GESELLSCHAFT
FÜR INFORMATIK

Lars Grunske, Janet Siegmund,
Andreas Vogelsang (Hrsg.)

Software Engineering 2022

Fachtagung des GI-Fachbereichs Softwaretechnik

21. – 25. Februar 2022
Berlin/Virtuell

Gesellschaft für Informatik e.V. (GI)

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

Volume P-320

ISBN 978-3-88579-714-2
ISSN 1617-5468

Volume Editors
Prof. Dr. rer. nat. Lars Grunske
Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany

Prof. Dr.-Ing. Janet Siegmund
Technische Universität Chemnitz
Fakultät für Informatik
Straße der Nationen 62, 09111 Chemnitz, Germany

Prof. Dr. Andreas Vogelsang
Universität zu Köln
Mathematisch-Naturwissenschaftliche Fakultät
Albertus-Magnus-Platz, 50923 Köln, Germany

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Herzlich willkommen zur Tagung Software Engineering 2022 (SE 22) 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 etabliert, und spricht sowohl Softwareentwickler:innen aus der Praxis, als auch an Forscher:innen aus dem akademischen Umfeld an.

Die SE präsentiert im wissenschaftlichen Hauptprogramm ein "Best-Of" der Arbeiten, die in hochrangigen internationalen Fachzeitschriften und Konferenzen von deutschsprachigen Autor:innen veröffentlicht wurden und umfasst eine große Bandbreite an Themen, was sich in einem vielfältigen Programm widerspiegelt.

Darüber hinaus sind alle Daten und Artefakte einer Arbeit 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 39 Einreichungen, von denen letztendlich 33 akzeptiert wurden.

Das wissenschaftliche Hauptprogramm der SE 22 wird durch Workshops ergänzt, in denen weitere Themen im kleineren Kreis intensiv diskutiert werden:

Anforderungsmanagement in Enterprise Systems-Projekten (AESP'22)
Automotive Software Engineering (ASE'22)
Avionics Systems and Software Engineering (AvioSE'22)
Software Engineering in Cyber-Physical Production Systems (SECPPS)

Wir danken allen, die zum Gelingen der Konferenz beigetragen haben, insbesondere den Autor:innen, den Gutachter:innen, den Keynote-Speakern, den Organisator:innen der Workshops und Tracks, den Teilnehmer:innen, den Sponsoren (Porsche Innovations, CQSE, IAV) und der GI e.V., dem Proceedings Chair Hoang Lam Nguyen, dem Publicity Chair Sandro Schulze sowie allen Helfer:innen, die die Durchführung der Konferenz auch im virtuellen Format ermöglicht haben.

Berlin, im Februar 2022

Lars Grunske, Janet Siegmund und Andreas Vogelsang

Sponsoren

Wir danken den folgenden Unternehmen und Institutionen für die Unterstützung der Konferenz.

Gold Sponsoren

IAV GmbH Ingenieurgesellschaft
Auto und Verkehr

Carnotstraße 1
10587 Berlin

Webseite:
www.iav.com

CQSE GmbH

Centa-Hafenbrädl-Straße 59
81249 München

Webseite:
www.cqse.eu

Bronze Sponsoren

Porsche Digital GmbH

Grönerstraße 11/1
71636 Ludwigsburg

Webseite:
www.porsche.digital

Tagungsleitung

Gesamtleitung:	Lars Grunske, Humboldt-Universität zu Berlin
Leitung des Programmkomitees:	Janet Siegmund, Technische Universität Chemnitz Andreas Vogelsang, Universität zu Köln
Industrie:	Peter Munk, Robert Bosch GmbH Stefan Sauer, Universität Paderborn
Workshops:	Judith Michael, RWTH Aachen Andreas Wortmann, Universität Stuttgart
Publicity:	Sandro Schulze, Universität Potsdam
Proceedings:	Hoang Lam Nguyen, Humboldt-Universität zu Berlin
Workshop Proceedings:	Jérôme Pfeiffer, Universität Stuttgart

Programmkomitee

Stefan Biffl	TU Wien
Ruth Breu	Universität Innsbruck
Sabine Glesner	TU Berlin
Martin Glinz	Universität Zürich
Paula Herber	Universität Münster
Marie-Christine Jakobs	TU Darmstadt
Anne Koziolek	Karlsruher Institut für Technologie
Barbara Paech	Universität Heidelberg
Martin Pinzger	Universität Klagenfurt
Rick Rabiser	Johannes Kepler Universität Linz
Ina Schaefer	TU Braunschweig
Sibylle Schupp	TU Hamburg
Christoph Seidl	IT University of Copenhagen
Gabriele Taentzer	Philipps-Universität Marburg
Manuel Wimmer	Johannes Kepler Universität Linz
Uwe Zdun	Universität Wien

rSE22 Track-Komitee

Stephan Janosch (Track Chair)	MPI-CBG
Jan-Phillip Dietrich	PIK Potsdam
Bernadette Fritzsch	AWI Bremerhaven
Eileen Kühn	Karlsruher Institut für Technologie
Anna-Lena Lamprecht	Universität Utrecht
Jan Linxweiler	TU Braunschweig

Inhaltsverzeichnis

Wissenschaftliches Hauptprogramm

Workshops

Wissenschaftliches Hauptprogramm

Orthogonal Uncertainty Model: Documenting Uncertainty in the Engineering of Cyber-Physical Systems

Torsten Bandyszak[1], Marian Daun[2], Bastian Tenbergen[3], Patrick Kuhs[4], Stefanie Wolf[5], Thorsten Weyer[6]

Abstract: In this talk, we present our contribution "Orthogonal Uncertainty Modeling in the Engineering of Cyber-Physical Systems" published in IEEE Transactions on Automation Science and Engineering in July 2020 [Ba20]. We have proposed a modeling language for "Orthogonal Uncertainty Models" (OUMs). OUMs constitute a dedicated, central artifact to document, analyze, understand, and discuss potential uncertainties that may occur during operation. Thereby, OUMs support the systematic consideration of potential uncertainties the system might face during operation. OUMs allow documenting various aspects of uncertainty (e.g., cause and effect) in a dedicated artifact. Trace links are used to relate uncertainty to, e.g., system requirements across different model-based artifacts. We have applied OUMs to an industrial case study from the industry automation domain.

Keywords: Uncertainty; Uncertainty Modeling; Model-Based Engineering; Cyber-Physical Systems

1 Motivation

Cyber-physical systems (CPS) typically operate in highly dynamic and uncertain contexts. Many kinds of uncertainty occur in the operation of CPS, e.g., due to imperfection of sensors. Other concerns, such as the real-time criticality of CPS increases the complexity of uncertainty consideration as well [BWD21]. Uncertainty needs to be handled systematically in the engineering. If uncertainty remains unconsidered, severe safety hazards may occur during operation.

Engineering CPS that are able to safely handle uncertainty requires identifying and documenting such uncertainty. In requirements engineering, potential operational uncertainty must be uncovered, modelled, and discussed with stakeholders. Integrating information about potential uncertainty into other development artifacts (e.g., behavioral models of the system and the context) has two major drawbacks: 1) It causes redundancy of uncertainty information, and 2) related uncertainty information (e.g., cause and effect of an uncertainty) is spread across several models. Uncertainty is in fact a cross-cutting concern, similar to, e.g., variability in software product line engineering.

[1] Universität Duisburg-Essen, paluno – The Ruhr Institute for Software Technology, torsten.bandyszak@uni-due.de
[2] Universität Duisburg-Essen, paluno – The Ruhr Institute for Software Technology, marian.daun@uni-due.de
[3] State University of New York at Oswego, Department of Computer Science, bastian.tenbergen@oswego.edu
[4] Universität Duisburg-Essen, paluno – The Ruhr Institute for Software Technology, patrick.kuhs@stud.uni-due.de
[5] Siemens AG, Technology, wolf.stefanie@siemens.com
[6] Universität Koblenz-Landau, Institute for Software Technology, thorstenweyer@uni-koblenz.de

2 The OUM Modeling Language

OUMs allow engineers to capture uncertainty information in a dedicated, model-based development artifact. OUMs can be connected to any other development artifact via trace links. That way, e.g., the effect of uncertainty on the fulfillment of system requirements can be documented. OUMs are graphical models, which are easy to understand and support the identification, documentation, analysis and communication of uncertainty.

The OUM language is based on common ontological concepts for uncertainty. However, in contrast to existing uncertainty modeling approaches, OUMs are specifically focused on modeling uncertainty from the point of view of a system in operation. Among others, one specific modeling concept is the so-called *Observation point*, which allows specifying how (e.g., based on which inputs) the system will become aware of uncertainty during operation. OUMs also depict the *Rationale*, *Effect*, *Activation Condition*, and *Mitigation* of uncertainty. Furthermore, it is possible to explicitly model relationships between different uncertainties.

We have applied the OUM approach to the case examples of a fleet of transport robots [Ba18], a smart factory [Ba20], and a cooperative ACC system [Ba21]. The results from these case studies show that OUMs support systematically uncovering and investigating uncertainty in several interrelated development artifacts and at different levels of abstraction.

3 Data Availability

The OUM notation has become part of a comprehensive modeling tool suite for CPS development, which is implemented as a Microsoft Visio extension and publicly available [7].

Literaturverzeichnis

[Ba18] Bandyszak, Torsten; Daun, Marian; Tenbergen, Bastian; Weyer, Thorsten: Model-based Documentation of Context Uncertainty for Cyber-Physical Systems. In: 2018 IEEE 14th International Conference on Automation Science and Engineering. S. 1087–1092, 2018.

[Ba20] Bandyszak, Torsten; Daun, Marian; Tenbergen, Bastian; Kuhs, Patrick; Wolf, Stefanie; Weyer, Thorsten: Orthogonal Uncertainty Modeling in the Engineering of Cyber-Physical Systems. IEEE Trans. on Automation Science and Engineering, 17(3):1250–1265, 2020.

[Ba21] Bandyszak, Torsten; Jöckel, Lisa; Kläs, Michael; Törsleff, Sebastian; Weyer, Thorsten; Wirtz, Boris: Handling Uncertainty in Collaborative Embedded Systems Engineering. In (Böhm, Wolfgang; Broy, Manfred; Klein, Cornel; Pohl, Klaus; Rumpe, Bernhard; Schröck, Sebastian, Hrsg.): Model-Based Engineering of Collaborative Embedded Systems: Extensions of the SPES Methodology. Springer, Cham, S. 147–170, 2021.

[BWD21] Bandyszak, Torsten; Weyer, Thorsten; Daun, Marian: Uncertainty Theories for Real-Time Systems. In (Tian, Yu-Chu; Levy, David Charles, Hrsg.): Handbook of Real-Time Computing. Springer, Singapore, S. 1–34, 2021.

[7] See `https://github.com/SSEPaluno/SPES-Modeling-Tool`

Feature Trace Recording – Summary

Paul Maximilian Bittner[1], Alexander Schultheiß[2], Thomas Thüm[3], Timo Kehrer[4], Jeffrey M. Young[5], Lukas Linsbauer[6]

Abstract: In this work, we report about recent research on Feature Trace Recording, originally published at the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) 2021 [Bi21]. Tracing requirements to their implementation is crucial to all stakeholders of a software development process. When managing software variability, requirements are typically expressed in terms of features, a feature being a user-visible characteristic of the software. While feature traces are fully documented in software product lines, ad-hoc branching and forking, known as clone-and-own, is still the dominant way for developing multi-variant software systems in practice. Retroactive migration to product lines suffers from uncertainties and high effort because knowledge of feature traces must be recovered but is scattered across teams or even lost. We propose a semi-automated methodology for recording feature traces proactively, *during* software development when the necessary knowledge is present. To support the ongoing development of previously unmanaged clone-and-own projects, we explicitly deal with the absence of domain knowledge for both existing and new source code. We evaluate feature trace recording by replaying code edit patterns from the history of two real-world product lines. Our results show that feature trace recording reduces the manual effort to specify traces.

Keywords: feature traceability; feature location; disciplined annotations; clone-and-own; software product lines

1 Summary

For comprehending, maintaining, and extending existing software, it is crucial to find locations of interest in a software system quickly, reliably, and exhaustively. To that end, tracing requirements to their implementation is one of the most common activities of developers and crucial to all stakeholders of a software development process. When managing software variability, requirements are typically expressed in terms of *features*. A *feature trace* identifies those artefacts of the software system that implement a certain feature, thus indicating where, how, and which features are implemented.

While features, their dependencies, and their locations are fully documented in software product-line engineering, it is rarely adopted in practice for multiple resons. In practice,

[1] University of Ulm, Germany, paul.bittner@uni-ulm.de
[2] Humboldt-University of Berlin, Germany, alexander.schultheiss@informatik.hu-berlin.de
[3] University of Ulm, Germany, thomas.thuem@uni-ulm.de
[4] Humboldt-University of Berlin, Germany, timo.kehrer@hu-berlin.de
[5] Oregon State University, Corvallis, Oregon, USA, youngjef@oregonstate.edu
[6] TU Braunschweig, Germany, l.linsbauer@tu-braunschweig.de

development often begins with only a single variant of the software system to reduce complexity and costs, or because the need for future variants is unknown. To derive a new variant of the system, developers clone the whole software system to alter specific parts independently from the previous variant (e. g., using branches or forks), also known as clone-and-own. Unfortunately, propagating changes such as bug fixes to other cloned variants is increasingly difficult and ambiguous with a growing number of variants because knowledge of feature traces is scattered across the team or even lost.

We propose *feature trace recording*, a semi-automated methodology to infer feature traces semi-automatically upon source code changes. The main idea is that by recording feature traces during development, they neither have to be recovered retroactively, suffering from uncertainties, nor specified explicitly in a separate step, which causes overheads to the actual source code editing and becomes increasingly difficult as time passes since the last edit. Our core design philosophy is that contributing domain knowledge in the form of feature traces must place minimal burden on the user in order to be accepted. As developers might not always know to which feature an edited artefact belongs, and because feature traces are initially absent in clone-and-own development, we specifically deal with absent domain knowledge on both new and existing source code.

