighted, and debugging is performed manually. These challenges call for a method *counterexample explanation* to ease the error comprehension. Thus, the systematic literature review (SLR) is performed on counterexample explanation to provide an overview on the state of the art, mainly focusing on the types of counterexample representation, the methods to transform or optimize a counterexample, to provide an explanation, influence of the input system and requirement on counterexample explanation, and the different domains and applications to evaluate approaches to counterexample explanation. The SLR has been conducted as part of a research project between BOSCH and HU Berlin on making model-checking results in contract-based design of safety-critical systems easier to understand [Ka20].

The survey is performed by collecting answers for the eight specific research questions from 116 primary studies. By surveying 116 primary studies, four counterexample representations types are identified: graphical, textual, tabular, and trace representations. Among these, graphical and trace formats are predominantly used (89 of 116 primary studies, 77%) to represent a counterexample. For instance, the counterexample is presented in a SysML or UML diagram. A trace representation is a modified form of a counterexample with the addition or removal of (sub)-traces. Trace representation is further categorized into three types based

[1] Bosch Corporate Sector Research, Renningen, Germany. ArutPrakash.Kaleeswaran@bosch.com
[2] Humboldt-Universität zu Berlin, Software Enginering Group, Unter den Linden 6, 10099 Berlin, Germany. {thomas.vogel, grunske}@informatik.hu-berlin.de

on processing a counterexample: minimized counterexample, multiple counterexamples, and witness traces (traces that satisfy the failed specification, which is contrary to the counterexample). Among 54 primary studies that process the counterexample, majority (38, 70%) use minimized counterexample approach. Providing additional information along with the counterexample explanation improves error comprehension, for example, highlighting erroneous state transitions in the state-machine. However, only 8 of 116 primary studies are found to provide additional information along with the counterexample explanation.

In this study, we also investigated whether counterexample explanations are represented and explained with the user given input domain or those different from the input domain. Among 81 primary studies (excluding 31 studies that use trace representation), 60 studies represent the counterexample different from the given input domain, and only 21 studies represent the counterexample in the given input domain. Furthermore, we have also collected verification tools and frameworks used for verification and explaining counterexamples, as well as temporal logic and specification properties used to express system specifications. Mostly used verification tools are NuSMV/nuXmv, PRISM, and SPIN, frameworks are ASSERT, DiPro, and MODCHK, temporal logics are LTL and CTL, and specification properties are safety properties. Finally, focusing on the evaluation methods, evaluation aspects, and applications used for the evaluation, the majority employs use-case based evaluation method focusing on effectiveness by using industrial applications.

This survey reveals several open points and future directions to enhance the counterexample explanation approach. One of the main findings is still the counterexample explanation needs improvement for non-experts in formal methods. We found many frameworks for counterexample explanation and a considerable number of these are open-source. Therefore, it seems advisable to build upon existing frameworks and improve them. To improve the effectiveness and usability of counterexample explanations, user studies need to be performed where one could gain insights on the degree of error comprehension. Such user studies will provide evidence for the effectiveness and usability of such approaches from a user's point of view and hence can guide future research on counterexample explanation.

Data Availability. We published an online appendix [Ka21] on Zenodo (`https://doi.org/10.5281/zenodo.5679227`) that lists the data items to be extracted and a bibliography of the primary studies, and tabulates primary studies and extracted data for the quantitative items.

Bibliography

[Ka20] Kaleeswaran, Arut Prakash; Nordmann, Arne; Vogel, Thomas; Grunske, Lars: Counterexample Interpretation for Contract-Based Design. In: 7th Intl. Symposium on Model-Based Safety and Assessment, IMBSA. volume 12297 of LNCS. Springer, pp. 99–114, 2020.

[Ka21] Kaleeswaran, Arut Prakash; Nordmann, Arne; Vogel, Thomas; Grunske, Lars: Appendix of the paper: A Systematic Literature Review on Counterexample Explanation. 2021.

[Ka22] Kaleeswaran, Arut Prakash; Nordmann, Arne; Vogel, Thomas; Grunske, Lars: A systematic literature review on counterexample explanation. Information and Software Technology, 145:106800, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 73

Meetings and Mood – Related or Not? Insights from Student Software Projects (Summary)

Jil Klünder[1], Oliver Karras[2]

Abstract: Meetings are part of most software projects which is why they have been frequently analyzed by researchers. Often, this research focuses on the interactions. We analyze meetings from a more abstract view by applying sentiment analysis to the statements made during the meeting. That is, we analyze whether the statements are positive, negative, or neutral, and how the statements made are related to the mood of a team before and after the meeting. Our results are based on insights from 21 student software projects and show some interesting findings, including that the amount of positive and negative statements during the meeting has no measurable influence on the mood afterwards.

This summary refers to the paper "Meetings and Mood – Related or Not? Insights from Student Software Projects" [KK22]. This paper was published in the proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2022.

Keywords: Software development teams, sentiment analysis, meeting, mood

1 Introduction

As most software projects require team work, adequate collaboration and interactions between the project team members are essential. This also includes the information exchange in meetings. So far, research has focused on interactions in meetings, that is what kind of statements they made (e.g., whether they blame or support others, or express interest). In our research [KK22], we investigated meetings from a more abstract view by analyzing the polarity of the statements made. We then compared the amount of positive and negative statements with respect to their relation to the mood of a team and conflicts.

2 Methodology

Overall, we wanted to analyze relations between (1) the mood before a meeting, (2) the polarity of statements in a meeting, and (3) the mood and other social aspects after a meeting. For this purpose, we analyzed the transcripts of 21 student software project meetings. In addition, in these projects, we collected data on social aspects such as the affective

[1] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, Deutschland, jil.kluender@inf.uni-hannover.de
[2] TIB - Leibniz-Informationszentrum für Technik und Naturwissenschaften, Welfengarten 1B, 30167 Hannover, Deutschland, oliver.karras@tib.eu

state and the perceived likelihood for social or task-related conflicts. The data analysis followed a two-step approach: (1) We applied sentiment analysis to the meeting transcripts using the SEnti-Analyzer [HOK20] and retrieved the amount of positive, negative, and neutral statements for each meeting. (2) Then, we applied hypothesis testing to analyze the data with respect to the relationships mentioned above. In total, we tested 8 main and 20 sub-hypotheses [KK22].

3 Results

The data analysis provides some interesting insights, including that (1) the amount of positive statements depends on the positive mood before the meeting, that (2) the perceived likelihood for social conflicts depends on the amount of negative statements during the meeting, and that (3) the perceived likelihood for task-related conflicts depends on the amount of positive and negative statements during the meeting. Even more interesting, there are two cases for which we did not find a measurable influence: (1) We neither find an evidence for an influence of the overall mood before the meeting and the polarity of statements during the meeting, nor (2) an evidence for an influence of the polarity of statements during the meeting on the mood of a team after the meeting.

4 Conclusion

We can conclude that *starting a meeting with a high positive mood can smooth both the meeting start as well as the meeting as a whole* and that *the polarity of statements made during the meeting has no measurable influence on the mood afterwards*. Nevertheless, one should not assume that the behavior during the meeting does not matter. Indeed, we argue that it is more likely that such behavior influences the short-term emotional state after the meeting (which needs to be proven by future research) rather then the long-term affective state which we considered in this research.

Data Availability

Due to ethical concerns, we are not allowed to share the raw data publicly.

Bibliography

[HOK20] Herrmann, Marc; Obaidi, Martin; Klünder, Jil: SEnti-Analyzer: Joint Sentiment Analysis For Text-Based and Verbal Communication in Software Projects. Technical Report 1.0, Software Engineering Group, Leibniz Universität Hannover, 2020. arXiv.

[KK22] Klünder, Jil; Karras, Oliver: Meetings and Mood–Related or Not? Insights from Student Software Projects. In: Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 75

Evaluation Methods and Replicability of Software Architecture Research Objects

Marco Konersmann[1], Angelika Kaplan[2], Thomas Kühn[2], Robert Heinrich[2], Anne Koziolek[2], Ralf Reussner[2], Jan Jürjens[3,4], Mahmood al-Doori[3], Nicolas Boltz[2], Marco Ehl[3], Dominik Fuchß[2], Katharina Großer[3], Sebastian Hahner[2], Jan Keim[2], Matthias Lohr[3], Timur Sağlam[2], Sophie Schulz[2], Jan-Philipp Töberg[2, §]

Abstract: Our paper at the 19th IEEE International Conference on Software Architecture (ICSA 2022) started by noticing that Software Architecture (SA) as research area experienced an increase in empirical research. Empirical research builds a sound foundation for validity and comparability. A current overview of the evaluation and replicability of SA research objects could help to discuss our empirical standards as a community. However, no current overview existed. We assessed the current state of practice of evaluating SA research objects and replication artifact provision in full technical conference papers from ICSA and the European Conference on Software Architecture (ECSA) 2017–2021. We first developed a categorization schema for SA research object evaluation and artifact provisioning. In a systematic literature review with 153 papers, we then classified the papers according to that schema. From our findings we derive and describe four proposals for improving the state of practice in evaluating SA research objects.

Keywords: software architecture research, meta-research, systematic literature review, evaluation

Motivation: Software Architecture (SA) as research area experienced an increase in empirical research [GW16], which can be considered important for the validity and comparability. Our paper [Ko22] creates an overview of the evaluation and replicability of SA research objects to help discussing our empirical standards as a community.

Research Method: We categorized SA research w.r.t. their evaluation and replicability and created an overview of the current state of practice in evaluating SA research. Therefore, we created a classification schema for the validation of SA research evaluations and conducted a systematic literature review (SLR) of 153 full technical papers published at ECSA and ICSA from 2017 to 2021. We discussed our findings and presented proposals for improvement.

Findings: Although there are valid reasons for not publishing replication packages, our results indicate that improvements of generalizability and repeatability of evaluations could enhance the field's maturity. We summarize the answers to our research questions as follows:

RQ 1: What is the distribution of research objects and their evaluation and how did their proportions change over time? SA research at the ECSA and ICSA is quite diverse w.r.t. research objects with a focus on analysis and design methods (33% of research objects).

[1] Software Engineering, RWTH Aachen University, Germany, konersmann@se-rwth.de
[2] Karlsruhe Institute of Technology, Germany, {firstname.lastname}@kit.edu, §uexdy@student.kit.edu
[3] University of Koblenz-Landau, Germany, {lastname,mahmoodaldoori,mehl,matthiaslohr}@uni-koblenz.de
[4] Fraunhofer Institute for Software and Systems Engineering, ISST, Germany

Case studies and technical experiments are the dominating evaluation methods. Most (58%) evaluation methods are used to measure exactly one quality. We see that neither research objects nor evaluation methods heavily changed in the past five years with a trend to more artifact provisioning since 2019.

RQ 2: How are specific research objects evaluated and how accessible are their evaluation artifacts? The most prominent way of evaluation in the investigate papers is to measure the functional suitability and performance using technical experiments and case studies. The human-centered practice architecture decision making is mostly evaluated with human-centered evaluation methods: interviews and focus groups. Few comparative methods, like benchmarks, are used. Overall, we see no clear agreement on which properties should be evaluated for specific research objects or which methods to use for specific properties.

RQ 3: Which guidelines are used for evaluation? 17% of the papers reference evaluation guidelines. 14% reference guidelines for threats to validity. The two most-referenced guidelines in both categories describe how to conduct and report case studies and how to describe their threats to validity. Overall, we can observe that guidelines are not systematically referenced in the investigated papers.

Conclusion: We derive and describe four proposals for improving the state of practice in evaluating SA research objects: (P1) to foster the generalizability of evaluation results, (P2) to develop more benchmarks to compare approaches, (P3) to foster the provision of replication packages, and (P4) to build guidelines for what and how to evaluate, and which threats to validity should be discussed for these methods. Researchers can use our results to identify recommendations on relevant properties and methods for evaluation and to find reusable artifacts to compare their approaches with existing research. Reviewers can use our results to compare the evaluation and replicability of submissions with the state of the practice.

Data Availability: We provide a replication package[5] with the tabulated and visualized review data, a BibTeX file with all papers considered, scripts for summarizing and visualizing, and a copy of a wiki that was used for collaboration of the reviewers in our SLR. All investigated papers are listed online[6].

References

[GW16] Galster, Matthias; Weyns, Danny: Empirical Research in Software Architecture: How Far have We Come? In: 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016, Venice, Italy, April 5-8, 2016. IEEE Computer Society, pp. 11–20, 2016.

[Ko22] Konersmann, Marco; Kaplan, Angelika; Kühn, Thomas; Heinrich, Robert; Koziolek, Anne; Reussner, Ralf; Jürjens, Jan; Al-Doori, Mahmood; Boltz, Nicolas; Ehl, Marco; Fuch, Dominik; Großer, Katharina; Hahner, Sebastian; Keim, Jan; Lohr, Matthias; Sağlam, Timur; Schulz, Sophie; Töberg, Jan-Philipp: Evaluation Methods and Replicability of Software Architecture Research Objects. In: 2022 IEEE 19th International Conference on Software Architecture (ICSA). IEEE, Los Alamitos, CA, USA, pp. 157–168, March 2022.

[5] Replication Package: https://doi.org/10.5281/zenodo.6044059

[6] Wiki: https://gitlab.com/SoftwareArchitectureResearch/StateOfPractice/-/wikis/Results

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 77

Collaborative Program Comprehension in Extended Reality

Alexander Krause-Glau[1] Malte Hansen[2] Wilhelm Hasselbring[3]

Abstract: This contribution has been published in the journal *Information and Software Technology* (Elsevier) in 2022 [KHH22], https://doi.org/10.1016/j.infsof.2022.107007.

Keywords: Program Comprehension; Software Visualization; City Metaphor; Extended Reality; Virtual Reality; Augmented Reality

1 Introduction

In software visualization research, various approaches strive to create immersive environments by employing extended reality devices. In that context, only few research has been conducted on the effect of collaborative, i.e., multi-user, extended reality environments. ExplorViz's multi-user modes are our approach to enable heterogeneous collaborative software visualizations. Unlike related work, we approach the latter by introducing a multi-user augmented reality environment for software visualizations based on off-the-shelf mobile devices.

In this paper [KHH22], we present our journey toward a web-based approach to enable (location-independent) collaborative program comprehension using desktop [Fi13], virtual reality [FKH15b], physical 3D Models [FKH15a], and mobile augmented reality devices [KHH21]. We designed and implemented a device-heterogenous multi-user mode in our web-based live trace visualization tool ExplorViz [FKH17; HKZ20]. Users can employ desktop, mobile, and virtual reality devices to collaboratively explore software visualizations. We conducted user studies for common program comprehension tasks [Fi15; FKH15c; KBH22; KHH22]. In this context, we also investigate the scalable implementation and deployment of ExplorViz in the cloud [KH22].

2 Data Availability

A supplementary data package is available at Zenodo (https://doi.org/10.5281/zenodo.5790175). The source code is available at GitHub (https://github.com/ExplorViz) and the software documentation at https://explorviz.dev/.

[1] Kiel University, Software Engineering Group, D-24098 Kiel, Germany akr@informatik.uni-kiel.de
[2] Kiel University, Software Engineering Group, D-24098 Kiel, Germany mha@informatik.uni-kiel.de
[3] Kiel University, Software Engineering Group, D-24098 Kiel, Germany hasselbring@email.uni-kiel.de

Literatur

[Fi13] Fittkau, F.; Waller, J.; Wulf, C.; Hasselbring, W.: Live Trace Visualization for Comprehending Large Software Landscapes: The ExplorViz Approach. In: 1st IEEE International Working Conference on Software Visualization (VISSOFT 2013). S. 1–4, Sep. 2013.

[Fi15] Fittkau, F.; Finke, S.; Hasselbring, W.; Waller, J.: Comparing Trace Visualizations for Program Comprehension through Controlled Experiments. In: Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC 2015). IEEE, S. 266–276, Mai 2015.

[FKH15a] Fittkau, F.; Koppenhagen, E.; Hasselbring, W.: Research Perspective on Supporting Software Engineering via Physical 3D Models. In: Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015). IEEE, S. 125–129, Sep. 2015.

[FKH15b] Fittkau, F.; Krause, A.; Hasselbring, W.: Exploring Software Cities in Virtual Reality. In: Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015). IEEE, S. 130–134, Sep. 2015.

[FKH15c] Fittkau, F.; Krause, A.; Hasselbring, W.: Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment. In: Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015). IEEE, S. 36–45, Sep. 2015.

[FKH17] Fittkau, F.; Krause, A.; Hasselbring, W.: Software Landscape and Application Visualization for System Comprehension with ExplorViz. Information and Software Technology 87/, S. 259–277, Juli 2017.

[HKZ20] Hasselbring, W.; Krause, A.; Zirkelbach, C.: ExplorViz: Research on software visualization, comprehension and collaboration. Software Impacts 6/, S. 100034, Nov. 2020, URL: https://doi.org/10.1016/j.simpa.2020.100034.

[KBH22] Krause-Glau, A.; Bader, M.; Hasselbring, W.: Collaborative Software Visualization For Program Comprehension. In: 10th IEEE Working Conference on Software Visualization (VISSOFT 2022). Limassol, Cyprus, Okt. 2022.

[KH22] Krause-Glau, A.; Hasselbring, W.: Scalable Collaborative Software Visualization as a Service: Short Industry and Experience Paper. In: 10th IEEE International Conference on Cloud Engineering (IC2E 2022). Sep. 2022.

[KHH21] Krause, A.; Hansen, M.; Hasselbring, W.: Live Visualization of Dynamic Software Cities with Heat Map Overlays. In: 2021 Working Conference on Software Visualization (VISSOFT). IEEE, S. 125–129, Sep. 2021.

[KHH22] Krause-Glau, A.; Hansen, M.; Hasselbring, W.: Collaborative program comprehension via software visualization in extended reality. Information and Software Technology 151/, Nov. 2022, URL: https://doi.org/10.1016/j.infsof.2022.107007.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 79

Incremental Software Product Line Verification — A Performance Analysis with Dead Variable Code

Christian Kröher[1] Moritz Flöter[2] Lea Gerling[3] Klaus Schmid[4]

Abstract: In this work, we summarize our journal paper published in Empirical Software Engineering (EMSE) in 2022 [Kr22]. Verification approaches for Software Product Lines (SPL) aim at detecting variability-related defects and inconsistencies. In general, these analyses take a significant amount of time to provide complete results for an entire, complex SPL. If the SPL evolves, these results potentially become invalid, which requires a time-consuming re-verification of the entire SPL for each increment.

However, in previous work we showed that variability-related changes occur rather infrequently and typically only affect small parts of a SPL. In this paper, we utilize this observation and present an *incremental dead variable code analysis* as an example for *incremental SPL verification*, which achieves significant performance improvements. It explicitly considers changes and partially updates its previous results by re-verifying changed artifacts only. We apply this approach to the Linux kernel demonstrating that our fastest incremental strategy takes only 3.20 seconds or less for most of the changes, while the non-incremental approach takes 1,020 seconds in median. We also discuss the impact of different variants of our strategy on the overall performance, providing insights into optimizations that are worthwhile.

Keywords: Software product line analysis; Evolution; Incremental verification; Dead variable code analysis

1 Summary

Software Product Line (SPL) engineering aims at developing software as a set of related products. They share a common infrastructure, but vary in their individual capabilities. This *variability* enables the configuration and combination of generic artifacts to create a wide range of specific product variants.

A significant challenge in SPL engineering is the correct and consistent evolution of variability information. One of the reasons for this is the scattering of variability information

[1] University of Hildesheim, Institute of Computer Science, Software Systems Engineering, Universitätsplatz 1, 31141 Hildesheim, Germany kroeher@sse.uni-hildesheim.de
[2] University of Hildesheim, Institute of Computer Science, Software Systems Engineering, Universitätsplatz 1, 31141 Hildesheim, Germany moritzf@gmail.com
[3] University of Hildesheim, Institute of Computer Science, Software Systems Engineering, Universitätsplatz 1, 31141 Hildesheim, Germany gerling@sse.uni-hildesheim.de
[4] University of Hildesheim, Institute of Computer Science, Software Systems Engineering, Universitätsplatz 1, 31141 Hildesheim, Germany schmid@sse.uni-hildesheim.de

across different artifacts, like code and build files. In large SPLs, this includes thousands of files, which are connected by a vast number of variability-related configuration options, conditions and references. This complexity contains the risk of unintentionally introducing variability-related defects in or inconsistencies among those files as part of an evolutionary change. SPL verification approaches exist to detect such problems requiring multiple seconds up to entire days to provide their results. Upon evolution of a SPL, these delays will occur for every update (even minor changes), if verification is used to ensure correctness and consistency continuously.

In this paper, we present an approach based on a regression concept over all relevant types of artifacts and the results of analyzing its performance. The approach relies on a fine-grained commit analysis as well as the consideration of the relations between input artifacts, extracted variability information, and the core algorithm of an analysis. We explicitly omit introducing additional models for change impact analysis, but utilize already available information only, like the changes documented by individual commits of a repository. The core analysis algorithm remains unmodified as we exemplify by the *dead variable code analysis*, which identifies never used variable code blocks.

In order to analyze potential strategies for our approach and to compare their performance, we conceptually introduce different levels of change granularity and technically realize adaptation options to switch among them. The result is a set of three incremental variants of the dead variable code analysis, which consider changes on the level of files, file content, and variability information as part of file content changes. Our analysis compares these three incremental variants and the original non-incremental one with respect to their performances by applying them to a subset of the Linux kernel evolution history.

The results of our performance analysis show that our incremental approach significantly accelerates a dead variable code analysis. Our fastest incremental variant takes only 3.20 seconds or less for most of the changes, while the non-incremental variant takes 1,020 seconds in median. At the same time, the analysis results are 100% accurate and the introduced overhead is neglectable.

2 Data Availability

The original journal paper is publicly available via the DOI 10.1007/s10664-021-10090-6. All data related to this paper is publicly available on GitHub (`https://github.com/SSE-LinuxAnalysis/IncrementalAnalysesEvaluation/`).

Bibliography

[Kr22] Kröher, Christian; Flöter, Moritz; Gerling, Lea; Schmid, Klaus: Incremental Software Product Line Verification - A Performance Analysis with Dead Variable Code. Empirical Software Engineering, 27(68):1–41, March 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 81

Hacking or Engineering? Towards an Extended Entrepreneurial Software Engineering Model

Marco Kuhrmann[1], Jürgen Münch[2], Jil Klünder[3]

Abstract: Entrepreneurial software engineering gains increasing interest in research and practice, as it is not necessary to be an experienced software engineer to start a software startup. Nevertheless, there are (e.g., quality and security) requirements that need to be fulfilled. Based on a proposed solution fo a systematic software development for early-stage startups, we identify the methodological and technical priorities of software startups.

This summary refers to the paper "Hacking or Engineering? Towards an Extended Entrepreneurial Software Engineering Model" [KMK22]. This paper was published in the proceedings of the International Conference on Software and Systems Processes 2022.

Keywords: Software development in startups; software process; hybrid development method

1 Introduction

Startups play a key role in software-based innovation. They make an important contribution to an economy's ability to compete and innovate, and their importance will continue to grow due to increasing digitalization. However, the success of a startup depends primarily on market needs and the ability to develop a solution that is attractive enough for customers to choose. A sophisticated technical solution is usually not critical, especially in the early stages of a startup. It is not necessary to be an experienced software engineer to start a software startup. However, this can become problematic as the solution matures and software complexity increases. Based on a proposed solution for systematic software development for early-stage startups, in this paper, we present the key findings of a survey study to identify the methodological and technical priorities of software startups, and we update the initial "Entrepreneurial Software Engineering Model", respectively.

2 Methodology

We conducted a survey study to analyze *how software startups develop software and how research can provide them with proper methodological support*. In particular, we are

[1] Reutlingen University, Alteburgstraße 150, 72762 Reutlingen, Germany kuhrmann@acm.org
[2] Reutlingen University, Alteburgstraße 150, 72762 Reutlingen, Germany j.muench@computer.org
[3] Leibniz University Hannover, Welfengarten 1, 30167 Hannover, Germany jil.kluender@inf.uni-hannover.de

interested in the requirements engineering process, the architecture and design approaches, the product development strategies, and in how far these processes change over time. The research is grounded in an initial "Entrepreneurial Software Engineering" model [BMK21], which provides the basic structure for the questionnaire. In total, we received 70 responses of which 40 responses are complete. These responses are described and analyzed using descriptive and analytical statistics.

3 Results

The results of the survey draw a mixed picture. On the one hand, all software engineering disciplines are considered important. On the other hand, we observe inconclusive requirements engineering approaches and absent or unknown architecture and design approaches.

In particular, from a software engineering perspective, several methods and practices reported to be used do not solve the problem they are meant to solve (according to the entrepreneurs). A structured and systematic method is often absent. In addition, the pragmatic approach used in the very early stages often does not change over time: If a startup decides in the very beginning to start without a profound architectural design, it is unlikely that this will change after a couple of months.

4 Discussion and Conclusion

Among other things, we found that requirements engineering and architecture pose challenges for startups. In addition, we found evidence that startups' software development approaches do not tend to change over time. An early investment in a more scalable development approach could help avoid long-term software problems. To support such an investment, we propose an *extended* model for "Entrepreneurial Software Engineering" that, notably, extends the requirements engineering parts to provide a better guideline in the early stages of developing a software product. This extended model also provides a foundation for future research.

Literaturverzeichnis

[BMK21] Brunner, Daniel; Münch, Jürgen; Kuhrmann, Marco: Entrepreneurial Software Engineering: Towards a Hybrid Development Method for Early-Stage Startups. 45. WI-MAW-Rundbrief, 27(1):5–15, March 2021.

[KMK22] Kuhrmann, Marco; Münch, Jürgen; Klünder, Jil: Hacking or Engineering? Towards an Extended Entrepreneurial Software Engineering Model. In: Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering. ICSSP. ACM, S. 66–76, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 83

Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses

Elias Kuiter[1] Sebastian Krieter[2] Chico Sundermann[3] Thomas Thüm[4] Gunter Saake[5]

Abstract: This work was published at the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE) 2022 [Ku22].

Feature modeling is widely used to systematically model features of variant-rich software systems and their dependencies. By translating feature models into propositional formulas and analyzing them with solvers, a wide range of automated analyses across all phases of the software development process become possible. Most solvers only accept formulas in conjunctive normal form (CNF), so an additional transformation of feature models is often necessary. However, it is unclear whether this transformation has a noticeable impact on analyses. We compare three transformations for bringing feature-model formulas into CNF. We analyze which transformation can be used to correctly perform feature-model analyses and evaluate three CNF transformation tools on a corpus of 22 real-world feature models. Our empirical evaluation illustrates that some CNF transformations do not scale to complex feature models or even lead to wrong results for model-counting analyses. Further, the choice of the CNF transformation can substantially influence the performance of subsequent analyses.

Keywords: Feature Modeling; Automated Reasoning; Conjunctive Normal Form

Many software systems in today's industry can be diversely configured to serve specific customer needs, making it necessary to systematically model their features and dependencies. To this end, *feature models* are widely used. *Satisfiability (SAT) solvers* search for satisfying assignments of propositional formulas and are routinely used for the automated analysis of feature models. Similarly, *model-counting (#SAT) solvers* count satisfying assignments of propositional formulas and also empower numerous feature-model analyses.

To analyze feature models using solvers, they must be translated into propositional formulas. For automated analysis using solvers, these formulas typically must be supplied in *conjunctive normal form (CNF)*, which necessitates a transformation into CNF. However, in many papers on feature-model and variability analysis, this step (although necessary) is not mentioned or discussed only superficially; and evaluations using tools for feature-model extraction or analysis typically do not document the chosen CNF transformation. Moreover, we repeatedly observed in industry collaborations that using different CNF transformations may affect the efficiency and results of analyses.

[1] Otto-von-Guericke University Magdeburg kuiter@ovgu.de
[2] University of Ulm sebastian.krieter@uni-ulm.de
[3] University of Ulm chico.sundermann@uni-ulm.de
[4] University of Ulm thomas.thuem@uni-ulm.de
[5] Otto-von-Guericke University Magdeburg saake@ovgu.de

To assess the impact of CNF transformations on SAT- and #SAT-based feature-model analyses, we describe and compare three state-of-the-art techniques for transforming feature-model formulas into CNF: the distributive [BL99], Tseitin [Ts83], and Plaisted-Greenbaum [PG86] transformation. As a tool for comparison, we propose a taxonomy of five properties (two of which have not been considered in the literature before) by which we classify these three transformations. We characterize how our taxonomy relates to selected feature-model analyses, finding that the distributive and Tseitin transformations are suitable for for model-counting analyses, while the Plaisted-Greenbaum transformation is not.

In addition, we empirically evaluate the efficiency of three CNF transformation tools commonly used for feature-model analyses on a corpus of 22 real-world feature models. We find that the selection of a CNF transformation has a substantial impact not only on the performance of the transformation itself, but also on the efficiency and even the correctness of subsequent analyses.

In summary, both our theoretical analysis and empirical evaluation show that the selection of CNF transformations is highly relevant for practitioners and researchers, especially when using performance-critical analyses, and has to be considered carefully.

Data Availability To ensure reproducibility, we disclose our fully automated evaluation pipeline[6] and all feature models, solvers, and results as a replication package.[7]

Bibliography

[BL99] Büning, Hans Kleine; Lettmann, Theodor: Propositional logic: deduction and algorithms, volume 48. Cambridge University Press, 1999.

[Ku22] Kuiter, Elias; Krieter, Sebastian; Sundermann, Chico; Thüm, Thomas; Saake, Gunter: Tseitin or not Tseitin? The Impact of CNF Transformations on Feature-Model Analyses. In: Proc. Int'l Conf. on Automated Software Engineering (ASE). ACM, October 2022.

[PG86] Plaisted, David A; Greenbaum, Steven: A Structure-Preserving Clause Form Translation. J. Symbolic Computation, 2(3):293–304, 1986.

[Ts83] Tseitin, Grigori S.: On the Complexity of Derivation in Propositional Calculus. In: Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970. Springer, pp. 466–483, 1983.

[6] Automation scripts available at: `https://doi.org/10.5281/zenodo.6922807`
[7] Replication package available at: `https://doi.org/10.5281/zenodo.6525375`

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 85

variED: An Editor for Collaborative, Real-Time Feature Modeling

Elias Kuiter[1] Sebastian Krieter[2] Jacob Krüger[3] Gunter Saake[4] Thomas Leich[5]

Abstract: This work was published in *Empirical Software Engineering* (EMSE) 26, 2 (2021) [Ku21].

Feature models are a helpful means to document, manage, maintain, and configure the variability of a software system. Various stakeholders in an organization may get involved in modeling the features in such a software system. Currently, collaboration in such a scenario can only be done with face-to-face meetings or by combining single-user feature-model editors with additional communication and version-control systems. While face-to-face meetings are often costly and impractical, using version-control systems can cause merge conflicts and inconsistency within a model. Advanced tools that solve these problems by enabling collaborative, real-time feature modeling, analogous to Google Docs or Overleaf for text editing, are missing. We describe the formal foundations of collaborative, real-time feature modeling; a conflict resolution algorithm; proofs that our formalization converges and preserves causality as well as user intentions; a prototype; and the results of an empirical evaluation to assess the prototype's usability. Our contributions provide the basis for advancing existing feature-modeling practices to support collaborative feature modeling. Our prototype is considered helpful and valuable by 17 users, also indicating opportunities for new research directions.

Keywords: Feature Modeling; Collaboration; Consistency Maintenance

Modeling the variability of a software product line in a feature model is essential for an organization to document and manage all implemented features, and also to derive valid configurations that are tailored to different customer requirements. To create a meaningful feature model, all relevant stakeholders must work collaboratively—however, there is neither a tool nor a technique that supports *collaborative, real-time editing* of the same feature model, similar to the text editors *Google Docs* and *Overleaf*. Nonetheless, such a tool promises advantages in several use cases, for instance when (a) working simultaneously on different or coordinated tasks, (b) sharing the model with domain experts for real-time feedback and evolution, or (c) teaching feature-modeling concepts and performing hands-on exercises. In particular, the COVID-19 pandemic highlighted the value of remote collaboration.

To support these use cases, we describe the conceptual foundations of collaborative, real-time feature modeling [Ku19, Ku21]. We define requirements that our technique aims to fulfill, derive formal specifications for the operations that we need to develop, extend

[1] Otto-von-Guericke University Magdeburg, Germany kuiter@ovgu.de
[2] University of Ulm, Germany sebastian.krieter@uni-ulm.de
[3] Eindhoven University of Technology, The Netherlands j.kruger@tue.nl
[4] Otto-von-Guericke University Magdeburg, Germany saake@ovgu.de
[5] Harz University of Applied Sciences, Wernigerode, Germany tleich@hs-harz.de

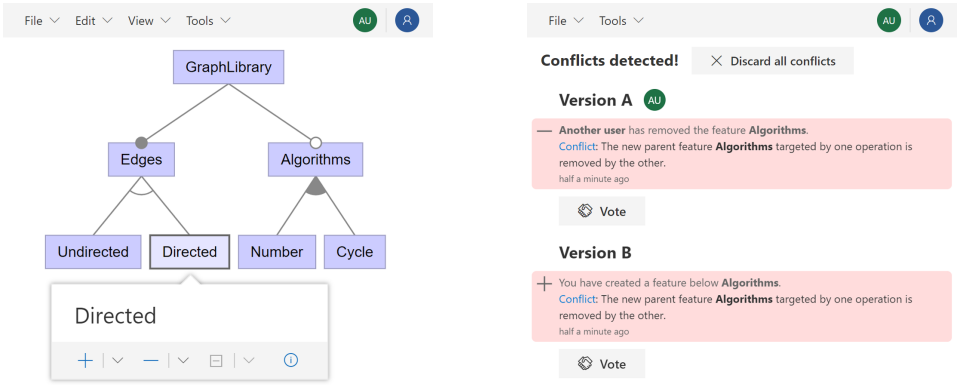


Fig. 1: Editing a feature model (left) and resolving a conflict (right) in variED.

the concurrency-control technique MVMD [SC02] to detect and resolve conflicts, and implement variED, an editor for collaborative, real-time feature modeling (cf. Figure 1).

We further prove that our technique is correct and evaluate it empirically in a user study with 17 participants. The results of our empirical user study show that our tool supports the defined use cases well and is a helpful means to extend current collaboration strategies (e.g., versioning via Git). More precisely, the results show that our tool facilitates important use cases that are not covered by currently employed strategies and it received far more positive feedback compared to these strategies, despite its technical limitations.

Data Availability We provide Zenodo records for the open-source implementation of our tool[6] as well as our questionnaire and anonymized responses.[7]

Acknowledgments The work of Elias Kuiter, Sebastian Krieter, and Jacob Krüger has been supported by a pure-systems Go SPLC 2019 Challenge project.

Bibliography

[Ku19] Kuiter, Elias; Krieter, Sebastian; Krüger, Jacob; Leich, Thomas; Saake, Gunter: Foundations of Collaborative, Real-Time Feature Modeling. In: Proc. Int'l Systems and Software Product Line Conf. (SPLC). ACM, pp. 257–264, September 2019.

[Ku21] Kuiter, Elias; Krieter, Sebastian; Krüger, Jacob; Saake, Gunter; Leich, Thomas: variED: An Editor for Collaborative, Real-Time Feature Modeling. Empirical Software Engineering (EMSE), 26(2), March 2021.

[SC02] Sun, Chengzheng; Chen, David: Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems. ACM Trans. on Computer-Human Interaction (TOCHI), 9(1):1–41, 2002.

[6] https://doi.org/10.5281/zenodo.4259912
[7] https://doi.org/10.5281/zenodo.4259914

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 87

Summary: Social Science Theories in Software Engineering Research

Tobias Lorey[1], Paul Ralph[2], Michael Felderer[3]

Abstract: Human aspects are becoming increasingly important in software engineering research. Despite this, it is unclear how software engineering utilizes existing theories from social sciences. As a result, the objective of this critical review is to assess which and how social science theories are used in five high-quality software engineering research journals. 87 unique social science theories from disciplines such as psychology, management, and economics are identified. However, the results show that less than two percent of articles employ a social science theory, and there are still challenges with incorporating social science theories in software engineering research.

Keywords: software engineering research; theories; social science

1 Summary

This summary reports on the paper *Social Science Theories in Software Engineering Research* [LRF22] which was published in the proceedings of the *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. We performed a critical review to investigate the social science theory-use in five high-quality software engineering journals. All research articles published in *ACM Transactions on Software Engineering and Methodology (TOSEM), Empirical Software Engineering (EMSE), IEEE Transactions on Software Engineering (TSE), Information and Software Technology (IST)*, and *Journal of Systems and Software (JSS)* between 2007 and 2019 were electronically searched for the term "theory". Non-research publications, such as editorials and commentaries, were excluded. Each search result was manually inspected whether a theory has actually been used and meets the definition of originating from a social science discipline.

Information from all relevant primary studies such as the theory name, author, and journal name have been extracted. We then classified theories according to various criteria such as the originating discipline, where in the paper the theory was used and for which purpose.

We identified 87 individual social science theories used in one or more software engineering research articles during the 13 year time span reviewed. This implies that less than two

[1] University of Innsbruck, Innsbruck, Austria, tobias.lorey@student.uibk.ac.at
[2] Dalhousie University, Halifax, Canada, paulralph@dal.ca
[3] University of Innsbruck, Innsbruck, Austria, michael.felderer@uibk.ac.at

percent of published articles in the field use a social science theory. The identified 87 theories originate mostly from psychology (34), management including information systems (23), and economics (10). We found that theories are most commonly used in the method and introduction sections. Theories were most often used to facilitate the design of a research method including the formulation of hypotheses and surveys and as an explanation of study results. The most used theory is the technology acceptance model (TAM) followed by diffusion of innovations theory, and dual coding theory.