In our envisioned methodology, software development is performed according to a session-oriented editing model where developers *may* specify the feature or feature interaction they are currently implementing. From the edits developers make under such a *feature context*, our recording algorithm infers feature traces for changed source code automatically. Thereby, feature trace recording is not tied to any specific task in the workflow of developers, such as commits to a version control system. Moreover, by employing *disciplined annotations* with Abstract Syntax Trees, we release developers from manual, laborious, and error-prone tasks, such as assigning opening and closing brackets to specify feature traces which are scattered among several code fragments.

2 Data Availability

The original publication is accessible under the DOI 10.1145/3468264.3468531. A preprint is also available online at `https://github.com/SoftVarE-Group/Papers/raw/master/2021/2021-ESECFSE-Bittner.pdf`. Our artifact is available on Github (`https://pmbittner.github.io/FeatureTraceRecording`) and Zenodo (DOI: 10.5281/zenodo.4900682).

Bibliography

[Bi21] Bittner, Paul Maximilian; Schultheiß, Alexander; Thüm, Thomas; Kehrer, Timo; Young, Jeffrey M.; Linsbauer, Lukas: Feature Trace Recording. In: Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE). ACM, New York, NY, USA, pp. 1007–1020, August 2021.

Identifying Challenges for OSS Vulnerability Scanners - A Study & Test Suite (Short Summary)

Andreas Dann[1] Henrik Plate[2] Ben Hermann[3] Serena Elisa Ponta[2] Eric Bodden[4]

Abstract: This short paper[5] presents a study investigating the impact of typical development practices, like re-compilation, re-bundling, on the performance of vulnerability scanners to detect known vulnerabilities in used open-source dependencies. In particular, the paper studies *(i)* types of modifications that affect the detection of vulnerable open-source dependencies and *(ii)* their impact on the performance of vulnerability scanners through an empirical study on 7024 Java projects developed at *SAP*.

Keywords: Security maintenance; Open-Source Software; Security Vulnerabilities

Introduction and Study Design The use of open-source software (OSS) is an established practice, even for industrial applications as much as 75% of the code comes from OSS [Pi16, He11, BHD12, Ba15]. At the same time, more than 67% of the applications include vulnerable OSS with on average 22 individual vulnerabilities, posing a high risk for the applications [Pi16].

To discover known-vulnerable OSS, several vulnerability scanners exist, e.g., the open-source tools OWASP Dependency-Check (OWASP) and Eclipse Steady, the free GitHub Security Alerts, and commercial tools such as Snyk, Black Duck, and WhiteSource. While previous studies [He11, BHD12, Ku18, Pi16, Pa20] investigated to which extent applications include (vulnerable) OSS, they did not investigate the impact of development practices, like forking, patching, re-bundling, on the performance of these scanners.

To fill this gap, we studied *(i)* development practices w.r.t use of OSS and *(ii)* their impact on the scanners' performance through a two-folded case study on 7024 Java projects developed at *SAP*.

First, we investigated typical practices regarding the use of OSS in an industrial context and classified the observed modifications into four different types. In particular, we studied the Bill of Materials (BoM) of 7024 Java project generated by Eclipse Steady [PPS18], and investigated how developers include (modified) OSS.

[1] The author is now with CodeShield GmbH. At the time of writing, he was with the Secure Software Engineering group, Heinz Nixdorf Institute, Paderborn University, Germany. andreas.dann@uni-paderborn.de
[2] The author is with SAP Security Research Mougins, France. <firstname>.<lastname>@sap.com
[3] The author is Professor for Secure Software Engineering, Technical University of Dortmund, Germany ben.hermann@cs.tu-dortmund.de
[4] The author is Professor for Secure Software Engineering at Paderborn University and Director for Software Engineering and IT-Security at Fraunhofer IEM, Paderborn, Germany. eric.bodden@uni-paderborn.de
[5] the full paper is available online: https://doi.org/10.1109/TSE.2021.3101739

Second, we further investigated the prevalence of the identified modifications and their impact on the performance of vulnerability scanners. To assess their prevalence, we checked how often they occur on Maven Central using a sample set of the 20 most-used OSS in our study. To assess the impact of modifications, we compared the performance of the open-source vulnerability scanners OWASP and Eclipse Steady, GitHub Security Alert, and three commercial scanners w.r.t. the modifications. To this end, we computed for each scanner precision, recall, and F1-score on a representative set of 16 CVEs.

Study Results Our study shows that projects do not only include unmodified OSS distributed by the original OSS-project but also include code that has been altered in some way and is re-distributed in the context of other (downstream) projects. In our study, we observed four modification types: re-compilation, re-packaging, metadata-removal (e.g., removed MANIFEST files), Uber-JARs, or combinations of those. Such modifications also commonly occur in an open-source context. On Maven Central, we found cases of Uber-JARs (87%), re-compilation (56%), re-packaging, and metadata-removal (57%).

Further, our study shows that the observed modifications heavily decrease the precision and recall of vulnerability scanners. None of the scanners considered was able to handle all types of modification. One commercial scanner could not identify any vulnerabilities if a modification has been applied to the dependencies. Most scanners relied heavily on metadata, and thus failed to detect vulnerabilities if metadata was removed or the dependency has been re-packaged. Only OWASP and one commercial scanner were able to detect vulnerabilities when multiple dependencies were bundled into a single re-packaged Uber-JAR. The results show that the identified modifications are major challenges for the detection of vulnerable OSS and further research is needed to improve the scanners' detection methods.

***Achilles* - Test Suit** To facilitate a reproducible and comparative assessment of vulnerability scanners, the paper also introduces *Achilles*, a novel test suite to replicate modified OSS, along with 2505 test cases (labeled true- and false-positives) for 723 distinct OSS artifacts.

Conclusion The paper shows that modified OSS, compromising re-compiled, re-packaged, or re-bundled classes from other open-source projects, commonly occur in an industrial and an open-source context. Further, the paper shows that all vulnerability scanners struggle to cope with modified OSS, as none of the scanners was able to handle all observed modification types.

Data Availability The test suite *Achilles*, along with the case study of vulnerability scanners, is made available as an open-source project[6].

[6] `https://github.com/secure-software-engineering/achilles-benchmark-depscanners`

Bibliography

[Ba15] Bavota, Gabriele; Canfora, Gerardo; Di Penta, Massimiliano; Oliveto, Rocco; Panichella, Sebastiano: How the Apache community upgrades dependencies: an evolutionary study. Empirical Software Engineering, 20(5):1275–1317, oct 2015.

[BHD12] Bauer, Veronika; Heinemann, Lars; Deissenboeck, Florian: A Structured Approach to Assess Third-Party Library Usage. In: Proceedings of the 2012 IEEE International Conference on Software Maintenance. ICSM'12, IEEE Computer Society, USA, p. 483–492, 2012.

[He11] Heinemann, Lars; Deissenboeck, Florian; Gleirscher, Mario; Hummel, Benjamin; Irlbeck, Maximilian: On the Extent and Nature of Software Reuse in Open Source Java Projects. In: Proceedings of the 12th International Conference on Top Productivity through Software Reuse. ICSR'11, Springer-Verlag, Berlin, Heidelberg, p. 207–222, 2011.

[Ku18] Kula, Raula Gaikovina; German, Daniel M.; Ouni, Ali; Ishio, Takashi; Inoue, Katsuro: Do developers update their library dependencies? Empirical Software Engineering, 23(1):384–417, feb 2018.

[Pa20] Pashchenko, Ivan; Plate, Henrik; Ponta, Serena Elisa; Sabetta, Antonino; Massacci, Fabio: Vuln4Real: A Methodology for Counting Actually Vulnerable Dependencies. IEEE Transactions on Software Engineering, 2020.

[Pi16] Pittenger, Mike: The State of Open Source Security in Commercial Applications. Technical report, Black Duck Software, 2016.

[PPS18] Ponta, Serena Elisa; Plate, Henrik; Sabetta, Antonino: Beyond Metadata: Code-Centric and Usage-Based Analysis of Known Vulnerabilities in Open-Source Software. In: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018. IEEE Computer Society, pp. 449–460, 2018.

Three Major Instructional Approaches for Requirements Engineering

Marian Daun[1], Alicia M. Grubb[2], Bastian Tenbergen[3]

Abstract: In this talk, we report on our findings from the paper *A Survey of Instructional Approaches in the Requirements Engineering Education Literature* [DGT21], which has been accepted at and published in the proceedings of the *2021 IEEE International Conference on Requirements Engineering*. The paper reports the findings of a systematic literature review to define and investigate the current state of research on requirements engineering education.

Keywords: Requirements Engineering Education; Systematic Literature Review

1 Introduction

Requirements engineering has established as a software engineering discipline and as important part of software engineering curricula. Therefore, an effort has been made to shape this discipline, establish the need to consider requirements engineering a vital part of software engineering studies, and to define the important core of requirements engineering every software engineer needs to understand during their studies (cf. [Gl21]). Due to the diverse and theory-heavy nature of requirements engineering, it poses major challenges for the learner and the instructor alike. To lay a foundation for a common instructional toolkit, we conducted a systematic literature review and investigated the requirements engineering education literature.

2 Findings

We included 152 primary studies from 1988 to 2020. Analysis allowed us to define best practices and current trends to provide guidance for requirements engineering instructors. Results indicate a steady increase in requirements engineering education literature, especially for the last couple of years. Most contributions are solution proposals and evidence is mostly

[1] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, Essen, Germany marian.daun@paluno.uni-due.de
[2] Smith College, Department of Computer Science, Northhampton, MA, USA amgrubb@smith.edu
[3] State University of New York at Oswego, Department of Computer Science, Oswego, NY, USA bastian.tenbergen@oswego.edu

presented in the form of experience reports, indicating overall low maturity of the field. We identified three recurring themes specific to instructional approaches in the requirements engineering education literature. These are:

1. Requirements engineering education approaches often use "project-based learning and team collaboration" to make students ready for solving requirements engineering problems on their own and to enable students working in collaborative manner.
2. Teaching "soft-skill development through real and realistic stakeholder involvement" is an important topic as students need to understand the significance of personal communication to elicit, negotiate, and validate requirements with stakeholders.
3. "Gamification and simluation" are used in requirements engineering education to make teaching materials more interesting and inspiring for students.

Comparing our findings with the idea of Mary Shaw to specialize requirements engineering education already in undergraduate software engineering curricula [Sh00], we conclude that a sizable proportion of the instructional approaches proposed after 2008 that we surveyed were targeted at undergraduate learners, and trained them for the role of "requirements engineer". However, we also found out that Shaw's aspiration for flexible yet robust pedagogical approaches and curricula remains unsatisfied. We only found very few studies dealing with how to instruct requirements consistency, traceability, safety, and security. Moreover, requirements engineering education presently suffers from a lack of a common pedagogical basis. In addition, while a plethora of successful and effective techniques are proposed, more thorough empirical investigations are still needed.

3 Data Availability

The identified publications as well as their mapping and classification are available as data set at doi.org/10.35482/csc.003.2021.

Literaturverzeichnis

[DGT21] Daun, Marian; Grubb, Alicia M.; Tenbergen, Bastian: A Survey of Instructional Approaches in the Requirements Engineering Education Literature. In (Moreira, Ana; Schneider, Kurt, Hrsg.): 29th IEEE International Requirements Engineering Conference, RE 2021, South Bend, USA, September 20-24, 2021. IEEE, 2021.

[Gl21] Glinz, Martin: The Challenge(s) of Teaching Requirements Engineering. In: Requirements Engineering: Foundation for Software Quality - 27th International Working Conference, REFSQ 2021, Essen, Germany, April 12-15, 2021, Keynote. 2021.

[Sh00] Shaw, Mary: Software Engineering Education: A Roadmap. In: Proceedings of the Conf. on The Future of Software Engineering. ICSE '00, Association for Computing Machinery, New York, NY, USA, S. 371–380, 2000.

A Precedence-Driven Approach for Concurrent Model Synchronization Scenarios using Triple Graph Grammars

Lars Fritsche[1], Jens Kosiol[2], Adrian Möller[3], Andy Schürr[4], Gabriele Taentzer[5]

Abstract: We summarize our paper *A Precedence-Driven Approach for Concurrent Model Synchronization Scenarios using Triple Graph Grammars* that has been published in the proceedings of the *13th ACM SIGPLAN International Conference on Software Language Engineering* (SLE 2020).

Keywords: bidirectional transformation; concurrent model synchronization; triple graph grammars

1 Summary

Model-driven engineering has proven to be an effective means to tackle the challenges that accompany the development of increasingly complex modern software systems. Often more than one model is needed to describe the developed system from different but overlapping perspectives. Keeping these models in a consistent state is a challenging task, often called *model synchronization*. Model synchronization becomes especially challenging when correlated models are changed concurrently. In such cases, not all changes can always be propagated between models as some may contradict each other and thus, are in conflict. This is the case, e. g., when a change in one model leads to the deletion of elements in the other whose existence is the prerequisite for changes performed in that second model. For a modern concurrent synchronization approach, it is therefore of paramount importance to identify synchronization conflicts reliably and give modelers the ability to orchestrate model synchronization processes for guiding the process in accordance with their goals.

Some approaches to concurrent synchronization provide solely one hard-wired solution of a conflict-detection and -resolution strategy (from a universe of many different options), others come without any formal guarantees for their synchronization results or have an exponential runtime behavior w.r.t. the size of the processed models. In [Fr20], we develop a framework that simplifies the implementation (orchestration of a family) of concurrent synchronization algorithms with the following properties:

- These algorithms are derived from a declarative rule-based formal specification of a model consistency relation in the form of so-called Triple Graph Grammars (TGGs).