We also sent a questionnaire to authors who published articles that we identified as primary studies. We asked them about their perceived challenges and benefits when using social science theories. Respondents to the questionnaire reported difficulty when using theories. It was noted that *everybody does it differently* and problems arise when looking for suitable social concepts to explain software engineering phenomena. Reported benefits included not needing to re-invent the wheel, allowing to relate human and technical aspects, and being able to use established concepts from other disciplines to answer questions in software engineering research. Given how human factors are increasingly important to investigating the field's phenomena, social science theories can help researchers grow and mature software engineering as a discipline.

2 Data Availability

We provide the anonymized data package at https://doi.org/10.5281/zenodo.6036076. It contains primary studies, inclusion and exclusion criteria, meta data of articles using theories and theory classifications.

References

[LRF22] Lorey, Tobias; Ralph, Paul; Felderer, Michael: Social Science Theories in Software Engineering Research. In: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). IEEE, pp. 1994–2005, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 89

Understanding and Predicting Typed Links in Issue Tracking Systems

Clara Marie Lüders, Abir Bouraffa, Tim Pietz, Walid Maalej[1]

Abstract: This talk summarizes two recent papers: "Beyond Duplicates: Towards Understanding and Predicting Link Types in Issue Tracking Systems" accepted at MSR 2022, and "Automated Detection of Typed Links in Issue Trackers" accepted at RE 2022. In issue trackers like JIRA, stakeholders connect issues via links of certain types, such as Epic-, Block-, Duplicate-, or Relate-links. While previous research focused on Duplicate-links, we aim at understanding and predicting other link types. In the MSR paper, we studied issues linking in 15 public JIRA repositories. We evaluated the robustness of state-of-the-art duplicate detection approaches on our dataset with diversified link types. We found that current deep-learning approaches confuse duplicates and other links. Extending the training sets with other link types partly solves this problem. In the RE paper, we trained and evaluated various machine learning models to detect typed links. We found that a BERT model trained on titles and descriptions of linked issues outperforms other deep learning models, achieving an average macro F1-score of 0.64. We also studied what impacts the prediction performance and found that this depends on how repositories are used (e.g. linking quality) and by whom.

Keywords: Issue Tracking System; Typed Link Detection; Dependency Management; Deep Learning

1 Summary of Papers

Software projects use Issue Tracking Systems (ITS) like JIRA to track and organize issues and their workflows. Issues are often interconnected via different links, such as the default JIRA link types Duplicate, Relate, Block, or Subtask. These links are essential for stakeholders to quickly find the information they want. However, managing and finding the links between issues in large ITS is challenging.

2 MSR22 Paper: Beyond Duplicates: Towards Understanding and Predicting Link Types in Issue Tracking Systems

Previous research has primarily focused on analyzing and predicting duplication links, but this work aims to understand the various other link types, their prevalence, and their characteristics. We studied 607,208 links connecting 698,790 issues in 15 public JIRA repositories [MLM22]. Besides the default types, the custom types Depend, Incorporate,

[1] Universität Hamburg, Abteilung, Vogt-Kölln-Straße 30, 22527 Hamburg, Deutschland [vorname.nachname]@uni-hamburg.de

Split, and Cause were also common. Motivated by the differences between the link types and their popularity, we evaluated the robustness of two state-of-the-art duplicate detection approaches [De17, He20] from the literature on the JIRA dataset. We found that current deep-learning approaches confuse Duplicate and other links in almost all repositories. On average, the classification accuracy dropped by 6% for one approach and 12% for the other. Extending the training sets with other link types partly solves this issue.

3 RE22 Paper: Automated Detection of Typed Links in Issue Trackers

In addition to analyzing the link types, we also examined how state-of-the-art machine learning models can automatically detect common link types in software projects. A BERT model trained on the titles and descriptions of linked issues significantly outperformed other deep learning models, achieving an average macro F1-score of 0.64 for detecting nine popular link types across all repositories (weighted F1-score of 0.73). The model performed exceptionally well on Subtask- and Epic-links, achieving F1-scores of 0.89 and 0.97, respectively. We found that Relate-links often get confused with the other links, which suggests that they are likely used as default links in unclear cases. We also observed significant differences across the repositories. We discuss different implementation strategies based on these findings. If we restrict the detection to links and non-links, the classifier achieves an average F1-score of 0.95.

4 Data Availability

The data set is available in zenodo[2] and both papers have a replication package[3][4].

Literaturverzeichnis

[De17] Deshmukh, Jayati; Annervaz, K. M.; Podder, Sanjay; Sengupta, Shubhashis; Dubash, Neville: Towards Accurate Duplicate Bug Retrieval Using Deep Learning Techniques. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, USA, S. 115–124, 2017.

[He20] He, Jianjun; Xu, Ling; Yan, Meng; Xia, Xin; Lei, Yan: Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks. In: Proceedings of the 28th International Conference on Program Comprehension. Association for Computing Machinery, New York, NY, USA, S. 117–127, 2020.

[MLM22] Montgomery, Lloyd; Lüders, Clara; Maalej, Walid: An Alternative Issue Tracking Dataset of Public Jira Repositories. In: 2022 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE, USA, 2022.

[2] https://doi.org/10.5281/zenodo.5882881

[3] https://github.com/RegenKordel/LYNX-BeyondDuplicates

[4] https://github.com/RegenKordel/LYNX-TypedLinkDetection

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 91

Empirical research on requirements quality: a systematic mapping study

Lloyd Montgomery[1] Davide Fucci[2] Abir Bouraffa[3] Lisa Scholz[4] Walid Maalej[5]

Abstract: Dieser Artikel ist ursprünglich im Requirements Engineering Journal in 2022 erschienen. Bisherige Forschung hat wiederholt gezeigt, dass qualitativ hochwertige Anforderungen maßgeblich für den Erfolg von Entwicklungsprojekten sind. Obwohl der Begriff "Qualität" im Bereich des Requirements Engineering allgegenwärtig ist und trotz umfangreicher Forschungsarbeiten, gibt es bisher keine Metastudien, die einen Überblick über Qualitätsattribute geben und diese miteinander vergleichen. Daher haben wir eine systematische Literaturstudie durchgeführt: Wir haben 6905 Artikel aus sechs akademischen Datenbanken abgerufen und auf 105 relevante Primärstudien heruntergebrochen. Diese nutzen empirische Forschung zur Definition, Verbesserung und Bewertung der Anforderungsqualität. Unsere Ergebnisse zeigen, dass die empirische Forschung zur Anforderungsqualität bislang hauptsächlich auf Verbesserungstechniken fokussiert, während sich nur wenige Primärstudien mit Definitionen und Bewertungen von Qualitätsattributen befassen. Von den 12 identifizierten Qualitätsattributen sind Mehrdeutigkeit, Vollständigkeit, Konsistenz und Korrektheit von Anforderungen die bekanntesten. Wir haben 111 Untertypen von Qualitätsattributen identifiziert, wie beispielsweise "Template- Konformität" für Konsistenz oder "passive Form" für Mehrdeutigkeit. Nur wenige Arbeiten haben bislang gezielt Qualitätsattribute von spezifischen Arten von Anforderungen, wie z.B. Anwendungsfälle oder User Stories, untersucht. Unsere Ergebnisse verdeutlichen somit die Notwendigkeit weiterer empirisch fundierter Forschung zur Definition von Anforderungsqualität unter Einsatz vielfältigerer Forschungsmethoden sowie zur Untersuchung eines breiteren Spektrums an Anforderungstypen.

Keywords: Systematic mapping study; Secondary study; Requirements quality; Empirical research

1 Zusammenfassung des Artikels

Requirements Engineering (RE) ist ein wichtiger Bestandteil von Softwareprojekten, der u.a. zu einer Reihe von Artefakten wie User Stories, Spezifikationsdokumente, Use Cases, oder Feature Requests führt. Die "Qualität der Anforderungen" bezieht sich auf die Qualität dieser Artefakte und umfasst Aspekte wie Lesbarkeit, Mehrdeutigkeit, Konsistenz und Überprüfbarkeit (um nur einige zu nennen). Dieser Forschungsbereich hat in der letzten Dekade zu einer Reihe von Veröffentlichungen auf führenden RE-Konferenzen geführt. Trotz der gestiegenen Aufmerksamkeit fehlte bisher ein ganzheitlicher Überblick über die empirische Erforschung der Anforderungsqualität. Um diese Lücke zu schließen, haben wir

[1] University of Hamburg, 20146 Hamburg, Germany lloyd.montgomery@uni-hamburg.de
[2] Blekinge Tekniska Högskola, Valhallavägen 1, 371 41 Karlskrona, Sweden davide.fucci@bth.se
[3] University of Hamburg, 20146 Hamburg, Germany abir.bouraffa@uni-hamburg.de
[4] University of Hamburg, 20146 Hamburg, Germany lisa.scholz@uni-hamburg.de
[5] University of Hamburg, 20146 Hamburg, Germany walid.maalej@uni-hamburg.de

eine systematische Mapping-Studie (SMS) durchgeführt und die Ergebnisse im RE Journal veröffentlicht [Mo22].

Unsere Studie bestand aus vier Hauptphasen: Artikelsuche, Artikelauswahl, Datenextraktion und Kartierung. Bei der Artikelsuche haben wir sechs Datenbanken verwendet: ACM Digital Library, IEEE Xplore, Elsevier ScienceDirect, SpringerLink, Web of Science und Google Scholar. Unsere Datenbankrecherche ergab 6.905 Artikel. Während der Auswahl der Artikel und der Datenextraktion reduzierten wir diese Zahl systematisch auf unsere endgültigen 105 Primärstudien.

Aus den 105 Primärartikeln haben wir 12 Typen von Anforderungsqualität und 111 Untertypen ermittelt. Die 12 Typen sind: Mehrdeutigkeit, Vollständigkeit, Komplexität, Konsistenz, Korrektheit, Redundanz, Relevanz, Wiederverwendbarkeit, Nachvollziehbarkeit, Verständlichkeit, Überprüfbarkeit und undefiniert (bzw. unklar). Eine vollständige Liste der Untertypen ist in dem Originalartikel zu finden [Mo22]. Wir haben eine große Vielfalt und ein großes Ausmaß an untersuchten Themen und Codes für Qualitätsattribute gefunden, die sich hauptsächlich auf Mehrdeutigkeit, Vollständigkeit, Konsistenz und Korrektheit konzentrieren. Es gibt einige Studien, die sich mit der Verbesserung der Anforderungsqualität befassen, aber nur wenige, die die Anforderungsqualität definieren oder evaluieren. Studien, die explizit Teilnehmenden einbeziehen, haben einen Median von 20 Teilnehmenden, während Studien, die Ersteller von Wahrheitsdatensätzen einbeziehen, einen viel niedrigeren Median von 4 Teilnehmern haben. Bei den Primärstudien ist die Strenge angemessen (mit Raum für Verbesserungen), während die Relevanz gering ist. Die Granularität der untersuchten Artefakte reicht vom allgemeinen Konzept der “Anforderungen” bis hin zur spezifischen Aufschlüsselung von Wörtern und Sätzen.

Unser Artikel diskutiert die Mängel an Definitionen und Bewertungen der Anforderungsqualität. Wir sind der Meinung, dass der Schwerpunkt auf die Definition von Qualität verlagert werden soll, damit Bemühungen zur Qualitätsverbesserung darauf ausgerichtet sind, wie die Entwicklungsteams mit Requirementsdokumenten arbeiten. Anschließend wird die Notwendigkeit einer stärkeren Beteiligung der Industrie sowie eine größeren Vielfalt der Anforderungsarten diskutiert.

2 Dataverfügbarkeit

Alle Daten und Skripte sind in unserem Replication Package online verfügbar[6].

Literaturverzeichnis

[Mo22] Montgomery, Lloyd; Fucci, Davide; Bouraffa, Abir; Scholz, Lisa; Maalej, Walid: Empirical research on requirements quality: a systematic mapping study. Requirements Engineering, S. 1–27, 2022.

[6] https://doi.org/10.5281/zenodo.5510222

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 93

On the validity of pre-trained transformers for natural language processing in the software engineering domain

Julian von der Mosel[1], Alexander Trautsch[2], Steffen Herbold[3]

Abstract: We summarize the article *On the validity of pre-trained transformers for natural language processing in the software engineering domain* [VTH22], which was published in the IEEE Transactions on Software Engineering in 2022.

Keywords: Defect Prediction; Costs; Return On Investment

1 Overview

The article "On the validity of pre-trained transformers for natural language processing in the software engineering domain" was published in the IEEE Transactions on Software Engineering in 2022. Transformers are the current state-of-the-art of natural language processing in many domains and are using traction within software engineering research as well. Such models are pre-trained on large amounts of data, usually from the general domain. However, we only have a limited understanding regarding the validity of transformers within the software engineering domain, i.e., how good such models are at understanding words and sentences within a software engineering context and how this improves the state-of-the-art. Within this article, we shed light on this complex, but crucial issue. We compare BERT transformer models trained with software engineering data with transformers based on general domain data in multiple dimensions: their vocabulary, their ability to understand which words are missing, and their performance in classification tasks.

2 Results

Our results show that for tasks that require understanding of the software engineering context, pre-training with software engineering data is valuable, while general domain models are sufficient for general language understanding, also within the software engineering domain.

[1] Georg-August-Universität Göttingen, Fakultät für Mathematik und Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland

[2] Universität Passau, Fakultät für Informatik und Mathematik, Dr.-Hans-Kapfinger-Straße 30, 94032 Passau, Deutschland alexander.trautsch@uni-passau.de

[3] Universität Passau, Fakultät für Informatik und Mathematik, Dr.-Hans-Kapfinger-Straße 30, 94032 Passau, Deutschland steffen.herbold@uni-passau.de

3 Data Availability

The seBERT model we pre-trained for this work, as well as all code to reproduce our experiments, is available online.[4]

Literatur

[VTH22] Von der Mosel, J.; Trautsch, A.; Herbold, S.: On the validity of pre-trained transformers for natural language processing in the software engineering domain. IEEE Transactions on Software Engineering/, S. 1–1, 2022.

[4] https://github.com/smartshark/seBERT

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 95

Evaluation of Usability Criteria Addressed by Static Analysis Tools on a Large Scale

Marcus Nachtigall[1] Michael Schlichtig[2] Eric Bodden[3]

Abstract: Static analysis tools support developers in detecting potential coding issues, such as bugs or vulnerabilities. Research emphasizes technical challenges of such tools but also mentions severe usability shortcomings. These shortcomings hinder the adoption of static analysis tools, and user dissatisfaction may even lead to tool abandonment. To comprehensively assess the state of the art, we present the first systematic usability evaluation of a wide range of static analysis tools. We derived a set of 36 relevant criteria from the literature and used them to evaluate a total of 46 static analysis tools complying with our inclusion and exclusion criteria - a representative set of mainly non-proprietary tools. The evaluation against the usability criteria in a multiple-raters approach shows that two thirds of the considered tools offer poor warning messages, while about three-quarters provide hardly any fix support. Furthermore, the integration of user knowledge is strongly neglected, which could be used for instance, to improve handling of false positives. Finally, issues regarding workflow integration and specialized user interfaces are revealed. These findings should prove useful in guiding and focusing further research and development in user experience for static code analyses.

Keywords: Automated static analysis; Software usability

1 Large Scale Usability Criteria Evaluation of Static Analysis Tools

Static analysis tools allow the analysis of program code without execution. Their results can help developers identify code issues. However, research has shown that static analysis tools often provide weak usability, which leads to dissatisfaction among developers [CB16, Jo13], e.g., insufficient explanations can lead to misjudgment of reports as false positives, too many false positives can cause distrust in a tool overall, and workflow integration is often lacking.

To comprehensively assess the state of the art of static analysis tools with respect to usability, we performed a large-scale empirical study where we compiled a list of 243 recommended static analysis tools [NSB22]. We derived the usability evaluation criteria starting with the collection of Nachtigall et al. [NDB19] and extended the collection with an additional literature review. Nachtigall et al. have grouped recurrent usability issues into six categories, which are connected to understandable *warning messages*, *fix support*, *false positives*, *integration of user feedback*, *workflow integration*, and a specialized *user interface*. We

[1] Heinz Nixdorf Institute, Paderborn University, Germany marcus.nachtigall@uni-paderborn.de
[2] Heinz Nixdorf Institute, Paderborn University, Germany michael.schlichtig@uni-paderborn.de
[3] Heinz Nixdorf Institute, Paderborn University & Fraunhofer IEM Paderborn, Germany eric.bodden@uni-paderborn.de

defined the evaluation criteria, evaluation protocol employing a multiple-raters approach, and inclusion/exclusion criteria. In total, 46 static analysis tools were evaluated [NSB22].

Tab. 1: Overview of tools considered, excluded, and included

	C/C++	Java	Python	C#	JavaScript	Multi-Lang.	Total
All considered tools	69	52	28	18	23	53	243
Excl.: Proprietary	29	19	0	4	3	26	81
Excl.: Not maintained	15	8	2	4	7	5	41
Excl.: Unable to install	8	4	2	0	3	3	20
Excl.: Other reasons	12	8	7	4	5	19	55
Included	5	13	17	6	5	0	46

Table 1 shows the number of collected, excluded, and included tools. What is striking is that about 25% could not be installed or were not maintained anymore. Full information on each considered tool is provided in our artifact (cf. Section 2).

2 Data Availability

All raw data is available at: `https://sites.google.com/view/datatoolsurvey/`. The artifact contains all collected tools (with reasoning), the list of evaluation criteria, data on tool evaluation per programming language and a table with all evaluated tools [NSB22].

Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297

Bibliography

[CB16] Christakis, Maria; Bird, Christian: What Developers Want and Need from Program Analysis: An Empirical Study. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ASE 2016, Association for Computing Machinery, New York, NY, USA, p. 332–343, 2016.

[Jo13] Johnson, B.; Song, Y.; Murphy-Hill, E.; Bowdidge, R.: Why don't software developers use static analysis tools to find bugs? In: 2013 35th International Conference on Software Engineering (ICSE). pp. 672–681, 2013.

[NDB19] Nachtigall, Marcus; Do, Lisa Nguyen Quang; Bodden, Eric: Explaining Static Analysis-A Perspective. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW). IEEE, pp. 29–32, 2019.

[NSB22] Nachtigall, Marcus; Schlichtig, Michael; Bodden, Eric: A Large-Scale Study of Usability Criteria Addressed by Static Analysis Tools. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2022, Association for Computing Machinery, New York, NY, USA, p. 532–543, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 97

BeDivFuzz: Integrating Behavioral Diversity into Generator-based Fuzzing — Summary

Hoang Lam Nguyen[1], Lars Grunske[1]

Abstract: This paper summarizes our work "BeDivFuzz: Integrating Behavioral Diversity into Generator-based Fuzzing" [NG22], presented at the 44th International Conference on Software Engineering (ICSE 2022).