[1] Technical University Darmstadt, Darmstadt, Germany lars.fritsche@es.tu-darmstadt.de
[2] Philipps-Universität Marburg, Marburg, Germany kosiolje@mathematik.uni-marburg.de
[3] Technical University Darmstadt, Darmstadt, Germany adrian.moeller@stud.tu-darmstadt.de
[4] Technical University Darmstadt, Darmstadt, Germany andy.schuerr@es.tu-darmstadt.de
[5] Philipps-Universität Marburg, Marburg, Germany taentzer@mathematik.uni-marburg.de

- They come with a number of predefined but extensible strategies for conflict detection as well as consistency restoration.
- Formal properties can be shown using state-of-the-art category-theory- and graph-transformation-based proof techniques (e. g., [Fr21; He15; OPN20]).
- We provide a reference implementation of our framework based on the state-of-the-art graph transformation tool *eMoflon*. A first evaluation shows that in our approach, the runtime of the synchronization algorithm depends polynomially on the size of processed model changes and not on the size of the whole models. This is achieved by limiting the effects of model updates to causally dependent areas in the regarded models and relying on incremental graph pattern matching techniques.

2 Data Availability

In the ACM Digital Library, we provide an artifact that contains the evaluation of our paper in the form of a virtual machine (https://dl.acm.org/do/10.1145/3410252/full/). An accompanying step-by-step guide explains how to configure the evaluation and how to access our source code.

References

[Fr20] Fritsche, L.; Kosiol, J.; Möller, A.; Schürr, A.; Taentzer, G.: A precedence-driven approach for concurrent model synchronization scenarios using triple graph grammars. In (Lämmel, R.; Tratt, L.; de Lara, J., eds.): Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2020. ACM, pp. 39–55, 2020, URL: https://doi.org/10.1145/3426425.3426931.

[Fr21] Fritsche, L.; Kosiol, J.; Schürr, A.; Taentzer, G.: Avoiding unnecessary information loss: correct and efficient model synchronization based on triple graph grammars. Int. J. Softw. Tools Technol. Transf. 23/3, pp. 335–368, 2021, URL: https://doi.org/10.1007/s10009-020-00588-7.

[He15] Hermann, F.; Ehrig, H.; Orejas, F.; Czarnecki, K.; Diskin, Z.; Xiong, Y.; Gottmann, S.; Engel, T.: Model synchronization based on triple graph grammars: correctness, completeness and invertibility. Softw. Syst. Model. 14/1, pp. 241–269, 2015, URL: https://doi.org/10.1007/s10270-012-0309-1.

[OPN20] Orejas, F.; Pino, E.; Navarro, M.: Incremental Concurrent Model Synchronization using Triple Graph Grammars. In (Wehrheim, H.; Cabot, J., eds.): Fundamental Approaches to Software Engineering – 23rd International Conference, FASE 2020. Vol. 12076. Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 273–293, 2020, URL: https://doi.org/10.1007/978-3-030-45234-6_14.

Lars Grunske, Janet Siegmund, Andreas Vogelsang (Hrsg.): SE 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 29

Identifying Domain-Based Cyclic Dependencies in Microservice APIs Using Source Code Detectors

Patric Genfer[1], Uwe Zdun[2]

Abstract: Isolation, autonomy, and loose coupling are critical success factors of microservice architectures, but unfortunately, systems tend to become strongly coupled over time and sometimes even exhibiting cyclic communication chains. These cycles can even manifest on a conceptual or domain level, making them hard to track for algorithms that rely solely on static analysis. Accordingly, previous attempts to detect cycles either focused on synchronous communication or had to collect additional runtime data, which can be costly and time-consuming. We suggest a novel approach for identifying and evaluating domain-based cyclic dependencies in microservice systems based on modular, reusable source code detectors. Based on the architecture model reconstructed by our detectors, we derived a set of architectural metrics for identifying and classifying domain-based cyclical dependencies. By conducting two case studies on open-source microservice architectures, we validated the feasibility and applicability of our approach.

Keywords: Microservice API; domain-based cyclic dependencies; metrics; source code detectors

1 Introduction

One of the main goals of microservices is to reduce the complexity of large monolithic applications by splitting them up into smaller, autonomously acting services [Th15]. In addition to being isolated from each other, lightweight inter-service communication is central in microservice architectures [Ne15]. These communications create dependencies between services and are often problematic when they form cycles, where a chain of service calls ends in the same service where it began [TL18]. Changing the communication flow from synchronous to asynchronous communication alone does not resolve these cyclic dependencies, but instead only shifts them to a different, more conceptual level [Wo17] where they now become part of the domain or business logic, making them even more difficult to track through static analysis. This work therefore presents a novel approach for identifying and evaluating both, synchronous and domain-based cyclic dependencies on microservice API operation level. For this, we reverse engineer a communication model from underlying microservice code artifacts by using lightweight source code detectors [Nt21] and based on this model, we define a set of architectural metrics for detecting and evaluating potential cyclic dependency structures.

[1] University of Vienna, Faculty of Computer Science, Research Group Software Architecture, Währinger Str. 29, 1090 Wien, Austria patric.genfer@univie.ac.at
[2] University of Vienna, Faculty of Computer Science, Research Group Software Architecture, Währinger Str. 29, 1090 Wien, Austria uwe.zdun@univie.ac.at

2 Architecture Reconstruction and Cycle Identification

To model the communication flow observable at the microservice API level, this paper uses a directed graph-based approach, similar to [Re18; ZNL17], but with a stronger focus on the inter-service communication. To reconstruct the microservice architecture from the underlying source code, we use a concept from our earlier research, called modular, reusable *source code detectors* [Nt21]. These lightweight source parsers scan the code according to predefined patterns and at the same time ignore any code artifacts unrelated to the communication model. While these detectors must still be adopted to identify technology-specific patterns, implementing and especially maintaining them requires less effort than comparable approaches like a complete AST reconstruction. Based on our formal communication model, we define a set of architectural metrics, both on service level but also on a more fine-grained API-level, that allow us to identify and assess cyclic dependencies within a microservice system.

3 Case Studies

We evaluated our approach by conducting two case studies with two different open-source microservice architectures, *Lakeside Mutual*[3] and *eShopOnContainers*[4], both taken from GitHub. We were able to identify domain-based cycles in both cases. Our case study has also shown that our approach is very well suited for agile development processes, as it requires only the underlying source code without gathering time-consuming runtime information, which makes it particularly interesting for continuous integration pipelines.

4 Conclusion

In this paper, we presented a novel approach for detecting technical and especially domain-based cyclic dependencies in microservice API architectures. Our approach confirms that the detection is possible by relying solely on static source code artifacts. While our source code detectors require some upfront implementation work, our case studies revealed that this effort is manageable and can also be reduced by reusing existing detectors where possible. The study results also show that by using our metrics, even inconspicuous domain-based cycles can be detected. The information gathered through our cycle analysis provides software experts with a solid foundation for making qualified decisions regarding a microservice system's architecture.

[3] https://github.com/Microservice-API-Patterns/LakesideMutual

[4] https://github.com/dotnet-architecture/eShopOnContainers

5 Data Availability

The source code and the data for the project are freely available. The data can be found under the following link: `https://swa.univie.ac.at/identifying-domain-based-cycles/`. This directory contains the generated communication models in png/svg format, together with the textual output of the cycle search and the calculated metrics for each case study. Source code for the detectors, the model generation and cycle search is available under `https://gitlab.swa.univie.ac.at/public-sources/microservice-cycles-public`.

References

[Ne15] Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Beijing Sebastopol, CA, 2015, ISBN: 978-1-4919-5035-7.

[Nt21] Ntentos, E.; Zdun, U.; Plakidas, K.; Genfer, P.; Geiger, S.; Meixner, S.; Hasselbring, W.: Detector-based component model abstraction for microservice-based systems. Computing/, pp. 1–31, 2021.

[Re18] Ren, Z.; Wang, W.; Wu, G.; Gao, C.; Chen, W.; Wei, J.; Huang, T.: Migrating Web Applications from Monolithic Structure to Microservices Architecture. In: Proceedings of the Tenth Asia-Pacific Symposium on Internetware. ACM, Beijing China, pp. 1–10, Sept. 2018, ISBN: 978-1-4503-6590-1.

[Th15] Thones, J.: Microservices. en, IEEE Software 32/1, pp. 116–116, Jan. 2015, ISSN: 0740-7459.

[TL18] Taibi, D.; Lenarduzzi, V.: On the definition of microservice bad smells. IEEE software 35/3, pp. 56–62, 2018.

[Wo17] Wolff, E.: Microservices: Flexible Software Architecture. Addison-Wesley, Boston, 2017, ISBN: 978-0-13-460241-7.

[ZNL17] Zdun, U.; Navarro, E.; Leymann, F.: Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns. In (Maximilien, M.; Vallecillo, A.; Wang, J.; Oriol, M., eds.): Service-Oriented Computing. Vol. 10601, Springer International Publishing, Cham, pp. 411–429, 2017, ISBN: 978-3-319-69034-6 978-3-319-69035-3.

An Evolutionary Analysis of Software-Architecture Smells

Philipp Gnoyke[1], Sandro Schulze[2], Jacob Krüger[3]

Abstract: In this extended abstract, we summarize our paper with the homonymous title published at the International Conference on Software Maintenance and Evolution (ICSME) 2021 [GSK21].

Keywords: software maintenance; software evolution; architecture smells; software quality; technical debt; empirical study

Background: Since user and technological requirements evolve over time, software systems must be maintainable to efficiently adapt; or they are at risk of becoming irrelevant. Design principles aim to keep software understandable, changeable, extensible, reusable, and testable. Violations of such principles on an architectural level are described as *Architecture Smells* (ASs), which often indicate degrading system quality. The amount and severity of smells in a system can furthermore be portrayed with *Technical Debt* (TD), which refers to *interest* in the form of reduced maintainability. Postponing or abandoning quality assurance can lead to *technical bankruptcy* if TD renders a system's evolution economically or practically impossible. In our study, we focused on three types of ASs: First, a *Cyclic Dependency* (CD) is a set of components (classes or packages) that depend on each other, so that their dependency graph forms a cycle. Second, a *Hub-Like Dependency* (HD) represents a component with a high number of incoming and outgoing dependencies. Finally, an *Unstable Dependency* (UD) describes a component that depends on less stable components than itself.

Objective: Only few studies have analyzed the evolution of ASs, as well as their long-term influence on software degradation and TD. A noteworthy study has been performed by Sas et al. [SAAF19], upon which we built. We aimed to expand the current understanding of ASs by providing a more precise conceptual representation of the evolution of ASs and TD, as well as by interpreting empirical observations. For this purpose, we analyzed how the number and composition of ASs in software projects evolve over time. Additionally, we studied how the evolution of ASs relates to the evolution of TD, and how the former impacts the latter. Finally, we identified factors influencing the lifespan of ASs.

Method: We propose an improved technique for tracking ASs over the version history of systems. This especially concerns CDs, for which we introduce the distinction between *subcycles* and *supercycles*. Since supercycles can merge or split between versions, we track their evolution in a novel way with *cyclic dependency evolution graphs*. Furthermore, we propose to distinguish *intra-* and *inter-version smells*, with the former being a smell

[1] Otto-von-Guericke University Magdeburg, Germany philipp.gnoyke@t-online.de
[2] University of Potsdam, Germany sandro.schulze@uni-potsdam.de
[3] Ruhr-University Bochum, Germany jacob.krueger@rub.de

instance in a particular version and the latter being a set of related intra-version smells over multiple versions. We define several properties for both types to assess their severity and growth/lifetime patterns. To empirically evaluate our new concepts and answer our research questions, we analyzed the evolution of ASs and TD in 14 open-source systems from the *Qualitas Corpus* with a total of 485 versions. For detecting intra-version smells and calculating properties, we used a modified version of *Arcan* [Ar17]. Moreover, we implemented our own tool *ASTDEA* to track and compute properties of inter-version smells.

Results: First, we found that TD and the number of ASs tend to increase along the code size of systems. When looking at their relative sizes, they remained mostly stable and decreased in some systems. So, we could not confirm exponential growth patterns. However, we found many ASs that, after being introduced, persisted for the entire observation period. Second, we noted that some AS types like CDs had a greater impact on TD and system degradation, while not always corresponding to their share on the number of ASs. Rather, aspects like the complexity of ASs seem to be more pivotal regarding their impact. Finally, we observed no universal patterns between the lifespan and certain properties of all AS types. However, especially complex CDs among classes tended to persist longer. For UDs, we observed this longevity in case that they overlapped with other smells or had a large difference in stability.

Conclusion: Our results indicate that it is in the interest of practitioners to remove ASs early on to prevent their manifestation, which especially holds true for CDs. Our improved technique for conceiving AS and TD evolution helps practitioners to identify and contend system degradation, as well as researchers to inspire new studies on ASs and TD.

Data Availability: Our replication package[4] contains the source code and compilation of ASTDEA and our modified version of Arcan, the raw output dataset, the queries we used to aggregate information from it, as well as additional diagrams and evaluations.

Bibliography

[Ar17] Arcelli Fontana, Francesca; Pigazzini, Ilaria; Roveda, Riccardo; Tamburri, Damian; Zanoni, Marco; Di Nitto, Elisabetta: Arcan: A Tool for Architectural Smells Detection. In: Proceedings of the International Conference on Software Architecture Workshops (ICSAW). IEEE, 2017.

[GSK21] Gnoyke, Philipp; Schulze, Sandro; Krüger, Jacob: An Evolutionary Analysis of Software-Architecture Smells. In: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2021.

[SAAF19] Sas, Darius; Avgeriou, Paris; Arcelli Fontana, Francesca: Investigating Instability Architectural Smells Evolution: An Exploratory Case Study. In: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019.