Keywords: Structure-aware fuzzing; behavioral diversity; random testing

1 Summary

Recent advances in structure-aware fuzzing (e.g., [Pa19, Ng21, SP21]) have been focusing on testing the actual core functionality of the software under test (SUT), going beyond the early input processing stages. Effectively testing programs that expect complex structured inputs (e.g., compilers) is challenging, mainly because the test inputs must (i) be *syntactically* valid to be successfully parsed, (ii) satisfy any additional *semantic* validity constraints to actually reach the SUT's core functionality, and (iii) exhibit some sort of *diversity* to trigger a variety of different program behavior.

When it comes to diversely testing the SUT, simply *covering* new program behaviors is not sufficient, since faults may only be revealed by specific inputs. Additionally, a fuzzer may be biased towards exploring particular behaviors, due to the structure of the SUT and random nature of input mutations. As a result, we argue that *behavioral diversity* of a fuzz campaign is only given if the fuzzer not only covers many different branches (i.e., high richness), but also diversely tests many of these behaviors (i.e., high evenness w.r.t. branch execution distribution).

BeDivFuzz aims to diversely test the SUT by leveraging a particular way to generate structured test inputs: imperative generators, which depend on a sequence of choices to produce concrete inputs [Pa19]. Our key insight is that by analyzing the internal structure of these generators, we can classify these choices into *structural* and *value* choices. As a result, by controlling either type of choices only, we can perform more fine-grained and controlled mutations in the space of syntactically valid inputs. For instance, by mutating the structural choices only, we can perform *structure-changing* mutations in the input (e.g., changing the shape of an XML-tree). On the other hand, by controlling only the value choices, we can perform *structure-preserving* mutations (e.g., changing attribute values). We leverage these two types of mutations in the following way:

1. We search for interesting *input structures* in the space of valid inputs through *structure-changing* mutations.
2. We produce different variants of the same input structure by applying *structure-preserving* mutations with the goal of exploring diverse execution traces.

[1] Humboldt-Universität zu Berlin, Department of Computer Science, Unter den Linden 6, 10099 Berlin, Germany
{nguyehoa,grunske}@informatik.hu-berlin.de

BeDivFuzz integrates this idea in a feedback-guided fuzzing loop to produce test inputs that not only increase code coverage, but also improve the diversity of the executed behaviors. In particular, our approach retains interesting input structures for further mutation, and uses an ϵ-greedy approach to bias the mutation strategy towards diverse execution traces.

To evaluate our approach, we use *Hill numbers* [Hi73], an established biodiversity index from the field of ecology. Inspired by the STADS (*Software Testing and Analysis as Discovery of Species*) framework introduced by Böhme [Bö18], we consider branches as species and compute the Hill Numbers of orders $q = 0, 1, 2$ over the distribution of unique branch execution counts. We call this metric *behavioral diversity* of a fuzzing campaign. An interesting property of this metric is that it can be mapped to other well-known measures of diversity, like species richness (which in our case corresponds to branch coverage), Shannon entropy, and the Simpson index [Hi73]. Our experimental evaluation indicates that BeDivFuzz significantly improves behavioral diversity compared to the baselines.

2 Data Availability

We have published a replication package on Zenodo [2], which includes the experimental data and instructions to reproduce the results. In addition, the most recent version of the tool is currently maintained on GitHub [3].

Literaturverzeichnis

[Bö18] Böhme, Marcel: STADS: Software Testing as Species Discovery. ACM Transactions on Software Engineering and Methodology (TOSEM), 27(2):1–52, 2018.

[Hi73] Hill, Mark O: Diversity and evenness: a unifying notation and its consequences. Ecology, 54(2):427–432, 1973.

[Ng21] Nguyen, Hoang Lam; Nassar, Nebras; Kehrer, Timo; Grunske, Lars: MoFuzz: A Fuzzer Suite for Testing Model-Driven Software Engineering Tools. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. ASE '20, Association for Computing Machinery, New York, NY, USA, S. 1103–1115, 2021.

[NG22] Nguyen, Hoang Lam; Grunske, Lars: BeDivFuzz: Integrating Behavioral Diversity into Generator-Based Fuzzing. In: Proceedings of the 44th International Conference on Software Engineering. ICSE '22, Association for Computing Machinery, New York, NY, USA, S. 249–261, 2022.

[Pa19] Padhye, Rohan; Lemieux, Caroline; Sen, Koushik; Papadakis, Mike; Le Traon, Yves: Semantic Fuzzing with Zest. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2019, Association for Computing Machinery, New York, NY, USA, S. 329–340, 2019.

[SP21] Srivastava, Prashast; Payer, Mathias: Gramatron: Effective Grammar-Aware Fuzzing. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA 2021, Association for Computing Machinery, New York, NY, USA, S. 244–256, 2021.

[2] https://zenodo.org/record/6320055

[3] https://github.com/hub-se/BeDivFuzz

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 99

A Summary of ReVision: History-based Model Repair Recommendations

Manuel Ohrndorf[1] Christopher Pietsch[2] Udo Kelter[3] Lars Grunske[4] Timo Kehrer[5]

Abstract: This work reports recent research results on history-based model repair recommendations in Model-Driven Engineering (MDE), originally published in Reference [Oh21]. Models in MDE are primary development artifacts that are heavily edited in all software development stages and can become temporarily inconsistent during editing. Model repair tools can support developers by proposing a list of the most promising repairs. Such repair recommendations will only be accepted in practice if the generated proposals are plausible and understandable and the set as a whole is manageable.

Our interactive repair tool ReVision [Oh18], aims at generating repair proposals for inconsistencies introduced by past incomplete edit steps. Such an incomplete edit step is either undone or extended to the full execution of a consistency-preserving edit operation. We evaluate our approach using histories of real-world models from popular open-source modeling projects. Our experimental results confirm our hypothesis that most of the inconsistencies can be resolved by complementing incomplete edits. In fact, 92.2% of the proposed complementations could be observed in the model history.

Keywords: model-driven software engineering; model repair; consistency; recommendations; history analysis

1 Summary

Model-Driven Engineering (MDE) raises the level of abstraction in software engineering by using models as primary artifacts. Thus, models in MDE are subject to continuous evolution and heavily edited during all stages of development and maintenance. As a consequence, models may get inconsistent for various reasons, e.g., due to misunderstandings when being edited collaboratively in teams. Technically, one main reason for consistency violations is the isolated editing of interrelated views or model fragments.

Inconsistencies are detected as violations of consistency rules defined for a specific modeling language. Violations of these rules can be automatically obtained using inconsistency detection techniques. While these techniques are widely established in practice, how to optimally support developers in resolving inconsistencies is still being actively discussed.

[1] Universität Bern, Switzerland manuel.ohrndorf@unibe.ch
[2] University Siegen, Germany cpietsch@informatik.uni-siegen.de
[3] University Siegen, Germany kelter@informatik.uni-siegen.de
[4] Humboldt-Universität zu Berlin, Germany grunske@informatik.hu-berlin.de
[5] Universität Bern, Switzerland timo.kehrer@unibe.ch

Repairing all inconsistencies in a single step often leads to solutions whose rationale is hard to grasp for developers. Following the generally accepted debugging strategy of fixing a single defect at a time, we iteratively repair each single violation of a consistency rule.

We assume that inconsistencies are introduced by past, incomplete editing processes that require additional changes to achieve a new consistent state. In general, there are many alternatives to resolve such inconsistencies. In such cases, recommender systems can generate a ranked list of suitable repair proposals from which a developer can choose.

Our repair recommendation tool ReVision requires specifying the transitions between *consistent* states by formal consistency-preserving edit operations (CPEOs). The main idea of our approach is to consider CPEOs as ideal edit operations and to recommend the "gap" between ideal edits and the edits which have caused an inconsistency as model repairs [Oh18, Oh21]. Therefore, incomplete edit steps are detected in the model history and can be either undone or extended to the full execution of a CPEO.

A systematic process supports the specification of CPEOs capturing typical complex edit steps, which are likely to be applied only partially, leading to model inconsistencies. Specifically, we follow an example-driven approach to manually specify sets of minimal yet valid example model fragments, which are then automatically composed into CPEOs.

We evaluate our approach using histories of real-world models obtained from popular open-source modeling projects. Our empirical study shows that, in most observable inconsistency repair cases, it is more likely that a developer wants to catch up on missing changes. In fact, 92.2% (510 complementations, 43 undos) of our repair proposals that could equally be observed (44 not observable) in the original modeling history are complementations.

In general, approaches that are not aware of the change history of a model are likely to propose repairs that just undo former changes that caused an inconsistency. Thus, a developer should be enabled to make an informed decision whether to undo former work or, if this work was just incomplete, retain and complete it.

1.1 Data Availability

ReVision and the evaluation data can be found at `https://repairvision.github.io/`.

Bibliography

[Oh18] Ohrndorf, Manuel; Pietsch, Christopher; Kelter, Udo; Kehrer, Timo: ReVision: a tool for history-based model repair recommendations. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. ACM, pp. 105–108, 2018.

[Oh21] Ohrndorf, Manuel; Pietsch, Christopher; Kelter, Udo; Grunske, Lars; Kehrer, Timo: History-based model repair recommendations. ACM Transactions on Software Engineering and Methodology (TOSEM), 30(2):1–46, 2021.

Jicer: Slicing Android Apps for Cooperative Analysis

Felix Pauck[1] Heike Wehrheim[2]

Abstract: Slicing allows to identify which program parts influence or are influenced by a certain statement of a program. Hence, if we know which statement is potentially causing an issue we can slice accordingly to only inspect the slice while debugging. With JICER, we proposed a slicer that can be used in a different context, namely cooperative Android app analysis. In combination with taint analysis tools, we employed JICER to get more accurate results.

Keywords: Cooperative Analysis; Android; Taint Analysis; Static Slicing

1 Cooperative Analysis with JICER

Android has become the most-used operating system, consequently it has also become a compelling target for attackers who, for example, try to steal users' private data. One instrument to detect privacy leaks before they are exploited is taint analysis. Fortunately, there exist many Android (taint) analysis tools that focus on different aspects related to the Android framework or an app's programming language. Even more luckily, there nowadays exist cooperative analysis frameworks that allow to compose (and evaluate) analysis combinations [PW19, PBW18]. With JICER [PW21] (Jimple Slicer) we proposed a static Android app slicer which is usable in cooperative analysis context. It allows slicing an app such that it can still be analyzed thereafter — a feature that distinguishes JICER form related Android slicing approaches. In the following the approach behind JICER is explained in more detail.

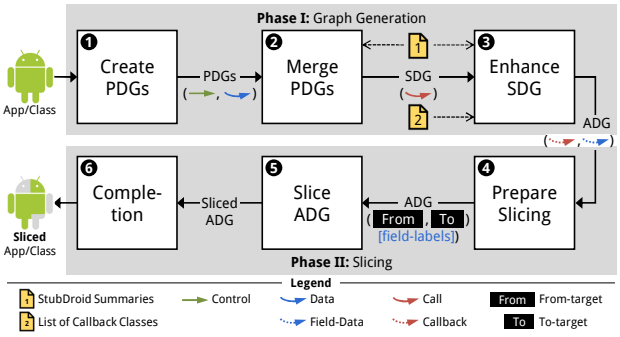


Fig. 1: Overview of the approach

JICER implements a six-step workflow that can be divided into two phases: graph generation and slicing (see Figure 1).

[1] Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany fpauck@mail.uni-paderborn.de
[2] University of Oldenburg, Ammerländer Heerstraße 114–118, 26129 Oldenburg, Germany heike.wehrheim@uol.de

Once an Android app (`.apk` file) is provided as input, a *program dependence graph (PDG)*, modeling control and data dependencies, is computed for all methods contained in this app (❶). The second step (❷) merges these PDGs into a single *system dependence graph (SDG)* as proposed by Horwitz et al. back in 1990 [HRB90]. This SDG is then enhanced with edges that model callbacks (with respect to e.g. life-cycles or GUI elements) and data dependencies related to fields. The output of this enhancement step (❸) is called *app dependence graph (ADG)*. With the generation of the ADG the graph generation phase ends.

To start the slicing phase, the slicing criteria provided by the user are identified first (❹). They define *from* where and *to* where to slice. Note that JICER allows to provide a to-criterion, a from-criterion or both. Depending on the criteria given, a backward slice (to), a forward slice (from) or a chop (from-to) is computed. Figuratively speaking, a chop can be understood as the intersection of a backward and a forward slice. Once the ADG is sliced (❺), finishing touches are performed during Step ❻. Statements that are mandatory for analyses but not included in the slice are added to the slice. For example, `setContentView` statements link a layout to an Android activity, hence, many analyses require such statements to be available to know which layout must be taken into account. At the end, the sliced Android package (`.apk` file) is output. JICER also supports other output formats but `.apk` files are the preferred choice in cooperative analysis context, because most analysis tools require `.apk` files as input. Accordingly, all these tools can be used to analyze slices produced by JICER.

The evaluation presented in the proposing paper [PW21] shows that JICER is able to slice real-world apps thereby reducing app size about ~55 to ~96%. Most importantly, in a cooperative analysis JICER can increase the overall precision (eliminating up to 82% of false positives that have been found without slicing).

2 Data Availability

An artifact submitted along with the JICER study, that has successfully undergone an artifact evaluation, is available at Zenodo (`https://doi.org/10.5281/zenodo.5462859`). It contains all tools and results determined in the original study. Furthermore, the up-to-date version of JICER can be found at Github (`https://FoelliX.github.io/Jicer`).

Bibliography

[HRB90] Horwitz, Susan; Reps, Thomas W.; Binkley, David W.: Interprocedural Slicing Using Dependence Graphs. ACM Trans. Program. Lang. Syst., 12(1):26–60, 1990.

[PBW18] Pauck, Felix; Bodden, Eric; Wehrheim, Heike: Do Android taint analysis tools keep their promises? In: Proceedings of the 26th ESEC/FSE, 2018. ACM, 2018.

[PW19] Pauck, Felix; Wehrheim, Heike: Together strong: cooperative Android app analysis. In: Proceedings of ESEC/FSE, 2019. ACM, 2019.

[PW21] Pauck, Felix; Wehrheim, Heike: Jicer: Simplifying Cooperative Android App Analysis Tasks. In: Proceedings of the 21st SCAM, 2021. IEEE, 2021.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 103

Variable Misuse Detection: Software Developers versus Neural Bug Detectors

Cedric Richter[1], Jan Haltermann[2], Marie-Christine Jakobs[3], Felix Pauck[4], Stefan Schott[5], Heike Wehrheim[6]

Abstract: Finding and fixing software bugs is a central part of software development. Developers are therefore often confronted with the task of identifying whether a code snippet contains a bug and where it is located. Recently, data-driven approaches have been employed to automate this process. These so called neural bug detectors are trained on millions of buggy and correct code snippets to *learn* the task of bug detection. This raises the question how the performance of neural bug detectors and software developers compare. As a first step, we study this question in the context of variable misuse bugs. To this end, we performed a study with over 100 software developers and two state-of-the-art approaches for neural bug detection. Our study shows that software developers are on average slightly better than neural bug detectors – even though the bug detectors are trained specifically for this task. In addition, we identified several bottlenecks in existing neural bug detectors which could be mitigated in the future to improve their bug detection performance.

Keywords: Bug detection; variable misuse bugs; empirical study

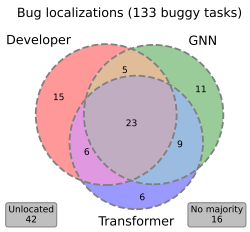
A Study of Developers and Neural Bug Detectors

Data-driven methods like neural bug detectors are becoming increasingly effective at the task of bug detection [He20]. Existing bug detectors often focus on frequent bug types such as variable misuses (a variable name is used although another was meant) or binary operator bugs (the wrong binary operator is used). While we can evaluate the bug detectors easily, e.g. on bugs mined from public repositories [KS20], it is unclear how human software developers would perform on these bug detection tasks and how they compare to existing neural bug detectors. To be able to compare software developers and neural bug detectors, we conducted a study with over 100 developers and evaluated them on the task of detecting variable misuse bugs in Java.

Our study was conducted in the form of a web survey, for which we developed a customized web interface. The interface is shown in Figure 1a. Here, the participant is shown a random

[1] University of Oldenburg, Oldenburg, Germany cedric.richter@uol.de
[2] University of Oldenburg, Oldenburg, Germany jan.haltermann@uol.de
[3] Technical University of Darmstadt, Darmstadt, Germany jakobs@cs.tu-darmstadt.de
[4] Paderborn University, Paderborn, Germany fpauck@mail.upb.de
[5] Paderborn University, Paderborn, Germany stefan.schott@upb.de
[6] University of Oldenburg, Oldenburg, Germany heike.wehrheim@uol.de

(a) User interface of the developer study



(b) Overlap of localized bugs

code snippet written in Java and must decide whether (1) the code snippet contains a variable misuse bug and (2) on which line the bug is located (if any). In total, participants are confronted with up to eight code snippets that were drawn randomly from a manually curated set of 310 possible code snippets and either contain a variable misuse bug or not. In the end, 304 snippets were solved by at least two participants. For each of these snippets, we computed the average developer performance (by aggregating all answers of this snippet) which we use to compare developers and bug detectors.

We compared the average developer performance with the performance of two state-of-the-art neural bug detectors on the same set of code snippets. Our results show that there is a significant overlap of bugs that can be detected by developers and bug detectors (e.g. as shown in Figure 1b). Still, we found that developers are generally better in avoiding false positives while maintaining a high bug detection rate. This observation has also led us to the discovery of several limitations of existing neural bug detectors such as (1) a misleading training distribution, (2) a missing robustness towards code length and (3) a lack of code context. A detailed evaluation and discussion of our results is available in [Ri22].

Data Availability

All our study results and an explorable version of the web survey are archived and available at Zenodo[7]. Our replication package also include participant answers (in an anonymous form).

Bibliography

[He20] Hellendoorn, Vincent J.; Sutton, Charles; Singh, Rishabh; Maniatis, Petros; Bieber, David: Global Relational Models of Source Code. In: ICLR. OpenReview.net, 2020.

[KS20] Karampatsis, Rafael-Michael; Sutton, Charles: How Often Do Single-Statement Bugs Occur?: The ManySStuBs4J Dataset. In: MSR. ACM, pp. 573–577, 2020.

[Ri22] Richter, Cedric; Haltermann, Jan; Marie-Christine, Jakobs; Felix, Pauck; Stefan, Schott; Wehrheim, Heike: Are Neural Bug Detectors Comparable to Software Developers on Variable Misuse bugs? In: ASE. 2022.

[7] https://doi.org/10.5281/zenodo.6958242

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 105

Introducing FUM: A Framework for API Usage Constraint and Misuse Classification

Michael Schlichtig[1], Steffen Sassalla[2], Krishna Narasimhan[3], Eric Bodden[4]

Abstract: Application Programming Interfaces (APIs) are the primary mechanism developers use to obtain access to third-party algorithms and services. Unfortunately, APIs can be misused, which can have catastrophic consequences, especially if the APIs provide security-critical functionalities like cryptography. Understanding what API misuses are, and how they are caused, is important to prevent them, e.g., with API misuse detectors. However, definitions for API misuses and related terms in literature vary. This paper presents a systematic literature review to clarify these terms and introduces FUM, a novel Framework for API Usage constraint and Misuse classification. The literature review revealed that API misuses are violations of API usage constraints. To address this, we provide unified definitions and use them to derive FUM. To assess the extent to which FUM aids in determining and guiding the improvement of an API misuses detector's capabilities, we performed a case study on the state-of the-art misuse detection tool CogniCrypt. The study showed that FUM can be used to properly assess CogniCrypt's capabilities, identify weaknesses and assist in deriving mitigations and improvements.

Keywords: API misuses; API usage constraints; classification framework; API misuse detection; static analysis

1 Classification of API Usage Constraints and Misuses with *FUM*

Reuse of functionalities and algorithms is key to software development and is enabled by APIs. However, misusing an API can cause serious consequences, such as program crashes or data leakage. We performed a systematic literature review on API misuses from November 2020 to February 2021 to improve our understanding of API misuses. We considered 69 publications to derive definitions and our classification **F**ramework for API **U**sage constraints and **M**isuses (*FUM*) [Sc22].

FUM is mainly based on the works of Monperrus et al. [Mo12], Amann et al. [Am19] and Li's refinement [Li20] on the work of Amann et al. [Am19]. API misuses are caused by the violation of an API usage constraint which is a restriction imposed by the API designer or expert to the API but cannot be checked by the programming language's compiler. Therefore,

[1] Heinz Nixdorf Institute, Paderborn University, Germany michael.schlichtig@uni-paderborn.de
[2] Hasso Plattner Institute, University of Potsdam, Germany steffen.sassalla@student.hpi.de
[3] Technische Universität Darmstadt, Germany kri.nara@cs.tu-darmstadt.de
[4] Heinz Nixdorf Institute, Paderborn University & Fraunhofer IEM Paderborn, Germany eric.bodden@uni-paderborn.de

FUM defines API usage constraint types and localizes them with the related part(s) of an API method call, e.g., a *Post-Call* is located at the *return value* (cf. Figure 1).

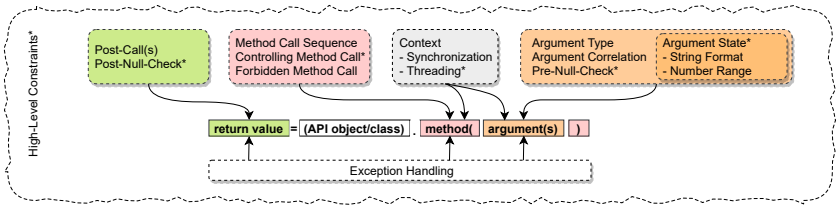


Fig. 1: *FUM* [Sc22] - Overview of API usage constraint types associated with parts of an API method call. Types marked with an asterisk are additions to the work of Monperrus et al. [Mo12]. Dashed colored boxes are specific to one single API method call part. Uncolored dashed boxes are API usage constraint types spanning multiple parts of an API method call.

2 Data Availability - Case Study

FUM was tested in a case study to evaluate a state-of-the-art cryptographic API misuse detector [Kr20] . The data of the case study is available at `https://doi.org/10.6084/m9.figshare.16832749` and contains the evaluation protocols of each API misuse sample and detailed theoretical explanations of suggested improvements.

Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297

Bibliography

[Am19] Amann, S.; Nguyen, H. A.; Nadi, S.; Nguyen, T. N.; Mezini, M.: A Systematic Evaluation of Static API-Misuse Detectors. IEEE Transactions on Software Engineering, 45(12):1170–1188, 2019.

[Kr20] Krüger, Stefan : CogniCrypt - The Secure Integration of Cryptographic Software. Ph.D. thesis, University Paderborn, October 2020.

[Li20] Li, Xia: An Integrated Approach for Automated Software Debugging via Machine Learning and Big Code Mining. Ph.D. thesis, The University of Texas at Dellas, 2020.

[Mo12] Monperrus, Martin; Eichberg, Michael; Tekes, Elif; Mezini, Mira: What should developers be aware of? An empirical study on the directives of API documentation. Empirical Software Engineering, 17(6):703–737, Dec 2012.

[Sc22] Schlichtig, Michael; Sassalla, Steffen; Narasimhan, Krishna; Bodden, Eric: FUM - A Framework for API Usage constraint and Misuse Classification. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 673–684, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 107

GenBenchDroid: Fuzzing Android Taint Analysis Benchmarks

Stefan Schott[1], Felix Pauck[2]

Abstract: The conventional approach of assessing the performance of Android taint analysis tools consists of applying the tool to already existing benchmarks and calculating its performance on the contained benchmark cases. Creating and maintaining a benchmark requires a lot of effort, since it needs to comprise various analysis challenges, and since each benchmark case needs a well documented ground-truth — otherwise one cannot know whether a tool's analysis is accurate. This effort is further increased by the frequently changing Android API. All these factors lead to the same, usually manually created, benchmarks being reused over and over again. In consequence analysis tools are often *over-adapted* to these benchmarks.

To overcome these issues we propose the concept of *benchmark fuzzing*, which allows the generation of previously unknown and unique benchmarks, alongside their ground-truths, at evaluation time. We implement this approach in our tool GenBenchDroid and additionally show that we are able to find analysis faults that remain uncovered when solely relying on the conventional benchmarking approach.

Keywords: Fuzzing; Benchmarks; Android Taint Analysis

1 Fuzzing Benchmarks with GenBenchDroid

Taint flows typically consist of a source, which introduces sensitive data to the app, a sink, which leaks sensitive data to the outside world and some intermediate data flow that connects source and sink. This data flow often comprises various programming *aspects*, like different data structures or multi-threading. The more complex the data flow is, the harder it is for analysis tools to uncover the taint flow. Thus, a proper benchmark needs to comprise benchmark cases with various degrees of complexity and analysis challenges (aspects). To be able to generate such Android apps, that can be used as benchmark cases, we split aspects that may possibly be contained inside taint flows into a set of *modules*. GenBenchDroid [SP22] interweaves these modules by inserting them into a *template*, which denotes the starting structure of an Android app. This allows GenBenchDroid to generate Android apps that comprise taint flows of various complexities.

Figure 1 shows an overview of GenBenchDroid's architecture. The *Fuzzer* component uses a grammar that knows about the aforementioned templates and modules (building blocks). This grammar is used to generate a *Benchmark Case Blueprint* (BCB),

[1] Paderborn University, Germany, Warburger Str. 100, 33098 Paderborn, Germany, stefan.schott@upb.de
[2] Paderborn University, Germany, Warburger Str. 100, 33098 Paderborn, Germany, fpauck@mail.upb.de

which specifies building blocks and their desired insertion order. The *Benchmark Case Generator* (BCG) component interweaves the building blocks that are specified in the BCB and generates an Android app, as well as the corresponding ground-truth. This ground-truth is determined by generating a graph that represents the used building blocks and by finding paths in this graph that connect source and sink modules.

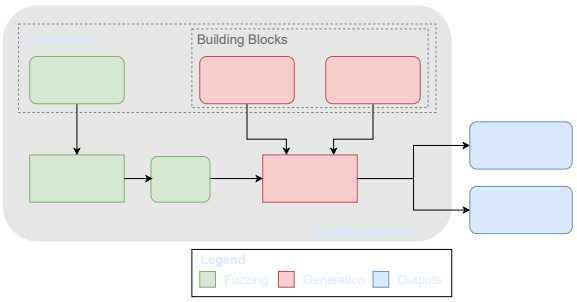


Fig. 1: Overview of GenBenchDroid

Benchmark Fuzzing offers many advantages that can improve and complement conventional benchmarking approaches. By design, benchmark fuzzing decreases the likelihood of over-adaptation, as benchmark cases are not known before evaluation time. Furthermore, our experiments on state-of-the-art taint analysis tools (FlowDroid [Ar14] and Amandroid [WRO18]) uncovered previously unknown analysis defects. We were able to uncover a scalability issue in Amandroid by generating benchmark cases of various sizes. Additionally, we uncovered analysis defects in FlowDroid and Amandroid that would only show up, if aspects appear in combination inside a single taint flow. These defects can hardly be uncovered by only relying on conventional benchmarks, as it is impossible to manually create arbitrary aspect combinations and since these aspects used in isolation are analyzed correctly by both tools.

2 Data Availability

GenBenchDroid and its source code, as well as all data that is related to our performed experiments is available at `https://doi.org/10.5281/zenodo.7023084`. An up-to-date version of GenBenchDroid can be found on Github (`https://github.com/stschott/GenBenchDroid`).

Bibliography

[Ar14] Arzt, Steven; Rasthofer, Siegfried; Fritz, Christian; Bodden, Eric; Bartel, Alexandre; Klein, Jacques; Le Traon, Yves; Octeau, Damien; McDaniel, Patrick: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices, 49(6):259–269, 2014.

[SP22] Schott, Stefan; Pauck, Felix: Benchmark Fuzzing for Android Taint Analyses. In: 22nd IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2022, Limassol, Cyprus, October 3-4, 2022. IEEE, 2022. To appear.

[WRO18] Wei, Fengguo; Roy, Sankardas; Ou, Xinming: Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. ACM Transactions on Privacy and Security (TOPS), 21(3):1–32, 2018.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 109

Quantifying the Potential to Automate the Synchronization of Variants in Clone-and-Own – Summary

Alexander Schultheiß[1], Paul Maximilian Bittner[2], Thomas Thüm[3], Timo Kehrer[4]

Abstract: We report about a recent empirical study on variant synchronization in clone-and-own, originally published at the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME) 2022 [Sc22]. In clone-and-own, a new variant of a software system is created by copying and adapting an existing one. While it is flexible, clone-and-own causes high maintenance effort in the long run as cloned variants evolve in parallel; certain changes, such as bug fixes, need to be propagated between variants. A recent line of research proposes to automate such synchronization tasks when migration to a software product line is not feasible. However, it is yet unclear how far this synchronization can actually be pushed. We present an empirical study in which we quantify the potential to automate the synchronization of variants in clone-and-own. We simulate automated variant synchronization using the history of BusyBox, a real-world multi-variant software system. Our results indicate that existing patching techniques propagate changes with an accuracy of up to 85%, if applied consistently from the start of a project. This can be even further improved to 93% by exploiting lightweight domain knowledge about which features are affected by a change, and which variants implement affected features.

Keywords: clone-and-own, variant synchronization, version control, software product lines

Summary

Today's software is often released in multiple variants to meet varying requirements. While there are systematic approaches to managing variability, such as software product lines where all variants are managed using an integrated platform, these approaches are not feasible for all projects. Instead, developers fall back to using clone-and-own, creating a new variant of a software system by copying and adapting an existing one (e. g., using branching/forking capabilities of a version control system). This way, new variants are created ad-hoc and without requiring upfront investments or knowledge about future variants. In the long term, however, clone-and-own projects suffer from ever-increasing maintenance costs. For example, if a bug is discovered and fixed in one variant, it is often unclear which other variants are affected by the same bug and how it should be fixed in these variants.

Researchers started to explore the continuum between ad-hoc clone-and-own and software product lines to reduce the burden on developers. In our project *VariantSync* [Ke21], we

[1] Humboldt-Universität zu Berlin, Germany, alexander.schultheiss@informatik.hu-berlin.de
[2] University of Ulm, Germany, paul.bittner@uni-ulm.de
[3] University of Ulm, Germany, thomas.thuem@uni-ulm.de
[4] University of Bern, Switzerland, timo.kehrer@inf.unibe.ch

propose to manage the development and maintenance of cloned variants by propagating changes of interest between them, thereby keeping cloned variants synchronized. To this end, we advocate to collect additional (lightweight) domain knowledge on cloned variants and software changes, which might be leveraged by change propagation techniques. However, it is yet unclear how far this synchronization can actually be pushed.

In this work [Sc22], we quantify the potential to automate the synchronization of variants in clone-and-own through an empirical study. Our study covers three aspects of patch-based variant synchronization. First, to gain insight on the difficulties of automated change propagation, we investigate how often propagating a change via patching succeeds or fails, depending on different levels of patch granularity (i. e., commit-, file-, and line-level patches). Second, we examine the correctness of automated patching by analyzing the outcome of each synchronization scenario. Third, we investigate the potential to improve the correctness of automated synchronization when developers document lightweight domain knowledge. Specifically, we employ lightweight domain knowledge about which features are affected by a change and which variants implement affected features – knowledge that is generally assumed to be available but typically undocumented in clone-and-own.

We inspect almost half a billion patch scenarios derived from a large-scale real-world system (BusyBox). We find that the majority of patches is applicable automatically, even when propagating changes blindly across variants. Blind patching produces correct results in the majority of cases with an accuracy of 85% and precision of 92%. This shows that the very applicability of patches is a useful indicator for determining if a target variant should actually receive a patch or not. Furthermore, we confirm the hypothesis that gathering lightweight domain knowledge might prove useful for automated synchronization. If automated patching with lightweight domain knowledge is applied from the start of a project, variants can be synchronized with high accuracy (93%) and almost perfect precision (97%).

Data Availability

The publication will be accessible under the DOI 10.1109/ICSME55016.2022.00032. A preprint can be found on our website (`https://seg.inf.unibe.ch/papers/SBTK_ICSME22.pdf`). Our artifact is available on Github (`https://github.com/VariantSync/SyncStudy`) and Zenodo (DOI: 10.5281/zenodo.7025599).

Bibliography

[Ke21] Kehrer, Timo; Thüm, Thomas; Schultheiß, Alexander; Bittner, Paul Maximilian: Bridging the Gap Between Clone-and-Own and Software Product Lines. In: Proc. Int'l Conf. on Software Engineering (ICSE). IEEE, pp. 21–25, 2021.

[Sc22] Schultheiß, Alexander; Bittner, Paul Maximilian; Thüm, Thomas; Kehrer, Timo: Quantifying the Potential to Automate the Synchronization of Variants in Clone-and-Own. In: Proc. Int'l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2022. To appear.

Gregor Engels. Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 111

Property-Driven Black-Box Testing of Numeric Functions

Arnab Sharma[1] Vitalik Melnikov[2] Eyke Hüllermeier[3] Heike Wehrheim[4]

Abstract:

In this work, we propose a *property-driven* testing mechanism to perform unit testing of functions performing numerical computations. Our approach, similar to the property-based testing technique, allows the tester to specify the requirements to check. Unlike property-based testing, the specification is then used to generate test cases in a targeted manner. Moreover, our approach works as a *black-box* testing tool, i.e. it does not require knowledge about the internals of the function under test. Therefore, besides on *programmed* numeric functions, we also apply our technique to machine-learned regression models. The experimental evaluation on a number of case studies shows the effectiveness of our testing approach.

Keywords: Property-based testing; Regression; Testing machine-learning models

1 Property-Driven Testing

Our testing approach is a form of *learning-based testing* (LBT) [Me], where – given a black-box system under test (SUT) – a *model* of the SUT is *learned*, and thereafter test cases are generated on the model. We in particular employ LBT for learning models of machine-learned functions. Based on this, we develop an approach which in addition to LBT allows for property specification and systematically generates test cases based on the property. Our approach consists of the following steps.

Property specification. We aim to generate test cases based on a requirement specification given by the user. In this, we follow the style used by property-based testing which takes the following form: $Assume \Rightarrow Assert$, where the $Assume$ specifies a pre-condition on the input, and $Assert$ specifies a post-condition on the output of the SUT. These logical conditions involve standard predicate logic using integers or real numbers and basic arithmetic and Boolean operators.

Test data generation. Once we have the property specification and the SUT, the next step is to use the property to generate test cases on the SUT. To this end, we first of all, *learn* a model using standard machine learning techniques. Since we consider numerical functions here, we learn regression models, to be precise either decision trees or neural networks. Next, the learned model as well as the negation of the property are translated into logical formulas. The conjunction of these two formulae is then given to an SMT solver. If the solver finds a (logically) satisfiable model to the formula as a counterexample to the property,

[1] Universität Paderborn, arnab.sharma@upb.de, [2] Universität Paderborn, melnikov@mail.upb.de, [3] LMU München, eyke@ifi.lmu.de, [4] Universität Oldenburg, heike.wehrheim@uol.de

Tab. 1: Results of detected violations (✓ = violation detected, ✗ = no violation detected)

SUT / *Property*	L-OWA MLC/PT	L-Uni MLC/PT	LAF MLC/PT	DeepSet MLC/PT	Total MLC/PT
infimum	✗/✗	✗/✗	✓/✓	✓/✓	2/2
supremum	✗/✗	✗/✗	✓/✓	✓/✓	2/2
monotonicity	✓/✓	✓/✗	✓/✓	✓/✓	**4/3**
Lipschitz	✓/✓	✓/✓	✓/✓	✓/✗	**4/3**
symmetry	✗/✗	✗/✗	✗/✗	✗/✗	0/0
idempotency	✗/✗	✓/✗	✓/✓	✓/✓	**3/2**
conjunction	✓/✓	✗/✗	✓/✓	✓/✓	3/3
disjunction	✓/✓	✓/✓	✓/✓	✓/✗	**4/3**
internality	✗/✗	✓/✗	✓/✓	✓/✓	**3/2**
invariance	✗/✗	✓/✗	✓/✓	✓/✓	**3/2**
additivity	✓/✓	✗/✗	✓/✗	✓/✗	**3/1**
total	5/5	**6/2**	**10/9**	**9/6**	**30/22**

we cross-check it on the SUT. This is necessary since we calculated the logical formula from the model approximating the SUT. If the counterexample is also valid for the SUT itself, we are done with testing and return the counterexample as a violation of the property. Otherwise, we use the counterexample to retrain the model.

We have implemented our approach as part of our existing tool MLCHECK and have evaluated it on a number of *predefined* (i.e., implemented) and machine-learned numeric functions [Sh]. Table 1 shows some experimental results for 4 machine-learned SUTs testing for 11 properties (all common to aggregation functions). The table also gives a comparison of our approach (MLC) with property-based testing (PT)[5]. The blue-shaded cells are cases where the property holds for the SUT.

Data Availability

Our artifact can be found at https://github.com/arnabsharma91/MLCHECK-formalise.

Bibliography

[Me] Meinke, Karl: Learning-Based Testing: Recent Progress and Future Prospects. In: Machine Learning for Dynamic Software Analysis: Potentials and Limits - International Dagstuhl Seminar 16172, Germany, 2016.

[Sh] Sharma, Arnab; Melnikov, Vitalik; Hüllermeier, Eyke; Wehrheim, Heike: Property-Driven Testing of Black-Box Functions. In: 10th IEEE/ACM International Conference on Formal Methods in Software Engineering, FormaliSE@ICSE 2022, Pittsburgh, PA, USA, 2022.