[4] https://figshare.com/s/fa17e81cf4f27c84d059

A Survey on the Relevance of the Performance of Model Transformations

Raffaela Groner[1] Katharina Juhnke[1] Stefan Höppner[1] Matthias Tichy[1] Steffen Becker[2]
Vijayshree Vijayshree[2] Sebastian Frank[2]

Abstract: When we are confronted with performance issues in a general-purpose language, like Java, it is a given to us that we have various tools and techniques at our disposal to help us. But is such support also needed when using model transformation languages? To address this question, we conducted a quantitative online survey as part of a mixed methods study with 84 respondents to our questionnaire. Our results show that a certain performance is desired but not always achieved. The developers would like to improve the performance, but they lack insights on how a transformation is performed. As a first step to mitigate this issue, we compiled a list of information regarding the models used, the transformations applied and their execution deemed to be helpful by the participants. Additionally, we used hypotheses tests to investigate possible influencing factors that cause participants to try to improve the performance of transformations. The main relevant factors found in our study are the satisfaction with the execution time, the size of the models used, the relevance of whether a certain execution time is not exceeded in the average case, and the knowledge of how a transformation engine executes a transformation.

Keywords: Model transformation; Performance; Survey; ATL; Henshin; QVTo; Viatra

Techniques for analyzing performance issues of programs written in a general-purpose language are widely used. Due to the increasing popularity of model-driven software development and the use of models at runtime, the question arises whether such a support is also necessary for transformation languages? Currently, research in this area mainly focuses on improving the execution of transformations, such as the work of Boronat [Bo18]. To address this question, we conducted a quantitative online survey as part of a mixed methods study with 84 respondents to our questionnaire.

Our results show that developers using transformation languages also face performance issues, but currently lack support to examine them properly. For example, only 13 of the 84 participants are always satisfied with the execution time of their transformations. 37 participants are often satisfied and 44 participants are only sometimes or rarely satisfied. Using the Mann-Whitney-U-Test, we investigated the differences between participants who have already tried to analyze or improve the performance and those who have not. The test results show that there is a significant difference between these two groups in four aspects: Participants who have tried to improve the performance tend to 1) be less satisfied with the

[1] Ulm University, Institute for Software Engineering and Programming Languages, James–Franck–Ring, D–89069 Ulm, Germany {raffaela.groner, katharina.juhnke, stefan.hoeppner, matthias.tichy}@uni-ulm.de
[2] University of Stuttgart, Institute for Software Engineering, Universitätsstraße, D–70569 Stuttgart, Germany {steffen.becker, vijayshree.vijayshree, sebastian.frank}@iste.uni-stuttgart.de

execution time, 2) use larger models, 3) deem it more important that a certain execution time is not exceeded in the average case, and 4) have more knowledge of how a transformation is executed by the engine. In particular, the knowledge about the transformation execution seems to be an essential influencing factor that causes participants to try to improve the performance of transformations, as we also discovered a very high demand for information. This includes overview information, like the number of applications per transformation rule, or detailed information, like which elements of the of the input model were transformed using which transformation rule. In addition, the developers would like to have more information about the structure of the models used, such as the depth of the inheritance hierarchy or the cycles they contain. Developers also want more support, in the form of profilers or analyses. For example, an analysis that helps to determine an optimal order for applying transformation rules was mentioned.

The results of our study show that techniques for analyzing the performance of transformations are needed and give an overview of the desired information that can help to support developers. More details about our study can be found in our corresponding publication, which was accepted by the Journal of Object Technology (JOT) in 2021 [Gr21a].

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Ti 803/4-1 and BE 4796/3-1

Data Availability

In [Gr21b], we published the following data: 1) Documentation of our search for suitable study participants. 2) The questionnaire used. 3) Additional material used for the design of the questionnaire. 4) The anonymized answers to our questionnaire. 5) Documentation of the results of the hypothesis tests and the corresponding SPSS project.

Literaturverzeichnis

[Bo18] Boronat, Artur: Expressive and Efficient Model Transformation with an Internal DSL of Xtend. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'18). ACM, New York, NY, USA, S. 78–88, 2018.

[Gr21a] Groner, Raffaela; Juhnke, Katharina; Götz, Stefan; Tichy, Matthias; Becker, Steffen; Vijayshree, Vijayshree; Frank, Sebastian: A Survey on the Relevance of the Performance of Model Transformations. Journal of Object Technology, 20(2):2:1–27, 2021.

[Gr21b] Groner, Raffaela; Juhnke, Katharina; Götz, Stefan; Tichy, Matthias; Becker, Steffen; Vijayshree, Vijayshree; Frank, Sebastian: , A Survey on the Relevance of the Performance of Model Transformations: Data of the Participant Search and the Questionnaire. Open Access Repositorium of the Ulm University, 2021.

An Empirical Study of Flaky Tests in Python

Martin Gruber[1] Stephan Lukasczyk[2] Florian Kroiß[3] Gordon Fraser[4]

Abstract: This is a summary of our work presented at the International Conference on Software Testing 2021 [Gr21b]. Tests that cause spurious failures without code changes, i.e., *flaky tests*, hamper regression testing and decrease trust in tests. While the prevalence and importance of flakiness is well established, prior research focused on Java projects, raising questions about generalizability. To provide a better understanding of flakiness, we empirically study the prevalence, causes, and degree of flakiness within 22 352 Python projects containing 876 186 tests. We found flakiness to be equally prevalent in Python as in Java. The reasons, however, are different: Order dependency is a dominant problem, causing 59 % of the 7 571 flaky tests we found. Another 28 % were caused by test infrastructure problems, a previously less considered cause of flakiness. The remaining 13 % can mostly be attributed to the use of network and randomness APIs. Unveiling flaky tests also requires more runs than often assumed: A 95 % confidence that a passing test is not flaky on average would require 170 reruns. Additionally, through our investigations, we created a large dataset of flaky tests that other researchers already started building on [MM21; Ni21].

Keywords: Flaky Test; Python; Empirical Study

1 Methodology

To investigate the diffusion and nature of test flakiness in Python, we scanned the Python package index (PyPI) for repositories with `pytest`-compatible tests, collecting a set of 22 352 projects containing a total of 876 186 test cases. We identified flaky tests within these projects and made a first distinction between order-dependent and non-order-dependent flakiness by executing each project's tests 200 times in default order and 200 times in random order. Splitting the test executions further into 20 batches of 20 reruns each allowed us to also detect infrastructure related problems, which manifest in the passing and failing of entire batches (which are executed on the same machine). Lastly, to retrieve a more fine-grained distinction within non-order-dependent flaky tests, we manually classified 100 of them along previously established root causes. To aid this process, we searched the execution traces of these tests for keywords which indicate specific types of flakiness.

[1] BMW Group | University of Passau, Germany martin.gr.gruber@bmw.de
[2] University of Passau, Germany stephan.lukasczyk@uni-passau.de
[3] University of Passau, Germany kroiss@fim.uni-passau.de
[4] University of Passau, Germany gordon.fraser@uni-passau.de

2 Results

Through repetitive reruns we encountered 7 571 flaky tests in 1 006 projects. 59 % of them are flaky only when executed in a certain test order, therefore being caused by order-dependencies. 28 % are related to infrastructure issue, and the remaining 13 % of non-order-dependent flaky tests are mostly caused by the usage of network and randomness APIs. While these results show that flaky tests are equally prevalent in Python as they are in Java (about one in 200 tests is flaky), the inspection of their root causes reveals strong differences, specially concerning non-order-dependent flakiness: While previous studies found concurrency, especially asynchronous waiting, to be the most common cause of flakiness, we find only little evidence for these in Python. On the other hand, networking and randomness—which are regarded as minor issues by other investigations—are causing 79 of the 100 non-order-dependent flaky tests we inspected. We suspect that these differences are caused by the type of software typically written in Python, which includes scientific software and web application.

Besides the diffusion and the nature of flaky tests in Python, we also investigated the number of reruns needed to unveil a test's flakiness. We abandoned the practice of basing this decision on the number of reruns needed to reveal a test's flakiness once, due to its instability and bad reproducibility. Instead, we developed our own technique by creating a probabilistic model. Our results suggest that in order to be 95 % sure that a passing test case is not flaky, on average at least 170 reruns are required. On the other hand, only one rerun is required to gain the same confidence that a test failure is not caused by flakiness.

3 Data Availability

Our published dataset [Gr21a] includes the test outcomes of all 400 test executions of each test case we investigated as well as the results of a manual classification of the root causes for 100 non-order-dependent flaky tests.

Literatur

[Gr21a] Gruber, M.: An Empirical Study of Flaky Tests in Python, Zenodo, Jan. 2021, URL: https://doi.org/10.5281/zenodo.4450434.

[Gr21b] Gruber, M.; Lukasczyk, S.; Kroiß, F.; Fraser, G.: An empirical study of flaky tests in python. In: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST). IEEE, S. 148–158, 2021.

[MM21] Mjörnman, J.; Mastell, D.: Randomness as a Cause of Test Flakiness, 2021.

[Ni21] Nilsson, J.: Possibilities of automatic detection of „Async Wait“ Flaky tests in Python applications, 2021.

Kann statische Analyse unnützen Code erkennen?

Roman Haas[1] Rainer Niedermayr[2] Tobias Roehm[3] Sven Apel[4]

Abstract: Dieser Beitrag fasst einen im ACM Journal Transactions on Software Engineering and Methodology veröffentlichten Artikel von Haas et al. [Ha20] zusammen. Darin geht es um die Identifikation von unnützem Code, also Code, der nicht mehr benötigt wird. In den meisten gewachsenen Softwaresystemen sind Teile des Codes im Laufe der Zeit unnütz geworden. Unnützer Code ist problematisch, weil dadurch Ressourcen bei der Entwicklung und Wartung verschwendet werden, beispielsweise bei der Vorbereitung für eine bevorstehende Migration oder Zertifizierung. Zwar kann mithilfe eines Profilers nicht mehr produktiv verwendeter Code aufgedeckt werden, aber oft ist es zu zeitaufwendig bis repräsentative Daten erhoben wurden. Im oben genannten Beitrag wird ein auf Codestabilität und -zentralität basierender statischer Analyseansatz vorgeschlagen, um unnützen Code zu identifizieren. Um die Nützlichkeit des Ansatzes zu untersuchen, wurde eine Studie mit 14 Open-Source- und Closed-Source Software-Systemen durchgeführt. Da es kein perfektes Orakel für unnützen Code gibt, wurden die Empfehlungen für unnützen Code auf Basis von Löschungen in der Codehistorie, Laufzeitnutzungsdaten und dem Feedback von 25 Entwicklern aus fünf Software-Projekten evaluiert. Die Studie zeigt, dass aus Stabilitäts- und Zentralitätsinformationen generierte Empfehlungen auf unnützen Code hinweisen. Insgesamt deuten die Ergebnisse darauf hin, dass die statische Analyse schnelles Feedback zu unnützem Code liefern kann und in der Praxis nützlich ist.

Keywords: Unnecessary code, code stability, code centrality

1 Zusammenfassung

Unnützer Code ist solcher, an dem kein Stakeholder Interesse hat. In den allermeisten großen Softwareprojekten werden Teile des Codes unnütz, beispielsweise weil Funktionalität erneut implementiert wird und die Originalimplementierung bestehen bleibt, oder weil sich die Interessen der Stakeholder ändern und so implementierte Funktionalität nicht mehr genutzt wird. Unnützer Code verschwendet Ressourcen unterschiedlichster Art, gleichzeitig lässt sich unnützer Code nur mit großem manuellen Aufwand erkennen. Im hier zusammengefassten Beitrag wird ein Ansatz zur statischen Identifizierung von unnützem Code vorgestellt. Dieser baut auf der Hypothese auf, dass der stabilste und gleichzeitig dezentralste Code in der Abhängigkeitsstruktur der Software wahrscheinlich unnütz ist. Zu diesem Zweck wurde eine Analyse implementiert, die Stabilitäts- und Zentralitätsmaße verwendet, um Dateien als unnützen Code zu empfehlen. Diese Empfehlungen sind als Ausgangspunkt für Praktiker gedacht, die unnützen Code aus ihrer Codebasis entfernen möchten.

[1] CQSE GmbH und Saarbrücken Graduate School of Computer Science
[2] Ergon Informatik AG
[3] CQSE GmbH
[4] Saarland University, Saarland Informatics Campus

Der auf statischer Analyse aufbauende Ansatz berechnet für jede Quelltextdatei des Softwaresystems einen Stabilitäts- und einen Zentralitätsscore. Diese spiegeln die Änderungshäufigkeit in der Historie des Repositories und die Zentralität der korrespondierenden Knoten im Abhängigkeitsgraph der Quelltextdateien wider. Als potentiell unnütz gelten Dateien, die besonders stabil und dezentral sind. Aus Gründen der Nutzerfreundlichkeit werden die Vorschläge für unnützen Code basierend auf der Dateisystemhierarchie gruppiert, sodass beispielsweise ganze Packages als potentiell unnütz vorgeschlagen werden können.

Anhand einer empirischen Studie wurde untersucht, ob mittels Stabilitäts- und Zentralitätsinformationen unnützer Code identifiziert werden kann. Da es keinen Goldstandard für die Frage nach unnützem Code gibt, wurden drei verschiedene Orakel für unnützen Code betrachtet: (1) Löschungen von als unnütz bezeichnetem Code in der Codehistorie, (2) Nutzungsdaten aus Laufzeitumgebungen und (3) Entwicklerbefragungen. Zunächst wurde in einer Untersuchung von in der Historie gelöschtem und als unnütz bezeichnetem Code festgestellt, dass dieser im Durchschnitt stabiler und dezentraler in der Abhängigkeitsstruktur der Studiensubjekte war. Das Kernergebnis der empirischen Studie ist, dass die Ergebnisse aus den drei Orakeln darauf hindeuten, dass Codestabilität und -zentralität zur Identifikation von unnutzem Code hilfreich sind. Einige der als unnütz erkannten Quelltextdateien wurden im Anschluss an die Befragungen sogar tatsächlich gelöscht, wodurch der Nutzen der vorgestellten Analyse unterstrichen wird. Nichtsdestotrotz muss mit falsch-positiven Empfehlungen gerechnet werden, insbesondere falls dynamische oder externe Codeabhängigkeiten bestehen.