[5] https://github.com/HypothesisWorks/hypothesis

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 113

Input Invariants

Dominic Steinhöfel[1], Andreas Zeller[2]

Abstract: To exhaustively test a program, we need inputs that the program does not reject. Such *valid* inputs must satisfy syntactic and semantic constraints of the input language. Grammar-based fuzzers efficiently produce *syntactically* valid system inputs but miss context-sensitive *semantic* constraints. Example semantic properties are length fields or checksums in binary inputs or definition-use constraints for variables in programming languages. We introduce ISLa [SZ22a], a *declarative specification language for context-sensitive properties* of structured system inputs. An ISLa specification, or *input invariant*, consists of a context-free *grammar* and a potentially context-sensitive ISLa *constraint*. Our ISLa *fuzzer* produces streams of inputs from invariants. We show that a few ISLa constraints suffice to generate diverse and 100% semantically valid inputs. Additionally, the fuzzer can *repair* and—preserving semantics—*mutate* inputs. Provided sample inputs, a program property, or both, our *ISLearn* prototype *mines* precise invariants. In follow-up work, we used ISLearn for *diagnosing failures*: "The *heartbleed* vulnerability is triggered if `length` exceeds the length of `payload`."

Keywords: fuzzing; specification language; grammars; constraint mining

1 Introduction

To improve a faulty program, one needs to (1) *test* the program to discover—and reproduce—a bug, (2) *debug* it to isolate the problem, and (3) *repair* the fault. The outcomes of step (1) is a bug-triggering *input*. Step (2) provides a sound *hypothesis*; and step (3) a *validated fix*. All these steps require understanding the language of program inputs and, ideally, its outputs. This understanding enables us to exhaustively test a program, evolve a failure hypothesis as a property of program inputs, and thoroughly validate a fix.

Our ISLa specification language can be used to describe a *theory of system inputs*. This theory facilitates *automating* the steps outlined above. ISLa constraints are *declarative*, human-readable, and can be used for testing, program understanding, and failure diagnosis.

2 Specifying Input Invariants with ISLa

An ISLa specification consists of a grammar and constraints. For example, the grammar for the TLS heartbeat extension allows us to decompose a heartbeat request into the sequence "`0x1 <length> <payload> <padding>`." Using an ISLa constraint, we can specify that

[1] CISPA Helmholtz Center for Information Security, Saarbrücken dominic.steinhoefel@cispa.de
[2] CISPA Helmholtz Center for Information Security, Saarbrücken zeller@cispa.de

<length> determines the length of <payload>: "uint16(<length>) = len(<payload>)." When specifying constraints in ISLa, one can use all function symbols in the SMT-LIB theory catalog, in particular from the theory of strings. In addition, ISLa supports *structural* relations (e.g., "before," "inside") and can be *extended* with domain-specific semantic predicates. Two types of quantifiers enable the *precise selection of input elements*. All these constituents result in an expressive yet tractable, grammar-aware string constraint language.

Our ISLa fuzzer generates system inputs by heuristically exploring the input language space, using modern SMT solvers (e.g., Z3) and custom strategies (e.g., for quantifiers) to solve constraints. Our evaluation, based on input languages ranging from C over XML to TAR, demonstrates that the fuzzer efficiently generates *100% precise inputs* from a few lines of ISLa specs while *exercising more language features* than a coverage-based grammar fuzzer.

3 Learning Input Specifications

Writing good specifications takes time, although the effort is well invested: *Code* changes frequently, but *system input specs* (e.g., of protocols or file formats) stay stable for *decades*. Luckily, tools like Mimid [GMZ20] mine *grammars* from parsers. Our ISLearn tool learns semantic constraints from sample inputs, a program property, or both. It instantiates *patterns* obtained from our ISLa case studies and enriched by simple string properties to find input invariants. It derived semantic properties for Graphviz DOT, ICMP Echo, and Racket with an accuracy of 78% to 97%, demonstrating that our patterns apply to new scenarios. In follow-up work, we extended this approach to find diagnoses of program failures for debugging. We integrated *machine learning* to restrict the search to relevant language features, tremendously improving performance; furthermore, an automated *feedback loop* refines candidate hypotheses. Our experimental results on real bug reports show that the generated diagnoses are close to those provided by human developers and that the enhanced tool outperforms both ISLearn and a previously proposed automated debugging tool.

Data Availability

Our ISLa and ISLearn artifacts are publicly available [SZ22b]. The current versions of the prototypes can be downloaded from `https://github.com/rindPHI/{isla|islearn}`.

References

[GMZ20] Gopinath, R.; Mathis, B.; Zeller, A.: Mining Input Grammars from Dynamic Control Flow. In: ESEC/FSE 2020. ACM, 2020.

[SZ22a] Steinhöfel, D.; Zeller, A.: Input Invariants. In: ESEC/FSE 2022. ACM, 2022.

[SZ22b] Steinhöfel, D.; Zeller, A.: Replication Package for "Input Invariants", `https://doi.org/10.1145/3554336`, ACM, 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 115

Automatisierte Identifikation von sicherheitsrelevanten Konfigurationseinstellungen mittels NLP

Patrick Stöckle[1] Theresa Wasserer[1] Bernd Grobauer[2] Alexander Pretschner[1]

Abstract: Dieser Vortrag wurde auf der 37. IEEE/ACM International Conference on Automated Software Engineering (ASE) präsentiert [St22]. Um Computerinfrastrukturen zu sichern, müssen die verantwortlichen Administratoren alle sicherheitsrelevanten Einstellungen konfigurieren und sichere Werte einsetzen. Hierbei stützen sie sich auf Sicherheitsexperten, die die sicherheitsrelevanten Einstellungen identifizieren und in Sicherheitskonfigurationsrichtlinien dokumentieren. Das Identifizieren der sicherheitsrelevanten Einstellungen ist allerdings zeitaufwändig und teuer, weshalb ihm oft keine Priorität beigemessen wird. Um dieses Problem zu lösen, nutzen wir aktuelle Verfahren der Computerlinguistik, um Einstellungen auf der Grundlage ihrer Beschreibung in natürlicher Sprache als sicherheitsrelevant zu klassifizieren. Allerdings zeigt unsere Evaluation, dass die trainierten Klassifikatoren nicht gut genug sind, um die menschlichen Sicherheitsexperten vollständig zu ersetzen sondern höchstens bei der Klassifizierung der Einstellungen helfen können. Durch die Veröffentlichung unserer gelabelten Datensätze und all unserer Modelle wollen wir Sicherheitsexperten bei der Analyse von Konfigurationseinstellungen unterstützen und weitere Forschung in diesem Bereich ermöglichen.

Keywords: Systemhärtung; Sicherheitskonfiguration; Computerlinguistik

Ein kritischer Teil der IT-Sicherheit in einer Organisation wie Siemens ist die sichere Konfiguration aller verwendeten Software. Hierfür müssen wir wissen, welche Konfigurationseinstellungen einer Software sicherheitsrelevant (SR) oder nicht-sicherheitsrelevant (NSR) sind. Wenn wir alle möglichen Einstellungen einer Software durchgehen, um alle SR Einstellungen zu finden, ist dies eine langwierige und zeitraubende Aufgabe. Daher lagern wir diesen Prozess an Organisationen wie das Center for Internet Security (CIS) aus. Das CIS bietet Sicherheitskonfigurationsrichtlinien für verschiedene Softwaresysteme, und wir können die CIS-Richtlinien verwenden, um unsere Software zu härten.

Es gibt jedoch Situationen, in denen dies nicht möglich ist: Erstens, wenn es keine CIS-Richtlinie für eine Software gibt, oder zweitens, wenn es ein neues Update der Software gibt und die CIS ihre Empfehlungen für das Update noch nicht veröffentlicht hat. Drittens, wenn wir in unserer Umgebung höhere Sicherheitsanforderungen haben und zusätzliche Regeln benötigen. Bei Siemens ist der dritte Anwendungsfall der wichtigste. In allen drei Fällen müssen die Sicherheitsexperten alle SR Einstellungen finden. Um das Auffinden der SR Einstellungen zu unterstützen und sicherzustellen, dass wir alle SR Einstellungen finden, wäre eine automatische Klassifizierung wünschenswert. Hierbei sind falsch-negative Ergebnisse (die Einstellung ist SR, aber wir klassifizieren sie als NSR) schwerwiegender

[1] Technische Universität München, Lehrstuhl für Software und Systems Engineering, Boltzmannstr. 3, 85748 Garching b. München, Deutschland vorname.nachname@tum.de
[2] Siemens AG, München, Deutschland bernd.grobauer@tum.de

als falsch-positive Ergebnisse. Ein Angreifer könnte eine falsch-negative, nicht gehärtete Einstellung ausnutzen, um das System anzugreifen. Ziel ist es also, dass Klassifikatoren falsch-negative Ergebnisse vermeiden, ohne jedoch jede Einstellung als SR zu kennzeichnen.

Unser laufendes Beispiel ist die automatische Härtung des Betriebssystems Windows 10 [SGP21] mit über 4500 Einstellungen. Außerdem gibt es eine CIS Windows 10-Richtlinie mit über 500 Regeln. Jede Regel bezieht sich dabei auf jeweils eine Einstellung. Im Mai 2021 veröffentlichte Microsoft das Update 21H1 für Windows 10 mit über 300 neuen Einstellungen. Daher mussten die Sicherheitsexperten bei Siemens nun die neuen SR Einstellungen identifizieren. Da das Durchsuchen nach SR Einstellungen langwierig und mühsam ist, forderten die Experten dafür automatisierte Unterstützung.

In diesem Artikel stellen wir unseren Vorschlag für diese Unterstützung vor. Wir verwenden verschiedene moderne Verfahren der Computerlinguistik (engl. natural language processing (NLP)), um automatisch zu klassifizieren, ob eine Einstellung SR ist. Um SR Begriffe zu identifizieren, verwenden wir bestehende Leitfäden. Anschließend nutzen wir die Beschreibungen der Einstellungen als Eingabe für die Klassifizierung.

Unser Forschungsbeitrag besteht aus drei Teilen: Erstens stellen wir den ersten Ansatz vor, der NLP-Techniken zur Identifizierung von SR Einstellungen verwendet. Zweitens ermöglichen wir durch die Veröffentlichung unserer gelabelten Datensätze, dass andere Forscher ihre Modelle ebenfalls auf ihnen trainieren können, um die beschriebene Problematik zu lösen. Drittens teilen wir den Code unserer Modelle, damit Sicherheitsexperten letztere bei der Richtlinienerstellung verwenden können.

Data Availability

Unser Datensatz ist öffentlich auf GitHub verfügbar.[3] Außerdem haben wir den Python-Code all unserer Modelle als Jupyter Notebooks auf Kaggle veröffentlicht.[4]

Literatur

[SGP21] Stöckle, P.; Grobauer, B.; Pretschner, A.: Automated Implementation of Windows-Related Security-Configuration Guides. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. ASE '20, Association for Computing Machinery, Virtual Event, Australia, 2021, URL: https://doi.org/10.1145/3324884.3416540.

[St22] Stöckle, P.; Wasserer, T.; Grobauer, B.; Pretschner, A.: Automated Identification of Security-Relevant Configuration Settings Using NLP. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. ASE '22, IEEE/ACM, Association for Computing Machinery, Rochester, MI, USA, 2022, URL: https://doi.org/10.1145/3551349.3559499.

[3] github/tum-i4/Automated-Identification-of-Security-Relevant-Configuration-Settings-Using-NLP

[4] Modell 1: kaggle/tumin4/sentiment-analysis, Modell 2: kaggle/tumin4/topic-modeling-and-latent-dirichlet-allocation, Modell 3: kaggle/tumin4/transformer-based-machine-learning

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 117

Generating Review Models to Validate Safety Requirements

Bastian Tenbergen[1], Thorsten Weyer[2]

Abstract: This talk discusses our approach for automatically generating review models for safety-critical systems presented in the paper [TW21] published in the Feb. '21 issue of the Journal of Software and Systems Modeling. We present a semi-automated formal approach and tool support to generate Hazard Relation Diagrams. Enabled by mitigation tables, the approach consists of two transformation steps using OMG's QVTo language [OMG16].

Keywords: Safety requirements; Hazards; Validation; Adequacy; Modeling relation diagrams

1 Introduction

Developing safety-critical systems includes identifying hazards and defining corresponding mitigations at the earliest possible stage of development. The thusly identified hazard-mitigating requirements (HMRs) must be valid with regard to the intended functionality and render the system sufficiently safe during operation. Yet, validating HMRs is burdened by the fact that hazards and their HMRs are a work product of safety assessment and requirements engineering, respectively. These work products are poorly integrated such that during validation, the information needed to determine the adequacy of HMRs is not available to stakeholders, potentially leading to lingering covert safety hazards. To aid in the validation of HRMs, we have proposed, improved, and evaluated [TWP18] a novel diagram type called "Hazard Relation Diagrams" (HRDs) which represent requirements, hazards, and their mitigation together in a single review model.

2 Generating Hazard Relation Diagrams

HRDs are generated from UML activity diagrams containing hazard-inducing requirements (HIRs) and tables containing the results of functional hazard analysis (FHA). HRDs contain one hazard per mitigation, however may contain multiple mitigation partitions for HMRs in geometrically distant areas in the activity diagram. HRDs are generated by first specifying HMRs using a mitigation template containing deletions, additions, and substitutions of activity diagram model elements documenting the HIRs. One mitigation

[1] State University of New York at Oswego, Dept. of Computer Science, NY, USA bastian.tenbergen@oswego.edu
[2] Technische Hochschule Mittelhessen, Gießen, Germany thorsten.weyer@mni.thm.de

template corresponds to one mitigation partition in the diagram that highlights the changes from HIRs to HMRs. Afterwards, the hazard, trigger conditions, and the safety goal from the FHA are appended into the activity diagram with HMRs, thus completing the HRD.

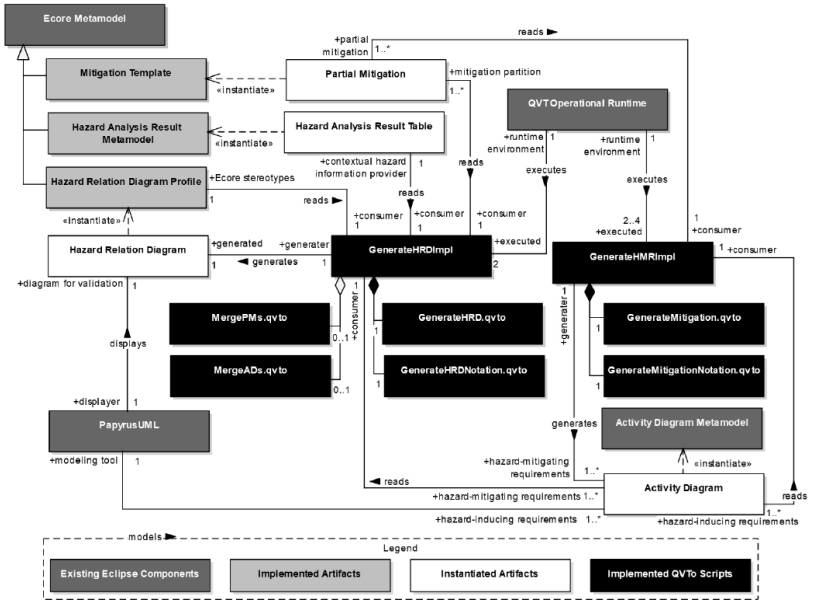


Fig. 1: Structure and Technical Interplay of the Tool Prototype Components.

Based on a canonical artifact formalization, transformation scripts have been implemented using OMG's Query/View/Transformation Operational Mappings language [OMG16] to implement the generation process. The metamodel of the tool prototype is shown in Fig. 1.

3 Data Availability

QVTo scripts, proof-of-concept implementation, and empirical data on the effectiveness of HRDs are available at `https://github.com/tenbergen/hazardrelationdiagrams`.

References

[OMG16] Object Management Group: Query/View/Transformation (QVT) Specification. OMG Document Number formal/2016-06-03., 2016. Version 1.3.

[TW21] Tenbergen, Bastian; Weyer, Thorsten: Generation of hazard relation diagrams: formalization and tool support. Software and Systems Modeling, 20(1):175–210, Feb 2021.

[TWP18] Tenbergen, Bastian; Weyer, Thorsten; Pohl, Klaus: Hazard Relation Diagrams: a diagrammatic representation to increase validation objectivity of requirements-based hazard mitigations. Requirements Engineering, 23(2):291–329, Jun 2018.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 119

Exploring the relationship between performance metrics and cost saving potential of defect prediction models

Steffen Tunkel[1] Steffen Herbold[2]

Abstract: We summarize the article *Exploring the relationship between performance metrics and cost saving potential of defect prediction models* [TH22], which was published in Empirical Software Engineering in 2022.

Keywords: Defect prediction; Performance metrics; Cost saving potential; Exploratory research

1 Overview

The article "Exploring the relationship between performance metrics and cost saving potential of defect prediction models" was published in Empirical Software Engineering in 2022. Performance metrics are a core component of the evaluation of any machine learning model and used to compare models and estimate their usefulness. Recent work started to question the validity of many performance metrics for this purpose in the context of software defect prediction. Within this study, we explore the relationship between performance metrics and the cost saving potential of defect prediction models. We study whether performance metrics are suitable proxies to evaluate the cost saving capabilities and derive a theory for the relationship between performance metrics and cost saving potential. We measure performance metrics and cost saving potential in defect prediction experiments. We use a multinomial logit model, decision, and random forest to model the relationship between the metrics and the cost savings.

2 Results

We could not find a stable relationship between cost savings and performance metrics. We attribute the lack of the relationship to the inability of performance metrics to account for the property that a small proportion of very large software artifacts are the main driver of the costs. Any defect prediction study interested in finding the best prediction model, must consider cost savings directly, because no reasonable claims regarding the economic benefits of defect prediction can be made otherwise.

[1] Universität Göttingen, Deutschland
[2] Universität Passau, Fakultät für Informatik und Mathematik, Dr.-Hans-Kapfinger-Straße 30, 94032 Passau, Deutschland steffen.herbold@uni-passau.de

3 Data Availability

The data and all analysis scripts are available online [He22].

Literatur

[He22] Herbold, S.: sherbold/replication-kit-defect-prediction- metrics: v1.0, Version v1,0, Aug. 2022, URL: https://doi.org/10.5281/zenodo.6992179.

[TH22] Tunkel, S.; Herbold, S.: Exploring the relationship between performance metrics and cost saving potential of defect prediction models. Empirical Software Engineering 27/7, Sep. 2022, URL: https://doi.org/10.1007/s10664-022-10224-4.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 121

A comprehensive empirical evaluation of generating test suites for mobile applications with diversity – Summary

Thomas Vogel[1] Chinh Tran[1] Lars Grunske[1]

Abstract: In this extended abstract, we summarize our work on analyzing the fitness landscape of the search-based app testing problem and building on that, improving and evaluating a specific solution for this problem. This work has been published under the title of "A comprehensive empirical evaluation of generating test suites for mobile applications with diversity" in the journal Information and Software Technology (IST) in 2021 [VTG21].