Data Availability

In einem Archiv unter `https://figshare.com/s/8c3f63d2c620d1aae83a` sind die Konfigurationsdaten und Ergebnisauswertungen der Studie frei verfügbar.

Danksagung

Diese Arbeit wurde in Teilen aus Mitteln des Bundesministeriums für Bildung und Forschung (BMBF), Förderkennzeichen “SOFIE, 01IS18012A” finanziert. Apels Arbeit wurde durch die Deutsche Forschungsgemeinschaft (DFG) unterstützt (AP 206/11-1). Für diesen Beitrag sind die Autoren verantwortlich.

Literatur

[Ha20] Haas, R.; Niedermayr, R.; Roehm, T.; Apel, S.: Is Static Analysis Able to Identify Unnecessary Source Code? ACM Transactions on Software Engineering and Methodology 29/1, S. 1–23, 2020.

CoVEGI: Cooperative Verification via Externally Generated Invariants

Jan Haltermann[1], Heike Wehrheim[2]

Abstract: Software verification has recently made enormous progress. To keep their tools up to date with novel methods and enhanced techniques, tool developers integrate these within their own framework almost exclusively by re-implementation. While this allows for a conceptual re-use of methods, it nevertheless requires novel implementations.
Our configurable framework named CoVEGI employs cooperative verification in order to avoid re-implementation and enable usage of novel tools as black-box components in verification. Specifically, cooperation is employed for invariant generation, which is key to the success of a verification run. CoVEGI allows a main verification tool to delegate the task of invariant generation to one or several specialized helper invariant generators, utilizing their results within its verification run. The experimental evaluation shows that the use of CoVEGI can increase the number of correctly verified tasks up to 17%, without increasing the used resources.

Keywords: Cooperation; Software Verification; Invariant Generation

The CoVEGI Framework

Cooperative verification is an approach to avoid unnecessary re-implementations. It builds on the idea of letting tools (and thus techniques) cooperate on verification tasks as black boxes, thereby leveraging the tools' individual strengths. The CoVEGI framework proposes a novel type of cooperation leveraging the strength of specialized tools for invariant generation.

The framework is depicted in Fig. 1 and consists of a *main verifier* and several *helper invariant generators*. The main verifier controls the overall verification process and can delegate the task of generating invariants to the helpers, as well as continue its verification process with the results from the helpers. When the helper invariant generators finish the computation of candidate invariants, these invariants are either injected into the running analysis or the main verifier is restarted with them. In addition, the framework offers several further configuration options (e.g. timeout for helpers or to wait for all helpers to terminate before witness injection). The information used for cooperation is exchanged using the existing and widely used *correctness witnesses* [Be16]. To use off-the-shelf tools as helpers

[1] Carl von Ossietzky University of Oldenburg, Department of Computing Science, Ammerländer Heerstraße 114-118, 26129 Oldenburg, Germany, jan.haltermann@uol.de
[2] Carl von Ossietzky University of Oldenburg, Department of Computing Science, Ammerländer Heerstraße 114-118, 26129 Oldenburg, Germany, heike.wehrheim@uol.de

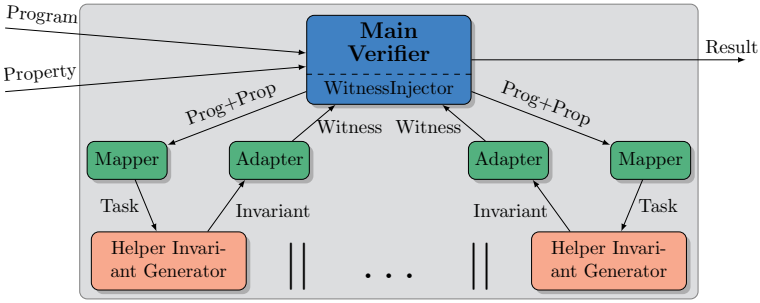

Fig. 1: Cooperative verification via externally generated invariants

that require a specific input format or do not produce correctness witnesses, CoVEGI foresees two additional sorts of components: A *mapper* translates the task into the helpers required input format and an *adapter* encodes the generated invariant as correctness witness.

The implementation of the CoVEGI framework is realized in CPAchecker and comprises two main verifiers (employing k-induction and predicate abstraction) and three helpers (SeaHorn, UltimateAutomizer, VeriAbs). We implemented two adapters, mainly for transforming invariants expressed in the LLVM IR language into correctness witnesses. We conduct an evaluation using a number of benchmarks from SV-COMP 2020. The experiments show that (1) using CoVEGI increases the number of correctly solved tasks by 12% for k-induction as main verifier and 14% for predicate abstraction, (2) the collaborative invariant generation does not negatively impact the effectiveness and (3) using two helpers can increase the number of correctly solved tasks even further (up to 17% for predicate abstraction). A detailed evaluation is available in [HW21a].

Data Availability
The implementation and all experimental data are archived and available at Zenodo [HW21b].

References

[Be16] Beyer, D.; Dangl, M.; Dietsch, D.; Heizmann, M.: Correctness witnesses: exchanging verification results between verifiers. In: Proc. FSE. ACM, pp. 326–337, 2016.

[HW21a] Haltermann, J.; Wehrheim, H.: CoVEGI: Cooperative Verification via Externally Generated Invariants. In: Proc. FASE. LNCS, Springer, pp. 108–129, 2021.

[HW21b] Haltermann, J.; Wehrheim, H.: Replication Package for article 'CoVEGI: Cooperative Verification via Externally Generated Invariants', Apr. 2021, URL: https://doi.org/10.5281/zenodo.5763499.

Contrasting Dedicated Model Transformation Languages vs. General Purpose Languages: A Historical Perspective on ATL vs. Java based on Complexity and Size – Extended Abstract

Stefan Höppner[1], Timo Kehrer[2], Matthias Tichy[1]

Abstract:

Model transformations are one key concept of model-driven engineering, and model transformation languages (MTLs) emerged with its popularity about 15 to 20 years ago. MTLs claim to ease model transformation development by abstracting from recurring transformation aspects and hiding complex semantics behind simple and intuitive syntax. Nonetheless, MTLs are rarely adopted in practice, there is still no empirical evidence for the claim of easier development, and the argument of abstraction deserves a fresh look in the light of modern general-purpose languages (GPLs) which have undergone a significant evolution in the last two decades. In our SoSyM paper [HKT21], we report on a study in which we compare the complexity and size of model transformations written in three different languages, namely (i) the Atlas Transformation Language (ATL), (ii) Java SE5 (2004-2009), and (iii) Java SE14 (2020); the Java transformations are derived from an ATL specification using a translation schema we developed. Based on the results of these comparisons, we discuss the concrete advancements in newer Java versions. We also discuss to which extent new language advancements justify writing transformations in a GPL rather than a dedicated MTL. We further indicate potential avenues for future research on the comparison of MTLs and GPLs.

Keywords: ATL; Java; Model transformations; Model transformation language; General purpose language; Comparison; MTL vs. GPL; Historical Perspecitve; Complexity Measure; Size Measure

In the literature, many advantages are ascribed to model transformation languages, such as better analysability, comprehensibility or expressiveness [GTG21]. Nowadays, however, such claims have two main flaws. First, there is a lack of actual evidence to have confidence in their genuineness [GTG21]. And second, most of these claims emerged around 15 years ago when the first model transformation languages were introduced. From this follows the presumption that transformations can just as well be written in a modern GPL. This presumption is confirmed by a community discussion on the future of model transformation languages [BCG19], and partially by an empirical study conducted by Hebig et al. [He18]. The presumption is mainly rooted in the idea that new language features allow developers to heavily reduce boilerplate code that MTLs claim to abstract away from. To validate and

[1] Ulm University, Institute for Software Engineering and Compiler Construction, James-Franck-Ring 1, 89081 Ulm, Germany [stefan.hoeppner|matthias.tichy]@uni-ulm.de

[2] Humboldt University Berlin, Institute for Informatics, Unter den Linden 6, 10099, Germany timo.kehrer@informatik.hu-berlin.de

better understand this argumentation, we elected to compare 12 transformations written in the Atlas Transformation Language (ATL), one of the most widely known MTLs, with the same transformations written in a recent version of Java (Java SE14), as well as an older version (Java SE5), that was recent when ATL was introduced. The **goal** of our SoSyM paper [HKT21] was to (i) analyse how much transformation code, written in Java, has improved over the years and (ii) contextualise these improvements by relating them to ATL code. To do so, we opted to use both size and complexity measures to gain insights into the makeup of transformation code for the discussion.

Our results show that new features introduced in Java since 2006 help to significantly reduce the complexity and lines of code of transformations written in Java. However, they do not reduce the number of words required to write the same transformations. This suggests an ability to express more information dense code in newer Java versions. Transformations in newer Java versions can also be written in a more data driven way that more closely resembles the way they are defined in MTLs. We also showed that, while the overall complexity of transformations is reduced, the distribution of how much of that complexity stems from code that implements functionality that ATL and other MTLs can hide from the developer stays about the same. This observation is further supported by the analysis of code size distribution. Here, we found that while large parts of the transformation classes relate to the transformation process itself, within those parts there is still significant overhead from tracing as well as general supplemental code required for the transformations to work. It follows, that while the overall complexity is reduced, the overhead entailed by using a general purpose language for writing model transformations is still present.

Overall we find that more recent Java versions make development of transformations easier because less work is required to set up a working transformation, and the creation of output elements and the assignment of their attributes are now a more prominent aspect within the code. From our results and experience with this and other projects, we also conclude that general purpose languages are most suitable for transformations where little to no tracing is required because the overhead associated with this transformation aspect is the most prominent one and holds the most potential for errors. However, we also believe that advanced features such as property preservation verification or bidirectional and incremental transformation development cannot currently be implemented with justifiable effort in a general purpose language.

Data Availability

All transformation and analysis code involved in our work is publicly accessible on the OPARU system of our university [HTK21].

Bibliography

[BCG19] Burgueño, Loli; Cabot, Jordi; Gerard, Sebastien: The Future of Model Transformation Languages: An Open Community Discussion. Journal of Object Technology, 18(3):7:1–11, July 2019. The 12th International Conference on Model Transformations.

[GTG21] Götz, Stefan; Tichy, Matthias; Groner, Raffaela: Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review. Software and Systems Modeling, 20(2):469–503, 2021.

[He18] Hebig, Regina; Seidl, Christoph; Berger, Thorsten; Pedersen, John Kook; Wasowski, Andrzej: Model Transformation Languages Under a Magnifying Glass: A Controlled Experiment with Xtend, ATL, and QVT. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA, 2018.

[HKT21] Höppner, Stefan; Kehrer, Timo; Tichy, Matthias: Contrasting dedicated model transformation languages versus general purpose languages: a historical perspective on ATL versus Java based on complexity and size. Software and Systems Modeling, 2021.

[HTK21] Höppner, Stefan; Tichy, Matthias; Kehrer, Timo: , Contrasting Dedicated Model Transformation Languages vs. General Purpose Languages: A Historical Perspective on ATL vs. Java based on Complexity and Size: Supplementary Materials, 2021.

Towards Shaping the Software Lifecycle with Methods and Practices

Jil Klünder[1] Melanie Busch[2] Natalie Dehn[3] Oliver Karras[4]

Abstract: Software development processes are very diverse, and the use of methods and practices varies in many facets. However, it is unclear for what reason single methods and practices are included in a development process and which phases they are meant to support. Based on a survey with 27 practitioners, we provide a mapping of methods and practices to different project phases and vice versa. Our results indicate that, when devising a development process, it is worth a thought if all phases of the software lifecycle are addressed and if the use of selected methods and practices is meaningful.

This summary refers to the paper *Towards Shaping the Software Lifecycle with Methods and Practices* [Kl21a]. This paper was published as full research paper in the proceedings of the Joint International Conference on Software and System Processes and International Conference on Global Software Engineering and received the IEEE Best Industry Experience Paper Award.

Keywords: Software lifecycle; software process; survey study

1 Introduction

Software development processes need to be adjusted to the project and the development team, resulting in a wide variety in the use of methods and practices [Kl20]. The analysis of a large international survey yields that there are, for example, many variations of Scrum that are somewhat unique, but nevertheless widely distributed in practice. This diverse use of basic methods such as Scrum or eXtreme Programming raises the question for what purpose they are used in the development process, i.e., whether they are for example used for project management or implementation.

In a survey with 27 practitioners, we analyzed the use of methods and practices throughout a software development process, revealing three insights: (1) There are discrepancies between the intended use of methods and practices according to literature and the real use in practice, (2) practices are used more consistently than methods, and (3) some phases of the software lifecycle are hardly covered by widely distributed methods and practices.

[1] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, jil.kluender@inf.uni-hannover.de
[2] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, melanie.busch@inf.uni-hannover.de
[3] Leibniz Universität Hannover, Welfengarten 1, 30167 Hannover, natalie.dehn@live.de
[4] TIB - Leibniz Information Centre for Science and Technology, Welfengarten 1B, 30167 Hannover, oliver.karras@tib.eu

2 Results

An analysis of 27 data points revealed that the top-3 methods Scrum, DevOps, and Kanban are used very diversely in the development processes. For example, Scrum as project management framework is reported to be mainly used for implementation. DevOps (for operation) and Kanban (for work organization) are used for a number of different project disciplines. Compared to the methods, the practices are used more stringent: The top-3 practices code reviews, coding standards, and daily stand-up meetings are mainly used for implementation (independent from their intended use).

In addition, we observe differences in the coverage of the project disciplines by the methods and practices. While project disciplines such as project management or implementation are frequently covered, other disciplines such as maintenance and evolution or configuration management are hardly covered by the methods and practices used in our study.

3 Conclusion and Future Work

Our paper [Kl21a] shows that several methods and practices are not used as they are meant to. Consequently, practitioners should think about the usefulness of a specific method in a given context and stop using methods just because everybody does so. Following this line of thought also points to the missing baseline of what, e.g., Scrum really is about. Therefore, future research should look at the practitioners' mental models of the methods and practices. And practitioners should be aware of three things: (1) whether they deviate from a specific method/practice, (2) how they do it, and (3) why. As long as these questions can be meaningfully answered, deviating from original ideas can be suitable. Otherwise, they should critically question whether the given process needs to be adjusted.