Keywords: Mobile apps; Search-based testing; Fitness landscape analysis

Context. In search-based software testing, we often use popular heuristics with default or best-practice configurations to automatically generate tests. Such an out-of-the-box use typically leads to suboptimal results, for instance, in terms of achieved coverage of the software under test. To yield better results, costly trial-and-error experiments are performed to find suitable configurations of a heuristic for a given search problem [AF13]. One example in this context is SAPIENZ [MHJ16] that uses a default NSGA-II heuristic to generate test suites for mobile applications (apps) without adapting this heurisitc to this specific testing problem. Consequently, a promising way to improve the effectiveness of SAPIENZ could be the identification and use of suitable configurations of NSGA-II for this specific problem.

Objective. Focusing on app testing, our objective was to improve the effectivenss of SAPIENZ while avoiding costly trial-and-error experiments to identify suitable configurations of the used NSGA-II. Particularly, we wanted to analytically understand the search problem of SAPIENZ and use this understanding to identify suitable configurations in an informed way. To achieve this objective, we performed a *fitness landscape analysis* [PA12, ME13] of SAPIENZ and used the obtained results to sytematically adapt the NSGA-II heuristic of SAPIENZ. While the analysis of SAPIENZ has been conducted earlier [VTG19], a comprehensive evaluation of our adaptation of SAPIENZ has been presented more recently [VTG21]. In the context of search-based testing, our work is novel as it targets the testing of mobile apps while others have analyzed the fitness landscape for the problem of unit testing (e.g., [AMG17]).

Method. Our fitness landscape analysis focused on the genotypic diversity and evolvability, that is, how the evolved test suites are spread in the search space and evolve over time regarding their fitness (achieved coverage, detected faults, and length of the test cases). We particularly selected diversity as a major aspect for our analysis since it is considered

[1] Humboldt-Universität zu Berlin, Software Engineering Group, Unter den Linden 6, 10099 Berlin, Germany.
{thomas.vogel, grunske}@informatik.hu-berlin.de

important for the performance of evolutionary algorithms, while the performance is analyzed by the evolvability. To perform the fitness landscape analysis, we implemented 11 metrics from the literature to characterize the search space of SAPIENZ regarding diversity and evolvability throughout the search process. Executing SAPIENZ with these metrics on five selected apps, we obtained data that we analyzed regarding diversity and evolvability.

Results. Our analsis indicated that the diversity of the evolved test suites decreases during the first 25 generations of search to a low level. At the same time, SAPIENZ loses its ability to produce better test suites (evolvability)—the search stagnates after 25 generations. Given these results, we adapted the heurisitc of SAPIENZ to preserve the diversity of the test suites during the search by four techniques: initializing the search with diverse test suites, dynamically controlling the diversity during search, eliminating duplicate test suites, and incorporating diversity into the selection. We evaluate the resulting SAPIENZ^{div} in a head-to-head comparison with SAPIENZ on 34 apps. SAPIENZ^{div} significantly outperformed SAPIENZ for coverage on 9/34 and for fault revelation on 20/34 apps while performing similarly on the remaining apps and tending to produce longer test cases than SAPIENZ.

Conclusion. Our work has shown that the understanding of the search problem obtained by the fitness landscape analysis helped us to find a more suitable configuration of SAPIENZ without the need of trial-and-error experiments.

Data Availability. SAPIENZ^{div} is available on GitHub: `https://github.com/thomas-vogel/sapienzdiv-ssbse19`

Bibliography

[AF13] Arcuri, Andrea; Fraser, Gordon: Parameter tuning or default values? An empirical investigation in search-based software engineering. Empirical Software Engineering, 18(3):594–623, 2013.

[AMG17] Aleti, Aldeida; Moser, I.; Grunske, Lars: Analysing the Fitness Landscape of Search-Based Software Testing Problems. Automated Software Engg., 24(3):603–621, 2017.

[ME13] Malan, Katherine M.; Engelbrecht, Andries P.: A survey of techniques for characterising fitness landscapes and some possible ways forward. Information Sciences, 241(Supplement C):148–163, 2013.

[MHJ16] Mao, Ke; Harman, Mark; Jia, Yue: Sapienz: Multi-objective Automated Testing for Android Applications. In: ISSTA'16. ACM, pp. 94–105, 2016.

[PA12] Pitzer, Erik; Affenzeller, Michael: A Comprehensive Survey on Fitness Landscape Analysis. In: Recent Advances in Intelligent Engineering Systems. Springer, pp. 161–191, 2012.

[VTG19] Vogel, Thomas; Tran, Chinh; Grunske, Lars: Does Diversity Improve the Test Suite Generation for Mobile Applications? In: SSBSE '19. Springer, pp. 58–74, 2019.

[VTG21] Vogel, Thomas; Tran, Chinh; Grunske, Lars: A comprehensive empirical evaluation of generating test suites for mobile applications with diversity. Information and Software Technology, 130:106436, 2021.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 123

Identifizierung von Vertraulichkeitsproblemen mithilfe von Angriffsausbreitung auf Architektur[1]

Maximilian Walter[2], Robert Heinrich[3], Ralf Reussner[4]

Keywords: Angreiferpropagation, Software-Architektur, Sicherheit

1 Einleitung und Übersicht

Wir sind dabei immer mehr verschiedene Bereiche unseres täglichen Lebens zu digitalisieren. Diese Systeme haben gemeinsam, dass sie häufig dynamische Zugangskontrollsysteme zusammen mit einer Vielzahl verschiedener verbundener Elemente, wie Komponenten oder Geräte, verwenden. Jedoch sind diese Systeme häufig verwundbar. Angreifer können die einzelnen verwundbaren Elemente zusammen mit legitimen Zugriffsberechtigungen nutzen, um einen verketten Angriffspfad durch die Architektur zu bilden. Daher ist es sinnvoll, diese Abhängigkeit zwischen Zugangskontrollrichtlinien und Schwachstellen zu analysieren. Zwar gibt es bereits Ansätze für die automatische Generierung von Angriffspfaden auf der Grundlage von Schwachstellen und Zugriffskontrolle, wie z.B. Bloodhound[5] bei Active Directory, doch sind Ansätze sehr auf einen Anwendungsbereich bezogen oder setzen häufig ein lauffähiges System voraus und können nicht während der Entwurfszeit oder der Wartung des Systems verwendet werden. Daher haben wir eine architekturgetriebene Angreiferpropagationsanalyse [Wa22] entwickelt, welche mögliche Ausbreitungen von Angreifern berechnen kann. Dabei haben wir die existierende Architekturbeschreibungssprache das Palladio-Komponentenmodell (PCM) [Re16] mit einer Zugriffskontrollmodellierung und einer Verwundbarkeitsmodellierung erweitert. Dies wird in unserer Analyse genutzt, um von einem möglichen Startpunk alle erreichbaren Architekturelemente zu identifizieren. Wir haben den Ansatz anhand verschiedener Fallstudien evaluiert und erhielten eine hohe Genauigkeit.

[1] Diese Arbeit wurde duch die DFG mit der Projektnummer 432576552, HE8596/1-1 (FluidTrust) und dem Forschungsbereich Engineering Secure Systems (46.23.03) der Helmholtz Gemeinschaft (HGF) durch KASTEL Security Research Labs unterstützt.

[2] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, maximilian.walter@kit.edu
[3] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, robert.heinrich@kit.edu
[4] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, ralf.reussner@kit.edu

[5] `https://bloodhoundenterprise.io/`

2 Modellierung und Analyse

Die Verwundbarkeitsmodellierung passiert auf gängige Standards wie z.B. CVSS[6] und die Zugriffskontrollmodellierung folgt dem Konzept der Attributbasierten-Zugriffskontrolle [Hu15]. Wir haben diese Konzepte in das PCM übertragen. Dies ermöglicht Software-Architekt:innen Verwundbarkeiten und Zugriffsrichtlinien direkt in der Software-Architektur zu spezifizieren. Die Wiederverwendung von bereits gängigen Konzepten unterstützt auch die Möglichkeit existierendes Wissen über die Klassifizierung von Schwachstellen und die Modellierung von Sicherheitseigenschaften wiederzuverwenden.

Die Analyse basiert auf dem KAMP-Ansatz [Ro]. Dieser wurde um neue Ausbreitungsregeln für Angreifer und Änderungen erweitert. Die Analyse berechnet iterativ von einem Startpunkt alle Nachbarelemente und überprüft, ob sie diese kompromittieren kann. Dies ist möglich, wenn es entweder eine Schwachstelle gibt, die die Analyse ausnutzen kann oder die Analyse die passende Zugriffsberechtigung hat. Dabei prüft die Analyse für jede Schwachstelle, ob sie die passenden Fähigkeiten hat. In jedem Iterationsschritt kann zudem die Analyse neue Zugriffsberechtigungen durch das Ausnutzen von Schwachstellen oder gespeicherten Passwörtern sammeln. Am Ende wird eine Liste mit allen potenziell betroffen Elementen zurückgegeben. Diese können Architekt:innen nutzen, um Stellen für Schutzmaßnahmen in die Software-Architektur zu identifizieren, damit diese Angriffspfade brechen können.

3 Data Availability

Wir stellen die Evaluationsdaten öffentlich zur Verfügung [Wa]. Dies umfasst die Modelle, die erwarteten Modelle sowie den Quellcode der Analyse inklusive einer virtuellen Maschine zum Ausführen der Analyse.

Literatur

[Hu15] Hu, V. et al.: Attribute-Based Access Control. Computer 48/2, S. 85–88, Feb. 2015, ISSN: 0018-9162.

[Re16] Reussner, R. et al.: Modeling and Simulating Software Architectures – The Palladio Approach. MIT Press, Cambridge, MA, 2016.

[Ro] Rostami, K. et al.: Architecture-Based Change Impact Analysis in Information Systems and Business Processes. In: ICSA'17. S. 179–188.

[Wa] Walter, M. et al.: Dataset - Architectural Attack Propagation Analysis for Identifying Confidentiality Issues, URL: https://doi.org/10.5445/IR/1000141655.

[Wa22] Walter, M. et al.: Architectural Attack Propagation Analysis for Identifying Confidentiality Issues. In: ICSA'22. IEEE, S. 1–12, 2022.

[6] https://www.first.org/cvss/

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 125

VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python – Summary

Laura Wartschinski[1] Yannic Noller[1] Thomas Vogel[1,2] Timo Kehrer[1,3] Lars Grunske[1]

Abstract: In this extended abstract, we summarize our work on VUDENC published in the journal Information and Software Technology (IST) in 2022 [Wa22]. VUDENC uses deep learning to learn features of vulnerable code from a real-world Python codebase and a network of long-short-term memory cells (LSTM) is then used to detect vulnerabilities in code at a fine-grained level.

Keywords: Vulnerability detection; Python; Deep learning

Context. Developing secure software system is important to mitigate vulnerabilities that otherwise could be exploited and have severe consequences such as loss, disclosure, or manipulation of data, or system failures. To avoid or at least reduce code flaws causing vulnerabilities, constructive engineering approaches can be used but need to be complemented by analyatical techniques to detect vulnerabilities. However, a manual detection of vulnerabilities requires expert knowledge and is time-consuming, and must therefore be supported by automated techniques.

Objective. We aim for an automated vulnerability detection technique that should achieve a high accuracy, point developers directly to vulnerable code fragments, scale to real-world software, generalize across the boundaries of a specific software project, and require no or only a moderate manual effort for the setup or configuration. We decided to focus on Python software, which has not been addressed yet by vulnerability detection with deep learning.

Method. To achieve these objectives, we developed VUDENC (Vulnerability Detection with Deep Learning on a Natural Codebase). It leverages deep learning to detect vulnerabilities in Python software. For this purpose, it automatically learns features of vulnerable code directly from a large and real-world Python codebase that we mined from GitHub and comprises (before filtering) 25,040 vulnerability-fixing commits in 14,686 different repositories. This dataset was labeled automatically according to the commit context and covers seven vulnerability types: SQL injection, Cross-site scripting (XSS), Command injection, Cross-site request forgery (XSRF), Remote Code Execution, Path disclosure, and Open Redirect. Accordingly, VUDENC can learn features for these types and detect vulnerabilities of these types. Using this dataset for training, VUDENC applies a word2vec model to identify semantically similar code tokens and provide a vector representation for deep learning.

[1] Humboldt-Universität zu Berlin, Department of Computer Science, Unter den Linden 6, 10099 Berlin, Germany.
{wartschinski, noller, thomas.vogel, kehrer, grunske}@informatik.hu-berlin.de
[2] University of Paderborn, Department of Computer Science, Warburger Straße 100, 33098 Paderborn, Germany.
[3] University of Bern, Department of Computer Science, Hochschulstrasse 6, 3012 Bern, Switzerland.

Particularly, a network of long-short-term memory cells (LSTM) is used to classify vulnerable code token sequences at a fine-grained level, highlight the specific areas in the source code that are likely to contain vulnerabilities, and provide confidence levels for its predictions.

Results. After determining suitable hyperparameters for the word2vec model and the LSTM model, we experimentally evaluated VUDENC on a test dataset of 1,009 vulnerability-fixing commits from different GitHub repositories and with different vulnerability types. VUDENC achieves a precision (i.e., the fraction of true positives in all positive predictions) of 82%-96%, indicating a very low false positive rate. The achieved recall (i.e., the rate of positives that were correctly identified in comparison to the total number of actual positives) of 78%–87% means that just 13%-22% of the samples labeled as vulnerable were missed. The overall F1 score (i.e., the harmonic mean of precision and recall) ranges from 80%-90%, which we see as a very satisfying result especially when considering related work and their effectiveness.

Conclusions. Our experimental results suggest that VUDENC is capable of outperforming most of its competitors in terms of vulnerably detection capabilities on real-world software. A comparable accuracy was only achieved on synthetic benchmarks, within single projects, or on a much coarser level of granularity such as entire source code files. In contrast, VUDENC uses a real-word dataset across multiple projects and detects vulnerabilities at the level of code fragments.

Data Availability. We provide a replication package covering the implementation and documentation of VUDENC on GitHub[4] and the following datasets on Zenode: the plain and embedded datasets for the seven types of vulnerabilities[5], the dataset of commits including their diff files mined from Github[6], and the Python corpus for training the word2vec model as well as one trained model[7].

Bibliography

[Wa22] Wartschinski, Laura; Noller, Yannic; Vogel, Thomas; Kehrer, Timo; Grunske, Lars: VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. Information and Software Technology, 144:106809, 2022.

[4] https://github.com/LauraWartschinski/VulnerabilityDetection

[5] https://doi.org/10.5281/zenodo.3559841

[6] https://doi.org/10.5281/zenodo.3559203

[7] https://doi.org/10.5281/zenodo.3559480

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 127

Preventing technical debt with the TAP framework for Technical Debt Aware Management

Marion Wiese[1] Paula Rachow[2] Matthias Riebisch[3] Julian Schwarze[4]

Abstract:

The talk is based on an article published in the Information and Software Technology journal in 2022 [Wi22].

Technical Debt (TD) is a metaphor for technical problems that are not visible to users and customers but hinder developers in their work, making future changes more difficult. TD is often incurred due to tight project deadlines. Furthermore, project management usually focuses on customer benefits and pays less attention to their IT systems' internal quality. We present a framework focusing on TD prevention and repayment in agile-managed projects. The framework was developed and applied in an IT unit of a publishing house. It includes a feasible method for TD prevention despite tight timelines by making TD repayment part of project management. We evaluated this framework in a comparative case study based on ticket statistics and two structured surveys targeting team members and IT managers. In the observed IT unit, the TAP framework raises awareness for the incurrence of TD. Decisions to incur TD are intentional, and TD is repaid timelier. Unintentional TD incurred by unconscious decisions is prevented. Furthermore, better communication and better planning of the project pipeline can be observed.

Keywords: Technical Debt; Project Management; Technical Debt Awareness; Technical Debt Repayment; Technical Debt Prevention; Technical Debt Backlog

1 Summary

Technical Debt (TD) is "*a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible*" [Av16]. In a technical metaphor for financial debt, a sub-optimal implementation or design is interpreted as debt. The initially named cause for TD incurrence is tight project deadlines, as described in the paper of Cunningham [Cu92]. TD is a serious problem in practice, often caused by tight deadlines due to fast-changing requirements in the age of digitalization.

TD prevention is stated as the preferable option for TD management by many practitioners. In frequent situations, however, a sub-optimal solution that can be implemented more

[1] Universität Hamburg, FB Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, marion.wiese@uni-hamburg.de
[2] Universität Hamburg, FB Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, paula.rachow@uni-hamburg.de
[3] Universität Hamburg, FB Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, matthias.riebisch@uni-hamburg.de
[4] Gruner+Jahr GmbH, Media Sales Services, Baumwall 11, 20459 Hamburg, schwarze.julian@guj.de

quickly must be chosen. To avoid unnecessary TD, a developing team should consciously and intentionally choose the optimal or sub-optimal solution.

In our paper, we present and evaluate the TAP framework for **T**echnical debt **A**ware **P**roject management, which was developed by an IT unit of a German publishing company. The TAP framework divides all tasks that are not visible to the customer into four categories: maintenance tasks, maintenance projects, deconstruction, and technical debt. The developers create corresponding backlog tickets. Particularly the TD tickets are a novel approach. TD tickets comprise only intentional TD, the incurrence of which is discussed and consciously accepted during an estimation meeting. TD tickets are associated with the project they were incurred in and must be repaid as part of the project after meeting the set deadline. This makes the project managers and their team responsible for the TD accumulated during their project, which should decrease their willingness to incur TD.

To evaluate the success of this approach, we analyzed ticket statistics and conducted two surveys addressing team members and IT managers, respectively. To enhance the validity of the results, the surveys were filled out by the observed team and a comparison team that did not use the framework. The research questions targeted the practicality of the framework's application and the TAP framework's perceived effects, benefits, and drawbacks. The evaluation shows that (1) the communication between different stakeholders is optimized, (2) the overall awareness for TD is raised, (3) the decisions on TD incurrence are made consciously and intentionally, (4) TD incurred by unintentional and unconscious decisions can be prevented, and (5) TD incurred by intentional and conscious decisions due to tight deadlines are repaid timely. Finally, the survey of IT managers points to optimized project pipeline planning and improved customer communication and understanding.

2 Data Availability

The additional material is publicly available at Zenodo (`https://doi.org/10.5281/zenodo.5788222`) and includes the survey results, a list of analyzed tickets, calculation details, and a case study protocol.

References

[Av16] Avgeriou, P.; Kruchten, P.; Ozkaya, I.; Seaman, C.: Managing Technical Debt in Software Engineering. Dagstuhl Reports 6/4, pp. 110–138, 2016, ISSN: 01635948.

[Cu92] Cunningham, W.: The WyCash portfolio management system. In: Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA. Vol. 2, pp. 29–30, 1992, ISBN: 0897916107.