Data Availability

The survey study and the anonymized answers are available online [Kl21b].

Bibliography

[Kl20] Klünder, Jil; Karajic, Dzejlana; Tell, Paolo; Karras, Oliver; Münkel, Christian; Münch, Jürgen; MacDonell, Stephen G; Hebig, Regina; Kuhrmann, Marco: Determining context factors for hybrid development methods with trained models. IEEE/ACM International Conference on Software and System Processes, 2020.

[Kl21a] Klünder, Jil; Busch, Melanie; Dehn, Natalie; Karras, Oliver: Towards Shaping the Software Lifecycle with Methods and Practices. IEEE/ACM Joint International Conference on Software and System Processes and International Conference on Global Software Engineering, 2021.

[Kl21b] Klünder, Jil; Busch, Melanie; Dehn, Natalie; Karras, Oliver: Methods and Practices in Different Software Project Disciplines (Dataset). Zenodo, 2021.

Continuous API Evolution in Heterogenous Enterprise Software Systems

Holger Knoche[1] Wilhelm Hasselbring[2]

Abstract: This contribution has been published at the 18th IEEE International Conference on Software Architecture (ICSA 2021) [KH21a].

Keywords: API Evolution; Enterprise Software

1 Introduction

The ability to independently deploy parts of a software system is one of the cornerstones of modern software development, and allows for these parts to evolve independently and at different speeds.

A major challenge of such independent deployment, however, is to ensure that despite their individual evolution, the interfaces between interacting parts remain compatible. This is especially important for enterprise software systems, which are often highly integrated and based on heterogenous IT infrastructures [Co05; Ha00].

2 Goals

In recent years, we have observed that also more conservative companies, such as banks and insurance companies, have begun to embrace such practices for their internal software systems [KH19], in particular for migrating legacy systems towards microservices [KH18]. We have designed an approach addressing the following goals:

1. The approach should be easy to use for the developers, and help to avoid common mistakes
2. The approach should have as little impact as possible on the developer's day-to-day work
3. The approach must also support aged implementation technologies and platforms
4. The runtime performance impact of the approach must be acceptable

[1] Software Engineering Group, Kiel University, Kiel, Germany hkn@informatik.uni-kiel.de
[2] Software Engineering Group, Kiel University, Kiel, Germany wha@informatik.uni-kiel.de

3 Approach

In [KH21a], we present an approach for interface evolution that is easy to use for developers, and also addresses typical challenges of heterogenous enterprise software, especially legacy system integration.

4 Data Availability

The data is available at Zenodo [KH21b]. Source code is available at GitHub (`https://github.com/holgerknoche/gutta-apievolution`).

Literatur

[Co05] Conrad, S.; Hasselbring, W.; Koschel, A.; Tritsch, R.: Enterprise Application Integration. Spektrum Akademischer Verlag, 2005.

[Ha00] Hasselbring, W.: Information System Integration. Communications of the ACM 43/6, S. 32–36, 2000, URL: `https://doi.org/10.1145/336460.336472`.

[KH18] Knoche, H.; Hasselbring, W.: Using Microservices for Legacy Software Modernization. IEEE Software 35/3, S. 44–49, Mai 2018, URL: `https://doi.org/10.1109/MS.2018.2141035`.

[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, URL: `https://doi.org/10.18417/emisa.14.1`.

[KH21a] Knoche, H.; Hasselbring, W.: Continuous API Evolution in Heterogenous Enterprise Software Systems. In: 2021 IEEE 18th International Conference on Software Architecture (ICSA). IEEE, S. 58–68, März 2021, URL: `https://doi.org/10.1109/icsa51549.2021.00014`.

[KH21b] Knoche, H.; Hasselbring, W.: Data for: Continuous API Evolution in Heterogenous Enterprise Software Systems, Zenodo, Okt. 2021, URL: `https://doi.org/10.5281/zenodo.5574601`.

Lars Grunske, Janet Siegmund, Andreas Vogelsang (Hrsg.): SE 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 51

How EvoStreets Are Observed in Three-Dimensional and Virtual Reality Environments

Rainer Koschke[1], Marcel Steinbeck[2]

Abstract: We present our paper published in the proceedings of the 27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). In an experiment in which 34 participants had to analyze cloning in existing software systems using the EvoStreets visualization, we found indications that the movement patterns of the participants differ depending on whether they are in a 2.5D (pseudo-3D; monitor with keyboard and mouse) or virtual reality (VR) environment. In a follow-up study, which we present in this work, we analyzed the results of this experiment in more details, to examine whether not only movement is affected by these environments, but also the way how EvoStreets are observed. Beyond that, the paper proposes six visualization and user interaction concepts that are specific to the kind of environment.

Keywords: Software Visualization; Virtual and Augmented Reality; Code Cities; EvoStreets

1 Summary

In order to assist developers in software maintenance tasks, different kinds of visualizations have been developed in the last decades. One of the most promising visualization concepts is the *Code-City* [WL07] metaphor and its derivative EvoStreets [St13]. In previous studies [SKR19a, SKR19b] we compared EvoStreets deployed in different environments, namely 2D (orthographic; monitor, keyboard and mouse), 2.5D (pseudo-3D; monitor, keyboard and mouse), and VR (head-mounted displays and hand-held controllers), to find answers to the question whether a certain environment is better suited for analyzing software clones than the others—in all three environments, the size of the virtual representation of the participants was chosen so that they are perspectively located in a city with large buildings. We could not find statistically significant differences among the three environments with regard to the time needed by the participants to solve the tasks we had set as well as the correctness of the supplied answers. According to our results, there seems to be a trend that comparing blocks in the 2.5D environment is more error prone than in 2D and VR, though [SKR19a]. Also, we found evidence that the *path length*, *average speed*, and *occupied volume* differ significantly between the 2.5D and the VR environments [SKR19b].

[1] Universität Bremen, AG Softwaretechnik, Bibliothekstrasse 5, 28359 Bremen, Deutschland
koschke@uni-bremen.de
[2] Universität Bremen, AG Softwaretechnik, Bibliothekstrasse 5, 28359 Bremen, Deutschland
marcel@informatik.uni-bremen.de

In [SKR20] (the paper presented in this work), we extracted so called *viewpoints* from the positional data of [SKR19b]. A viewpoint consists of a spatial position and the duration spent at that particular position—we call this time *residence time*. With regard to viewpoints we tried to answer the following research questions: *i) Do the two different environments, 2.5D and VR, effect viewpoints?* and *ii) Are there patterns regarding the changes of viewpoints that are specific to the 2.5D and VR environments?* We found that the height of viewpoints and the distance between consecutive viewpoints is significantly lower in the VR environment for two out of the three tasks. Although we could not find enough evidence to confirm our hypotheses (see [SKR20] for more details), we found indications that the height of viewpoints becomes larger at later phases of an analysis in the 2.5D environment for two out of the three tasks.

Conclusions: The results of our studies can be concluded as follows. Neither of the environments seems to be better suited for analyzing software clones in EvoStreets than the others. However, in 2.5D users seem to prefer viewing EvoStreets from farther away and to increase the distance to the city in the process of visual analysis. As a result, greater distances have to be covered by the users in 2.5D to view the EvoStreets under examination from different angles. Based on our findings, we proposed six different visualization and user interaction concepts in our earlier paper [SKR20] that adapt to the characteristics of the 2.5D and VR environments.

2 Data Availability

Our data of the experiment are available online[3]. The ZIP archive contains the two CSV files *TaskData.csv* and *ViewPoints.csv*, as well as the file *README.md* which describes the structure of the CSV files.

Literaturverzeichnis

[SKR19a] Steinbeck, Marcel; Koschke, Rainer; Rüdel, Marc-Oliver: Comparing the EvoStreet Visualization Technique in Two- and Three-Dimensional Environments—A Controlled Experiment. In: International Conference on Program Comprehension. S. 231–242, 2019.

[SKR19b] Steinbeck, Marcel; Koschke, Rainer; Rüdel, Marc-Oliver: Movement Patterns and Trajectories in Three-Dimensional Software Visualization. In: IEEE International Working Conference on Source Code Analysis and Manipulation. S. 163–174, 2019.

[SKR20] Steinbeck, Marcel; Koschke, Rainer; Rüdel, Marc-Oliver: How EvoStreets Are Observed in Three-Dimensional and Virtual Reality Environments. In: IEEE International Conference on Software Analysis, Evolution and Reengineering. S. 332–343, 2020.

[St13] Steinbrückner, Frank: Consistent Software Cities: supporting comprehension of evolving software systems. Dissertation, Brandenburgischen Technischen Universität Cottbus, Cottbus, 06 2013.

[WL07] Wettel, R.; Lanza, M.: Visualizing Software Systems as Cities. In: IEEE International Workshop on Visualizing Software for Understanding and Analysis. S. 92–99, June 2007.

[3] `http://softwareclones.org/research-data.php`

Dynamic Updates of Virtual PLCs deployed as Kubernetes Microservices

Heiko Koziolek, Andreas Burger, Abdulla PP,
Julius Rückert, Shardul Sonar, Pablo Rodriguez[1]

Abstract: 15th European Conference on Software Architecture (ECSA 2021)

Industrial control systems (e.g. programmable logic controllers, PLC or distributed control systems, DCS) cyclically execute control algorithms to automated production processes. Nowadays, for many applications their deployment is moving from dedicated embedded controllers into more flexible container environments, thus becoming "Virtual PLCs". It is difficult to update such containerized Virtual PLCs during runtime by switching to a newer instance, which requires transferring internal state. Former research has only proposed dynamic update approaches for single embedded controllers, while other work introduced special Kubernetes (K8s) state replication approaches, which did not support cyclic real-time applications. We propose a dynamic update mechanism for Virtual PLCs deployed as K8s microservices. This approach is based on a purpose-built K8s Operator and allows control application updates without halting the production processes. Our experimental validation shows that the approach can support the internal state transfer of large industrial control applications (100.000 state variables) within only 15 percent of the available cycle slack time. Therefore, the approach creates vast opportunities for updating applications on-the-fly and migrating them between nodes in a cloud-native fashion.

Keywords: Software architecture; PLC programs; Kubernetes; Docker; Microservices; Kubernetes Operator; Performance evaluation; Stateful applications; Dynamic software updates; OPC UA

Data Availability

Experimental data from this paper is available at:
https://github.com/hkoziolek/ECSA2021-experiment-data

Bibliography

[Ko21] Koziolek, Heiko; Burger, Andreas; Abdulla, P. P.; Rückert, Julius; Sonar, Shardul; Rodriguez, Pablo: Dynamic Updates of Virtual PLCs Deployed as Kubernetes Microservices. In (Biffl, Stefan; Navarro, Elena; Löwe, Welf; Sirjani, Marjan; Mirandola, Raffaela; Weyns, Danny, eds): Software Architecture - 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings. volume 12857 of Lecture Notes in Computer Science. Springer, pp. 3–19, 2021.

[1] ABB Research, Wallstadter Str. 59, D-68526 Ladenburg, Germany heiko.koziolek@de.abb.com

Model-based resource analysis and synthesis of service-oriented automotive software architectures

Stefan Kugele[1], Philipp Obergfell[2], Eric Sax[3]

Abstract: This summary refers to the paper *Model-based resource analysis and synthesis of service-oriented automotive software architectures* [KOS21]. This paper has been published in the Journal on Software and Systems Modeling (SoSyM) in September 2021.

Context: One drawback of today's automotive software architectures is their strong integration into the onboard communication network based on predefined dependencies at design time. The idea is to reduce this rigid integration and technological dependencies by using a service-oriented architecture (SOA) that dynamically regulates network communication at run-time. **Aim:** We target to provide a methodology for analysing hardware resources and synthesising automotive service-oriented architectures based on platform-independent service models that are transformed in a subsequent step into a platform-specific architecture realisation process. **Approach:** For the first part, we apply design space exploration and simulation to derive analysed deployment configurations at an early development stage. We refine these configurations to AUTOSAR Adaptive software architecture models required for a subsequent implementation process for the platform-specific part. **Result:** We present optimal deployment configurations for our next generation of E/E architecture. We also provide simulation results that demonstrate the ability of these configurations to meet the run time requirements.

Keywords: Automotive; Service-oriented architecture; Real-time behaviour; Model-based design

During the last decades, thousands of primarily software-controlled functions were included in modern cars, which are executed on many electronic control units (ECU). Driving forces for this development were (i) safety requirements, (ii) customer demands for more comfort and the newest infotainment systems, and (iii) advanced driver assistance systems allowing to reach higher levels of driving automation. These driving forces led to the current electric/electronic (E/E) architectures best characterised as historically grown, mostly federated, partly integrated architectures with often pragmatic, cost-efficient, and ad-hoc solutions. More than 100 custom-built ECUs realise the behaviour of the functions in an interplay of timed signals sent via heterogeneous bus systems with gateway structures.

In current AUTOSAR Classic Platform-based software architectures, communication paths are static, i.e., distributed software applications are strictly bound to ECUs at the development

[1] Technische Hochschule Ingolstadt, Research Institute AImotion Bavaria, 85049 Ingolstadt, Germany, Stefan.Kugele@thi.de

[2] BMW Group Research, New Technologies, Innovations, 85748 Garching bei München, Germany, Philipp.Obergfell@bmw.de

[3] Karlsruhe Institute of Technology, Institute for Information Processing Technologies, 76131 Karlsruhe, Germany, eric.sax@kit.edu

stage. This, however, impairs the possibility of developing both parts independently of each other. This ECU or message-centric focus can hardly cope with future automotive software and systems engineering challenges. During the entire life cycle, the software contained in the vehicle must always be kept up to date to follow customers' demands and to be in line with the state-of-the-art especially wrt. functional safety and security. To foster faster adoption of improved features and innovative services, more *agile* development methods need to be built into established processes. When designing a system's software architecture, the primary goals are to develop safe and secure, performant, flexible, adaptive, and maintainable systems. *Service-oriented architectures* are known for supporting the design of flexible systems since *late binding* at *run-time* facilitates the integration of new functionalities and services. SOA paradigms help to decouple software and hardware development to a large extent, facilitating fast and light-weight software updates in a fast-moving agile development process based on the idea of *continuous software engineering*.

Our guiding research question for this work is: *In a pre-development project, can the unification between a specification of an independently created novel E/E architecture and initial estimates of the resource consumption of the service-oriented software architecture succeed and a system specification be derived from it?*

To answer this research question, we presented a platform-independent meta-model for service-oriented architectures, drew a methodology for deploying corresponding platform-independent models on a centralised E/E architecture, and finally embedded the gained results in a platform-specific software development process based on AUTOSAR AP. For this purpose, experts first determined resource requirements; in particular, they provided time and memory requirements and safety assessments. The same values were determined for the centralised computing platform. In an approach that used simulation to estimate end-to-end latencies for a sub-function of automated driving, namely the provision of the environment model and a constraint-based design space exploration technique, deployment candidates were synthesised and evaluated. Since the architecture in this pre-development project is to be designed for SAE level 4 automated driving, special safety measures had to be taken into account in the architecture on both the hardware and software sides. The SOA-based software architecture can reconfigure itself if the environment model is not available in time so that fail-operational models can be used. We were able to show through simulation that all real-time requirements could be met for the reconfigurations considered. In summary, it can be said that the leading research question could be answered positively.

Data Availability The original paper [KOS21] is freely available as *Open Access* under the given reference. Unfortunately, we could not provide more detailed data and especially models for reasons of confidentiality.

References

[KOS21] Kugele, S.; Obergfell, P.; Sax, E.: Model-based resource analysis and synthesis of service-oriented automotive software architectures. Softw. Syst. Model. 20/6, pp. 1945–1975, 2021, URL: https://doi.org/10.1007/s10270-021-00896-9.

A Large-Scale Longitudinal Study of Flaky Tests

Wing Lam[1], Stefan Winter[2], Anjiang Wei[3], Tao Xie[4], Darko Marinov[5], Jonathan Bell[6]

Abstract: Flaky tests that non-deterministically pass or fail without any code changes constitute an impediment to regression testing. To understand when and how flaky tests can be detected most efficiently, we analyzed the commit histories of known flaky tests. We find that 75% of flaky tests are flaky when added, indicating substantial value for developers to run detectors specifically on newly added tests. The percentage of flaky tests that can be detected early increases to 85% when detectors are run on both newly added and directly modified tests.

Keywords: Flaky Tests; Regression Testing

Introduction: Flaky tests are software tests than can both pass and fail without changes to the code under test (CUT) or the test code. They constitute a major impediment to regression testing as their failures do not indicate actual software regressions in the CUT. Such false alarms limit the utility of automated regression tests and adversely affect development productivity, because they can mislead developers to debug failures in the wrong place and prevent the CUT's timely integration with other components of the project. The problem has attracted significant attention in the scientific community (see [Pa21] for an overview).

Because the root causes of flaky tests are manifold [Lu14], a variety of detectors have been proposed to assist with their automatic detection and removal. However, applying these detectors is costly, as they commonly require large numbers of test re-executions. Therefore, it is important to understand whether flaky tests are already flaky by the time they are added to the test code base or whether they only become flaky later as the project evolves. In the former case it would be sufficient to apply flaky test detectors when new tests are added to the test code base, whereas in the latter case detectors would need run periodically as the (test) code base evolves. To answer when flaky tests become flaky, we have conducted an empirical study on a large dataset of projects with known flaky tests.

Study Design: Our study has been based on the iDFlakies dataset of tests that are known to be flaky at a certain commit (*iDFlakies-commit*) in the projects' git histories. For each such flaky test, we performed the following steps (illustrated and summarized in Fig. 1) to determine when in the project history it becomes flaky.

[1] George Mason University, USA
[2] LMU Munich, Germany
[3] Stanford University, USA
[4] Peking University, China
[5] University of Illinois at Urbana-Champaign, USA
[6] Northeastern University, USA

1. As flaky tests can manifest differently in different execution environments, we reconfirmed that the tests are indeed flaky in the iDFlakies-commit. For this purpose we employed two state-of-the-art flaky test detectors, iDFlakies and NonDex, which target different types of flaky tests.
2. In a second step, we made sure that the identified flaky tests can be reproduced as flaky. To achieve a more fine-grained categorization of identified flaky tests, we replaced iDFlakies with two more targeted detectors (Isolation and OBO).
3. We identified the *test introducing commit (TIC)* for each such test and checked whether the test was flaky at that commit.
4. For the tests that we did not identify as flaky at their TIC, we searched the commit history between the TIC and the iDFlakies-commit for the *flakiness introducing commit* (FIC), at which the test first becomes detectable as flaky.

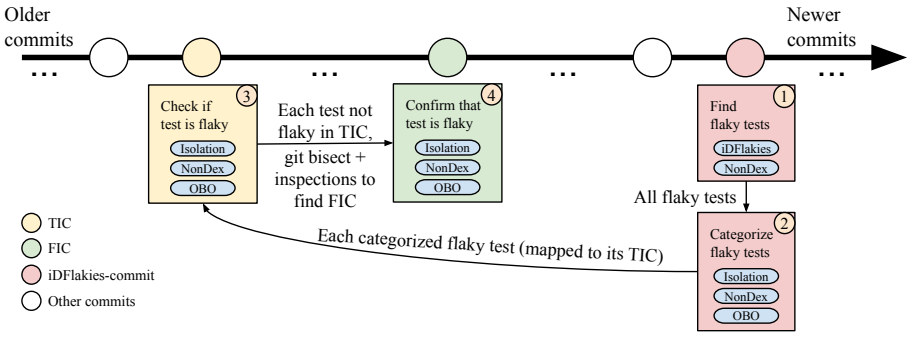

Fig. 1: Overview of the study design.

Results: In the first step of our study we identified 684 flaky tests in their iDFlakies-commit, out of which we were able to confirm and categorize 432. For 245 of these, we were able to compile and run the tests on their TIC and 184 of them (75%) were identified as flaky by detectors in our study. This indicates substantial value for developers to run detectors specifically on newly added tests. For the remaining 61 tests, we identified and inspected their FIC, which revealed that 24 of them become flaky when their test class is altered. Therefore, running detectors on newly added *or directly modified* tests would yield a 85% detection rate for the flaky tests in our study. Running detectors upon test class modifications would detect 8 more flaky tests, but with a median distance of 61 commits or 22 days (median) after their FIC. The remaining 29 flaky tests could be detected 3 commits or 3.6 days after their FIC, but only if detectors run on all tests whenever any test code is changed.

Data Availability: The original article detailing our study [La20] has been published in PACMPL, which is a Gold Open Access journal. The data and data processing scripts from our study are available on a dedicated project website at `https://sites.google.com/view/first-commit-flaky-test`. The data has been integrated in the Illinois Dataset of Flaky Tests (IDoFT) at `http://mir.cs.illinois.edu/flakytests/`.

Bibliography

[La20] Lam, Wing; Winter, Stefan; Wei, Anjiang; Xie, Tao; Marinov, Darko; Bell, Jonathan: A Large-Scale Longitudinal Study of Flaky Tests. Proc. ACM Program. Lang., 4(OOPSLA), 2020.

[Lu14] Luo, Qingzhou; Hariri, Farah; Eloussi, Lamyaa; Marinov, Darko: An Empirical Analysis of Flaky Tests. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. FSE 2014, pp. 643–653, 2014.

[Pa21] Parry, Owain; Kapfhammer, Gregory M.; Hilton, Michael; McMinn, Phil: A Survey of Flaky Tests. ACM Trans. Softw. Eng. Methodol., 31(1), oct 2021.

Lars Grunske, Janet Siegmund, Andreas Vogelsang (Hrsg.): SE 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 61

IDE Support for Cloud-Based Static Analyses

Linghui Luo[1], Eric Bodden[2]

Abstract: We present a user study with developers at Amazon Web Services on their expectations of IDE support for cloud-based static analyses. The paper was originally presented at ESEC/FSE 2021. Many companies are providing Static Application Security Testing (SAST) tools as a service. These tools fit well into CI/CD, because CI/CD allows time for deep static analyses on large code bases and prevents vulnerabilities in the early stages of the development lifecycle. In CI/CD, the SAST tools usually run in the cloud and provide findings via a web interface. Recent studies show that developers prefer seeing the findings of these tools directly in their IDEs. Most tools with IDE integration run lightweight static analyses and can give feedback at coding time, but SAST tools take longer to run and usually are not able to do so. Can developers interact directly with a cloud-based SAST tool that is typically used in CI/CD through their IDE? We conducted a user study to explore how such IDE support should be designed. Through this study we identified the key design elements expected by developers and investigated whether an IDE solution fits better into developers' workflow in comparison to a web-based solution.

Keywords: IDE integration; Static analysis; Cloud service; SAST tools; Security testing

1 Summary

More and more software companies are integrating Static Application Security Testing (SAST) tools into their continuous integration (CI) or continuous delivery (CD) pipelines. Popular SAST tools are often cloud-based and offer hooks to integrate with CI/CD systems such as GitHub Actions, Jenkins and Travis CI. The common workflow of using such tools is as follows: developers write code in their IDEs, and then commit and push code into a Git repository, which will trigger a SAST tool to run an analysis on the code. To view the analysis result, developers need to login to a dashboard in their web browsers. Although such workflow is widely spread, it poses a usability issue for developers—they are taken outside their IDEs, where they write code. Recent studies report that developers want to see the result of these SAST tools directly in their IDEs [DWA20, CB16]. Our recent study [Lu21] asks the question how we can give developers access to cloud-based SAST tools directly through their IDEs, and if this improves developers' workflows. To answer these questions, we conducted a multiple-staged user study with software developers at Amazon Web Services.

First, we started by interviewing nine developers to understand their expectations of how cloud-based analyses should be triggered from an IDE, how analysis results should be

[1] Universität Paderborn, Germany linghui.luo@upb.de
[2] Universität Paderborn and Fraunhofer IEM, Germany eric.bodden@upb.de

displayed there, and what UX features they would like. While some participants expected to trigger the analysis manually by clicking a button in the IDE or when they build the project, the others would like the analysis to be automatically triggered. All participants told us they expected the IDE support to display the result automatically once the analysis is completed in the cloud. Which part of the result should be displayed in the IDE depends on developers' primary goals. If they want to improve the overall code quality, showing the entire project is desired. If they are implementing a new feature, only findings that are in the diffs should be displayed.

Guided by our findings from the interviews, we developed an IDE prototype for an existing cloud-based SAST tool—Amazon CodeGuru Reviewer [AW19], using its infrastructure for CI/CD. Users still need to push their code to a remote Git repository. Yet the IDE prototype allows users to interact with the cloud service directly inside their IDEs, i.e., they can request CodeGuru Reviewer to reanalyze the code and view the findings along with the code.

Once we had the IDE prototype, we presented it to the same group of developers we interviewed before to evaluate whether the design met their expectations. While these developers were satisfied with most features built in the prototype, they found existing mechanisms for CI/CD, e.g., code uploading via Git, were cumbersome in the IDE.

Finally, to test if the IDE solution was an improvement over the web-based solution, we conducted a within-subjects usability test with 32 developers. We found that using the IDE prototype developers performed code scans three times more often than using the web-based solution. Our measurements also show a promising reduction in time for fixing code. However, bringing the findings of the tool into the IDE did not necessarily improve developers' workflow as we learned from the after-session interviews. Specifically, they expected real-time feedback, quick validation of each fix, more seamless analysis of code and interactive ways to suggest rescan. We also noticed that some developers had limited understanding of the capabilities of SAST tools and confusions regarding the built-in workflow of such IDE integration. Future work should study how to make such IDE integration more intuitive for users.

2 Data Availability

The study was conducted during the first author's internship at Amazon Web Services. Due to company policies, the interview transcripts and IDE prototype are not available. The interview questions, the list of codes for analyzing the interview data, the tasks used in the usability test and survey questions are available at `https://github.com/linghuiluo/FSE21Study`.

Bibliography

[AW19] Amazon CodeGuru Reviewer, `https://aws.amazon.com/codeguru`.

[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.

[DWA20] Do, Lisa Nguyen Quang; Wright, James; Ali, Karim: Why do software developers use static analysis tools? a user-centered study of developer needs and motivations. IEEE Transactions on Software Engineering, 2020.

[Lu21] Luo, Linghui; Schäf, Martin; Sanchez, Daniel; Bodden, Eric: IDE support for cloud-based static analyses. In (Spinellis, Diomidis; Gousios, Georgios; Chechik, Marsha; Penta, Massimiliano Di, eds): ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021. ACM, pp. 1178–1189, 2021.

TaintBench: Automatic Real-World Malware Benchmarking of Android Taint Analyses

Linghui Luo[1] Felix Pauck[2] Goran Piskachev[3] Manuel Benz[4] Ivan Pashchenko[5] Martin Mory[6] Eric Bodden[7] Ben Hermann[8] Fabio Massacci[9]

Abstract: Due to the lack of established real-world benchmark suites for static taint analyses of Android applications, evaluations of these analyses are often restricted and hard to compare. Even in evaluations that do use real-world applications, details about the ground truth in those apps are rarely documented, which makes it difficult to compare and reproduce the results. Our recent study fills this gap. It first defines a set of sensible construction criteria for such a benchmark suite. It further proposes the TaintBench benchmark suite designed to fulfil these construction criteria. Along with the suite, this paper introduces the TaintBench framework, which allows tool-assisted benchmark suite construction, evaluation and inspection. Our experiments using TaintBench reveal new insights of popular Android taint analysis tools.