[Wi22] Wiese, M.; Rachow, P.; Riebisch, M.; Schwarze, J.: Preventing technical debt with the TAP framework for Technical Debt Aware Management. Information and Software Technology 148/106926, 2022, ISSN: 0950-5849, URL: `https://doi.org/10.1016/j.infsof.2022.106926`.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 129

Variational Satisfiability Solving: Efficiently Solving Lots of Related SAT Problems – Summary

Jeffrey M. Young[1] Paul Maximilian Bittner[2] Eric Walkingshaw[3] Thomas Thüm[4]

Abstract: We report about recent research on satisfiability solving for variational domains, originally published in 2022 in the Empirical Software Engineering Journal (EMSE) within the special issue on configurable systems [Yo22]. Incremental SAT solving is an extension of classic SAT solving that enables solving a set of related SAT problems by identifying and exploiting shared terms. However, using incremental solvers effectively is hard since performance is sensitive to the input order of subterms and results must be tracked manually. This paper translates the ordering problem to an encoding problem and automates the use of incremental solving. We introduce *variational* SAT solving, which differs from incremental solving by accepting all related problems as a single variational input and returning all results as a single variational output. Variational SAT solving automates the interaction with the incremental solver and enables a method to automatically optimize sharing in the input. We formalize a variational SAT algorithm, construct a prototype variational solver, and perform an empirical analysis on two real-world datasets that applied incremental solvers to software evolution scenarios. We show that the prototype solver scales better for these problems than four off-the-shelf incremental solvers while also automatically tracking individual results.

Keywords: satisfiability solving, variation, choice calculus, software product lines

Summary

Satisfiability (SAT) solving is a ubiquitous technology in software product lines for a diverse set of analyses ranging from anomaly detection, dead code analysis, sampling, and automated analysis of feature models. The general pattern is to represent parts of the system or feature model as a propositional formula, and reduce the analysis to a SAT problem. However, modern software is constantly evolving and thus the translation step to a single SAT problem quickly becomes a translation to a set of SAT problems.

Incremental SAT solvers allows the user to hand-write a program to optimize for sharing of equal subterms in a set of related SAT problems to solve. Theoretically, this is more efficient because it reuses knowledge of shared terms. However, using an incremental solver in this way requires substantial manual effort and domain knowledge, produces a specific solution to a specific analysis, and it requires extra infrastructure to manage results.

[1] IOHK, Longmont, Colorado, jeffrey.young@iohk.io
[2] University of Ulm, Germany, paul.bittner@uni-ulm.de
[3] Unaffiliated, Corvallis, OR, USA, ewalkingshaw@acm.org
[4] University of Ulm, Germany, thomas.thuem@uni-ulm.de

In this paper, we show that the performance benefit of using an incremental solver to solve a large number of related SAT problems can be achieved while mitigating the main drawbacks of incremental solvers, that is, high manual effort and deep integration with the application. Our solution is to formalize a method of *variational* SAT solving that makes use of known commonalities among propositional formulas and automates the interaction with an incremental solver. Variational SAT solving takes as input a single *variational formula*, which encodes a set of related SAT problems to solve. It then compiles this variational formula into an efficient program to run on an incremental solver. Finally, it collects the results from the incremental solver into a single result that captures the solutions to all of the SAT problems described by the input variational formula.

Our approach has several benefits: (1) End-users are only required to provide a single variational formula, which represents a set of related propositional formulas, rather than a formula *and* a hand-written program to direct the solver. (2) It is general; while variational satisfiability solving is applied to feature model analyses in this work, it can be used for any analysis that can be encoded as a variational formula, which can be constructed from any set of related SAT problems. (3) With a variational formula, new kinds of syntactic manipulations and optimizations, such as factoring out shared terms, become possible and can be automated. (4) A *variational model* may be produced that encapsulates a set of satisfying assignments for all variants of the variational formula, alleviating the need to track the incremental solver's results when satisfying assigments are needed.

Data Availability

The original publication is accepted for submission and our camera-ready copy is currently processed by the publisher. A preprint is available online at `https://github.com/SoftVarE-Group/Papers/raw/main/2022/2022-EMSE-Young.pdf`. Our artifact is available on Github (`https://github.com/doyougnu/VSat/tree/EMSE-revision-1`) and Zenodo (DOI: 10.5281/zenodo.5543884). Our evaluation data is available on Zenodo (DOI: 0.5281/zenodo.5546009).

Bibliography

[Yo22] Young, Jeffrey M.; Bittner, Paul Maximilian; Walkingshaw, Eric; Thüm, Thomas: Variational Satisfiability Solving: Efficiently Solving Lots of Related SAT Problems. Empirical Software Engineering (EMSE), 28, November 2022.

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 131

DeepHyperion: Exploring the Feature Space of Deep Learning-based Systems through Illumination Search

Tahereh Zohdinasab[1] Vincenzo Riccio[2] Alessio Gambi[3] Paolo Tonella[4]

Abstract: In this extended abstract, we summarize our contributions to automated testing of Deep Learning-based systems published at the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA) in 2021 [Zo21a] and *just accepted* by the ACM Transactions on Software Engineering and Methodology (TOSEM) in 2022 [Zo22].

Deep Learning-based systems (DL Systems) find applications in safety-critical application domains and thus must be thoroughly tested. Existing DL system testing approaches can generate complex and fault-finding inputs but do not characterize them in a way that enables human interpretation and do not always consider test diversity. Our work addresses these challenges and can find effective and diverse test cases.

Keywords: Software testing; deep learning; search-based software engineering; self-driving cars

1 Methodology

Automatically and effectively testing DL systems, like self-driving cars, requires efficiently generating complex inputs, such as driving scenarios, and a strategy to balance exploration to discover untested behaviors and exploitation to identify highly effective test cases. Additionally, testing should produce results that developers can easily interpret and use to improve their design, for instance, by identifying features that are under-represented in training sets. We address the above challenges through DeepHyperion that leverages (1) model-based input representation for generating complex inputs, (2) Illumination Search to generate effective test cases while exploring the test space at large, and (3) metrics to quantify test input as well as DL systems under test's behavior features.

We proposed a two-step methodology that developers can follow to identify representative, discriminative, and quantifiable features that characterize the tests in an intuitive way. Our methodology consists of (i) Open Coding for selecting the independent variables describing the tests and (ii) Metric Identification for devising automatic procedures to quantify them. Having a way to quantify relevant test features, DeepHyperion implements the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites), the Illumination Search algorithm

[1] Università della Svizzera Italiana, Switzerland, tahereh.zohdinasab@usi.ch
[2] Università della Svizzera Italiana, Switzerland, vincenzo.riccio@usi.ch
[3] IMC University of Applied Sciences Krems, Austria, alessio.gambi@fh-krems.ac.at
[4] Universita della Svizzera Italiana, Switzerland, paolo.tonella@usi.ch

proposed by Mouret and Clune [MC15], to mutate a population of tests, called *seeds*. DeepHyperion aims to generate tests that maximize the likelihood of observing misbehaviors for each feature combination and produces N-dimensional maps that visualize the distribution of the fittest tests across the feature space. Those maps, in turn, enable developers to identify clusters of fault-revealing tests with similar features. Since DeepHyperion selects the individuals to mutate randomly, which is unbiased but ineffective, we extended it with a ranked selection scheme based on Contribution Score (CS), a novel metric that promotes individuals that contributed more to exploring the feature space.

2 Results

We evaluated DeepHyperion's effectiveness and efficiency in two application domains: recognition of hand-written digits, which is a classification problem, and steering angle prediction for self-driving cars in driving simulations, which is a regression problem. Our empirical results show that DeepHyperion over-performed existing approaches in both application domains by exposing more unique misbehaviors and exploring larger portions of the feature space. Additionally, our experiments confirmed that selecting individuals according to their Contribution Scores significantly improves DeepHyperion's efficiency. Regarding the usefulness of the N-dimensional feature maps generated by DeepHyperion, our evaluation shows how those maps help DL system developers in identifying shortcomings of training datasets and providing new data to expand them.

3 Data Availability

Our replication package [Zo21b] includes the original code we used for running the experiments and the data we collected during the evaluation of DeepHyperion. The latest version of the code, instead, is available on GitHub at:

`https://github.com/testingautomated-usi/DeepHyperion`

Bibliography

[MC15] Mouret, Jean-Baptiste; Clune, Jeff: Illuminating search spaces by mapping elites. CoRR, abs/1504.04909, 2015.

[Zo21a] Zohdinasab, Tahereh; Riccio, Vincenzo; Gambi, Alessio; Tonella, Paolo: DeepHyperion: exploring the feature space of deep learning-based systems through illumination search. In: Proc. of the Intl. Symposium on Software Testing and Analysis. ACM, pp. 79–90, 2021.

[Zo21b] Zohdinasab, Tahereh; Riccio, Vincenzo; Gambi, Alessio; Tonella, Paolo: , DeepHyperion Replication Package. URL: `https://doi.org/10.5281/zenodo.4742119`, May 2021.

[Zo22] Zohdinasab, Tahereh; Riccio, Vincenzo; Gambi, Alessio; Tonella, Paolo: Efficient and Effective Feature Space Exploration for Testing Deep Learning Systems. ACM Trans. Softw. Eng. Methodol., jun 2022. Just Accepted.

Workshops

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023, 135

Anforderungsmanagement in Enterprise Systems-Projekten

Christoph Weiss[1], Johannes Keckeis[2]

1 Introduction

Viele Enterprise Systems-Auswahl-, Einführungs- und Weiterentwicklungsprojekte scheitern aufgrund fehlender, falscher, mangelhafter bzw. lückenhafter Anforderungen. Dies darum, weil es in diesen Projekten häufig falsche Erwartungshaltungen, Definitions- und Meinungsverschiedenheiten zum Anforderungsmanagement zwischen Kunden und Lieferanten gibt.

Neben den von Enterprise Systems getriebenen Anforderungen sehen sich Unternehmen bei Enterprise Systems-Projekten oft mit zusätzlichen organisatorischen Anforderungen und Herausforderungen konfrontiert wie bspw.:

- Neue oder geänderte Geschäftsprozesse
- Neue oder geänderte Unternehmensorganisation
- Bedarf, Kapazität und Verfügbarkeit der relevanten Projekt-Personals wie bspw. Key-User
- ERP- und Prozesskompetenz der Mitarbeiter
- ERP Fähigkeit der Organisation und des Personals
- Finanzierung und Budgetierung – Operationalisierung von agilen Methoden

Diese Herausforderungen sollen bei diesem Workshop aufgezeigt, besprochen und diskutiert werden.

[1] SIS Consulting GmbH, Innsbruck, Austria, Christoph.Weiss@sis-consulting.com
[2] SIS Consulting GmbH, Innsbruck, Austria, Johannes.Keckeis@sis-consulting.com

20th Workshop on Automotive Software Engineering (ASE'23)

Stefan Kugele[1], Lars Grunske[2]

Abstract: Software-based systems play an increasingly important role and enable most innovations in modern cars. This workshop will address various topics related to automotive software development. The participants will discuss appropriate methods, techniques, and tools needed to address the most current challenges for researchers and practitioners.

Keywords: Automotive; Software Engineering; Workshop

The 20th Workshop on Automotive Software Engineering (ASE'23) addresses the challenges of automotive software development and suitable methods, techniques, and tools for this area. With the increasing number of connected vehicles, modern driver assistance systems and the challenges of fully automated driving, automotive software is more critical today than ever. Furthermore, the distraction-free and intuitive operation of vehicle applications via multimodal interfaces play an increasingly important role. Again, innovative technologies such as voice control, cloud computing or 5G connectivity have found their way into the car. These technological advances have changed the driving experience: Soon, the most popular communication and social media services will be integrated into the vehicle and can then be operated by users while driving.

The workshop's primary goal is to exchange and discuss how current challenges in automotive software engineering can be mastered. The thematic focus offers many cross-references to the Software Engineering (SE) conference, to which the workshop is co-located. The workshop is aimed at researchers, developers, and users from the automotive industry, as well as scientists from research institutes and universities who deal with automotive software engineering. Traditionally, the focus is less on theory and more on applied research. To ensure that only high-quality papers are selected for publication and presentation, at least three reviewers were selected for each of the papers submitted to this year's workshop. Many thanks to all the reviewers who contributed with outstanding commitment to the review process.

[1] Technische Hochschule Ingolstadt, AImotion Bavaria, Esplanade 10, 85049 Ingolstadt, stefan.kugele@thi.de
[2] Humboldt-Universität zu Berlin, Institut für Informatik, Rudower Chaussee 25, 12489 Berlin, grunske@informatik.hu-berlin.de

Programme Committee

Dr. Christian Allmann	Audi AG
Prof. Dr. Marcel Baunach	Technische Universität Graz
Dr. Klaus Becker	Viessmann Elektronik GmbH
Prof. Dr. Lenz Belzner	Technische Hochschule Ingolstadt
Dr. Mirko Conrad	samoconsult GmbH
Dr. Heiko Dörr	Method Park
Dr. Kerstin Hartig	Expleo Germany GmbH
Prof. Dr. Steffen Helke	Fachhochschule Südwestfalen
Prof. Dr. Paula Herber	Universität Münster
Prof. Dr. Thomas Kropf	Robert Bosch GmbH
Apl. Prof. Dr. Wolfgang Müller	Uni Paderborn
Dr. Thomas Noack	Datendeuter GmbH
Prof. Dr. Ralf Reißing	Hochschule Coburg
Prof. Dr. Eric Sax	Karlsruher Institut für Technologie / FZI
Prof. Dr. Jörn Schneider	Hochschule Trier
Prof. Dr. Ramin Tavakoli	Technische Hochschule Nürnberg
Dr. Thomas Vogel	Humboldt-Universität zu Berlin
Prof. Dr. Andreas Vogelsang	Universität Köln

Organization

Prof. Dr. Stefan Kugele	Technische Hochschule Ingolstadt
Prof. Dr. Lars Grunske	Humboldt-Universität zu Berlin

For many years, this workshop has been organized by the GI interest group (Fachgruppe) on "Automotive Software Engineering".[3] The steering committee was consequently involved in the organization of this workshop as well.

[3] https://fg-ase.gi.de

Gregor Engels, Regina Hebig, Matthias Tichy (Hrsg.): SE 2023,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 139

5th Workshop on Avionics Systems and Software Engineering (AvioSE'23)

Bjoern Annighoefer[1], Andreas Schweiger[2], Stéphane Poulaine[3]

Abstract: Systems and software engineering in aerospace is subject to special challenges. For their resolution the AvioSE'23 workshop connects academia and industry with selected scientific presentations of high quality, motivating keynote talks, and an interactive panel discussion.

Keywords: avionics; systems engineering; software engineering; formal method; model-based; requirement; qualification; certification; simulation; process; tool; platform; architecture; AI

1 Scope and History

Considerable advances for aerospace applications are expected with the introduction of new technologies. However, aerospace requirements do not allow the application of these straight away due to regulations and certification. Technologies and methods need to be amended or extended for meeting these. The resulting challenges are addressed in the workshop.

The AvioSE'19[4] edition dealt with general issues and AvioSE'20[5] addressed development tools. AvioSE'21[6] tackled topics for the deployment of AI to avionics. AvioSE'22[7] handled safe and secure avionics architectures (e.g. Integrated Modular Avionics, platforms, multi-core, networks, clouds, middleware).

[1] University of Stuttgart, Institute of Aircraft Systems (ILS), Germany, bjoern.annighoefer@ils.uni-stuttgart.de.
[2] Airbus Defence and Space GmbH, Manching, Germany, andreas.schweiger@airbus.com.
[3] Airbus Operations GmbH, Hamburg, Germany, stephane.poulaine@airbus.com.
[4] Annighoefer et al., 1st Workshop on Avionics Systems and Software Engineering (AvioSE'19), 2019. Annighoefer et al.; Challenges and Ways Forward for Avionics Platforms and their Development in 2019, in IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), 2019.
[5] Annighoefer et al., 2nd Workshop on Avionics Systems and Software Engineering (AvioSE'20).
[6] Annighoefer et al., 3rd Workshop on Avionics Systems and Software Engineering (AvioSE'21); A. Schweiger et al., Classification for Avionics Capabilities Enabled by Artificial Intelligence, IEEE/AIAA 40th Digital Avionics Systems Conference (DASC), 2021.
[7] Annighoefer et al., 4th Workshop on Avionics Systems and Software Engineering (AvioSE'22); B. Annighoefer et al., Domain-specific Drivers and Limits for Avionics Architectures — A Critical Review in the Context of the Avionics Application Domains, IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), 2022.

2 Workshop Objectives

The workshop accelerates the bidirectional transfer of knowledge between academia and industry. It provides a platform for researchers to present new methods, tools, and technologies from avionics systems and software engineering, e.g. model-based development, model-based methods, requirements engineering, formal methods, and virtual methods. These contributions are presented in a scientific format, but the small character of the workshop allows in-depth discussions. This in turn increases the precision and future adjustment of the works. Thus, the workshop provides the enabling platform for the stakeholders to discuss technical, but also process, and educational topics. Further, the forum offers the forming of research consortia, once specific issues have been identified, for which project partners share their research competence.

AvioSE'23 motivates researchers through keynote talks by three invited speakers. The keynotes highlight a dedicated topic, summarize its state-of-the-art, and emphasize urgent challenges.

A current topic is selected and addressed interactively by inviting all participants to discuss aspects and needs of modern avionics. We are connecting academics and professionals in a panel discussion with invited experts from academia, industry, and authorities. The expected outcome is the identification of current and future challenges as well as ideas on how to address these. The panel members' statements can be challenged by the audience. Major conclusions of the panel discussion are made available on a virtual platform.

Acknowledgements

Many people contributed to the success of this workshop. First of all, we want to give thanks to the authors of the accepted papers. Second, high appreciation goes to our keynote speakers. Third, sincere thanks are directed to the panelists for sharing their knowledge and their willingness to answer the questions. Fourth, we want to express our gratitude to the SE 2023 organizers for supporting and hosting our workshop. Finally, we are thankful for the contributions of the program committee's members for soliciting papers and writing peer reviews: Björn Annighöfer (University of Stuttgart), Jürgen Becker (KIT), Steffen Becker (University of Stuttgart), Stefan Brunthaler (Universität der Bundeswehr München) Umut Durak (DLR Braunschweig), Rolf Büse (Diehl Aerospace GmbH), Holger Flühr (FH Johanneum Graz), Ralf God (Hamburg University of Technology), Lars Grunske (Humboldt-Universität zu Berlin), Christian Heinzemann (Robert Bosch GmbH), Wolfgang Hommel (Universität der Bundeswehr München), Eric Knauss (University of Gothenburg), Winfried Lohmiller (Airbus Defence and Space GmbH), Christian Meißner (Volkswagen AG), Alexander Pretschner (Munich University of Technology), Stephan Rudolph (Northrop Grumman LITEF GmbH), Bernhard Rumpe (RWTH Aachen University), Andreas Schweiger (Airbus Defence and Space GmbH), Steven VanderLeest (RAPITA Systems), and Sebastian Voss (Aachen University of Applied Sciences).