Keywords: Taint analysis; Benchmark; Real-world benchmark; Android malware

1 Summary

In our recent study [Lu22], we proposed the TaintBench framework and suite. To build the TaintBench framework, we mainly extended: (I) Jadx [T] to be able to *construct* and document benchmark apps, (II) ReproDroid [T] to automatically load and *evaluate* benchmarks, and (III) VSC [T] (Visual Studio Code) to *inspect* and understand benchmark results (cf. Figure 1). We employed the TaintBench framework to construct the TaintBench suite. In a thorough and long-lasting process we selected 39 malware apps to be included in the final suite and precisely documented 221 taint flows as a baseline ground truth. 186 of these taint

Fig. 1: Framework Overview

[1] Universität Paderborn, Germany linghui.luo@upb.de
[2] Universität Paderborn, Germany fpauck@mail.upb.de
[3] Fraunhofer IEM, Germany
[4] Universität Paderborn, Germany
[5] University of Trento, Italy
[6] Universität Paderborn, Germany
[7] Universität Paderborn and Fraunhofer IEM, Germany
[8] Technische Universität Dortmund, Germany
[9] University of Trento, Italy and Vrije Universiteit Amsterdam, Netherlands

flows are *expected* to be found by taint analysis tools. The remaining 35 taint flows are *not* expected (*unexpected*) to be found, however, these taint flows have been detected by analysis tools and were identified as false positives.

After constructing the suite, we employed the framework again to automatically re-evaluate two state-of-the-art analysis tools (AMANDROID [We14] and FLOWDROID [Ar14]). We considered two different versions of both tools. On the one hand, the same version as the one used in the REPRODROID study [PBW18], and on the other hand, the up-to-date version (when we refer to the latter, we mark it with an asterisk). For comparison purposes we also employed the DROIDBENCH micro benchmark suite in our experiments. Figure 2 shows the results of two experiments we conducted. For the first experiment (DB1 and TB1) we configured the analysis tools such that the default list of sources and sinks was used. Lists of sources and sinks with respect to the respective benchmark suite have been used during the second experiment (DB2 and TB2). The outcome already shows three findings: (1) The tools are less effective on real-world malware apps in TAINTBENCH—they have especially low recall, (2) The tools detect way more correct taint flows when using benchmark specific lists of sources and sinks, and (3) AMANDROID* and FLOWDROID* are less precise than their predecessors. These effects are harder to notice in the context of DROIDBENCH. Hence, we suggest to employ TAINTBENCH in future evaluations of Android taint analyses.

	DroidBench				TaintBench			
	Expected Cases: 163				Expected Cases: 186			
	Unexpected Cases: 41				Unexpected Cases: 35			
	DB1		DB2		TB1		TB2	
	TP	FP	TP	FP	TP	FP	TP	FP
Amandroid	80	12	95	15	2	2	31	19
Amandroid*	41	10	46	10	5	1	6	7
FlowDroid	89	12	96	12	26	0	84	10
FlowDroid*	87	12	95	12	16	1	41	14

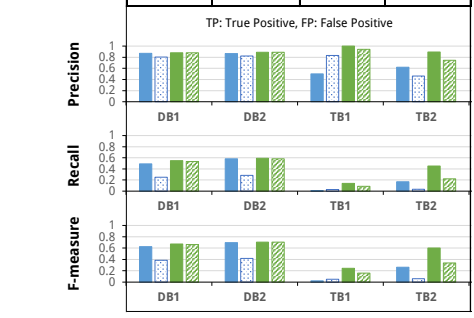

Fig. 2: Evaluation Results

2 Data Availability

The TAINTBENCH framework, the suite and all results of the study [Lu22] can be obtained from the project's website: `https://TaintBench.github.io` (also available on Zenodo: `https://zenodo.org/record/5734328`)

Bibliography

[Ar14] Arzt, Steven; Rasthofer, Siegfried; Fritz, Christian; Bodden, Eric; Bartel, Alexandre; Klein, Jacques; Traon, Yves Le; Octeau, Damien; McDaniel, Patrick D.: FlowDroid: precise

context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: Proceedings of PLDI, 2014. ACM, 2014.

[Lu22] Luo, Linghui; Pauck, Felix; Piskachev, Goran; Benz, Manuel; Pashchenko, Ivan; Mory, Martin; Bodden, Eric; Hermann, Ben; Massacci, Fabio: TaintBench: Automatic real-world malware benchmarking of Android taint analyses. Empir. Softw. Eng., 27(1):16, 2022.

[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.

[We14] Wei, Fengguo; Roy, Sankardas; Ou, Xinming; Robby: Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In: Proceedings of CCS, 2014. ACM, 2014.

[T] Tools, `https://github.com/skylot/jadx`, `https://FoelliX.github.io/ReproDroid`, `https://code.visualstudio.com` (last accessed: 01.12.2021).

Seamless Variability Management With the Virtual Platform (Summary)

Wardah Mahmood[1], Daniel Strüber[2], Thorsten Berger[3], Ralf Lämmel[4], Mukelabai Mukelabai[5]

Abstract: This extended abstract summarizes our paper with the same title published at the main track of the International Conference of Software Engineering (ICSE) 2021 [Ma21].

Keywords: software product lines; variability management; clone management; re-engineering; framework

1 Introduction

Customization is a general trend in software engineering, demanding systems that support variable stakeholder requirements. Two opposing strategies are commonly used to create variants: software clone&own and software configuration with an integrated platform. Organizations often start with the former, which is cheap, agile, and supports quick innovation, but does not scale. The latter scales by establishing an integrated platform that shares software assets between variants, but requires high up-front investments or risky migration processes.

Researchers have contributed frameworks for migrating from clone&own to a fully integrated platform. The frameworks, however, rely on heuristics and significant developer intervention, rendering them error-prone and ineffective. Additionally, the frameworks assume a binary mode of working; once an organization has switched to an integrated platform, it ceases to use clone&own. In reality, developers still employ clone&own owing to its convenience and availability. So, could we have a method that allows an easy transition or even combine the benefits of both strategies?

We propose a method and tool that supports a truly incremental development of variant-rich systems, exploiting a spectrum between both opposing strategies. We design, formalize, and prototype the variability-management framework *virtual platform*. It bridges the gap between clone&own and platform-oriented development by introducing governance levels

[1] Chalmers | University of Gothenburg, Sweden wardah@chalmers.se
[2] Radboud University, Netherlands d.strueber@cs.ru.nl
[3] Ruhr University Bochum, Germany and Chalmers | University of Gothenburg, Sweden thorsten.berger@rub.de
[4] University of Koblenz-Landau, Germany laemmel@uni-koblenz.de
[5] Chalmers | University of Gothenburg, Sweden muka@chalmers.se

between both; each adding a further level of detail and offering incremental benefits in return.

The virtual platform offers operators for engineering and evolving a system, relying on programming-language-independent conceptual structures representing software assets. The operators comprise traditional, asset-oriented operators and novel, feature-oriented operators for incrementally adopting concepts of an integrated platform. They record metadata that is exploited by other operators to support the transition. Specifically, we record metadata pertaining to clone traceability (where are the clones?) and feature-to-asset traceability (what is in the clones?). Among others, they eliminate expensive feature-location effort or the need to trace clones. The recorded metadata also enables two novel use-cases: feature cloning and feature change propagation.

Our evaluation simulates the evolution of a real-world, clone-based system, comprising four medium-sized variants. The results indicate that using virtual platform led to cost savings in terms of saved time spent on clone detection and feature location.

2 Data Availability

The source code of the Scala-based prototype of virtual platform is publicly available online [Vi21b]. Additionally, an online appendix with a technical report about our operators, additional examples, and evaluation data can be found online [Vi21a].

Literaturverzeichnis

[Ma21] Mahmood, Wardah; Strueber, Daniel; Berger, Thorsten; Laemmel, Ralf; Mukelabai, Mukelabai: Seamless Variability Management With the Virtual Platform. In: 43rd International Conference on Software Engineering (ICSE). 2021.

[Vi21a] Virtual Platform Online Appendix. `https://bitbucket.org/easelab/2021-icse-vponlineappendix`, 2021.

[Vi21b] Virtual Platform prototype. `https://bitbucket.org/easelab/workspace/projects/VP`, 2021.

Automated Process-Centric Quality Constraints Checking for Quality Assurance in Safety-critical Systems

Christoph Mayr-Dorn[1] Michael Vierhauser[2] Stefan Bichler[3] Felix Keplinger[4] Jane Cleland-Huang[5] Alexander Egyed[6] Thomas Mehofer[7]

Abstract: This abstract summarizes the work published as an ICSE 2021 research track paper [Ma21] available at https://doi.org/10.1109/ICSE43902.2021.00118. We propose an approach that, on the one hand, assists in checking compliance with traceability requirements but, on the other, hand allows engineers to temporarily deviate from the prescribed software engineering process. Through the observation of developer activities in the form of changes to engineering artifacts in tools such as Jira or Jama, we build up a representation of the ongoing process progress. This tracking in the background does not force the software developer to work only on activities as defined in a process description. At the same time, it enables us to provide timely feedback to the developer on whether tasks fulfill all QA criteria. This approach lifts the burden off QA engineers in manually checking QA constraints, often a time-consuming, tedious, and error-prone task where feedback reaches developers usually very late. We evaluate our approach by applying it to two different case studies; one open source community system and a safety-critical system in the air-traffic control domain. Results from the analysis show that trace links are often corrected or completed after the fact and thus timely and automated constraint checking support has significant potential on reducing rework.

Keywords: software engineering process; traceability; developer support; artifact change monitoring

1 Summary

In safety-critical systems, traceability requirements are often mandated by standards (e.g., ED-109A for air traffic management systems) as one form of quality assurance (QA) mechanism. Yet developers and QA engineers are overwhelmed by the complexity and extent of adhering to and evaluating QA constraints. Inspection of fine-grained constraints over engineering artifacts and their traces thus typically occurs only at the end of the engineering work when subsequent feedback often interrupts developers who have moved on to their next task. Such inspection during the engineering process is difficult as the current industry practice is to use semi-formal process descriptions that are not executable.

[1] Johannes Kepler University, Linz, Austria christoph.mayr-dorn@jku.at
[2] Johannes Kepler University, Linz, Austria michael.vierhauser@jku.at
[3] Johannes Kepler University, Linz, Austria
[4] Johannes Kepler University, Linz, Austria
[5] University of Notre Dame, Notre Dame, USA JaneHuang@nd.edu
[6] Johannes Kepler University, Linz, Austria alexander.egyed@jku.at
[7] Frequentis AG, Vienna, Austria thomas.mehofer@frequentis.com

To this end, we propose our Passive **Pro**cess Execution and Quality **Con**straint Support Framework **ProCon**. Two key aspects that characterize our framework are (a) the integrated handling of explicitly distinct processes and constraints, and (b) the tracking of engineering progress achieved through linking process specification to software engineering artifact details. The main difference to contemporary process support solutions [Gr02] is the framework's ability to track the process in the background, based on the software engineers' activities performed in the tools they are using in their daily work rather than requiring engineers to interact with a process engine. Hence, engineers are free to deviate from the process, but still receive guidance even in the presence of a deviation.

Deviations from the process are tracked via process and quality constraints evaluated in the Drools rule engine. These constraints, and their respective evaluation results, are treated as first-class citizens in the software engineering process for determining process progress and serving as explicit feedback to the developers via the framework's web-based user interface.

We implemented our framework as a Java prototype connecting to Jama and Jira as representative engineering tools that enable managing of requirements, design documents, work items, test cases, etc as well as the traces amongst these artifacts. We evaluated our approach by extracting the change history of the artifacts involved in processes from the open-source UAV project Dronology [CHVB18] and our industry partner Frequentis, a supplier of air traffic management systems. For the industry use case, we tracked the process deviations and constraint violations during the replay of more than 14,000 artifact changes for 109 process instances. We found **temporary** deviations and violations in 20% of processes, concluding that automated developer feedback is highly desirable.

1.1 Data Availability

The prototype and data used in the original paper are available at Figshare `https://doi.org/10.6084/m9.figshare.12840053`.

Literaturverzeichnis

[CHVB18] Cleland-Huang, Jane; Vierhauser, Michael; Bayley, Sean: Dronology: An Incubator for Cyber-Physical Systems Research. In: Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results. ICSE-NIER '18. ACM, S. 109–112, 2018.

[Gr02] Gruhn, Volker: Process-centered software engineering environments, a brief history and future challenges. Annals of Software Engineering, 14(1-4):363–382, 2002.

[Ma21] Mayr-Dorn, Christoph; Vierhauser, Michael; Bichler, Stefan; Keplinger, Felix; Cleland-Huang, Jane; Egyed, Alexander; Mehofer, Thomas: Supporting Quality Assurance with Automated Process-Centric Quality Constraints Checking. In: 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021. IEEE, S. 1298–1310, 2021.

How Gamification Affects Software Developers: Cautionary Evidence from a Natural Experiment on GitHub

Lukas Moldon[1], Markus Strohmaier[1], Johannes Wachs[2]

Abstract: We examine how the behavior of software developers changes in response to removing gamification elements from GitHub, an online platform for collaborative programming. We find that the unannounced removal of daily activity streak counters from the user profile pages was followed by significant changes in behavior. Long-running streaks of activity were abandoned and became less common. Weekend activity decreased and days in which developers made a single contribution became less common. Synchronization of streaking behavior in the platform's social network also decreased, suggesting that gamification is a powerful channel for social influence. Software developers that were publicly pursuing a goal to make contributions for 100 days in a row abandon this quest following the removal of the streak counter. Our findings provide evidence for the significant impact of gamification on the behavior of developers. They urge caution: gamification can steer the behavior in unexpected and unwanted directions.

Keywords: gamification; behavior; software engineering, natural experiment; GitHub

Summary of Research

In our work published at ICSE 2021 [MSW21], we exploited an unexpected user interface design change on GitHub, the largest platform for collaborative software development, to study how gamification can influence user behavior in the context of software development. From one day to the next in May 2016 streak counters, measuring both current and all-time best counts of how many days in a row a user had made a contribution on GitHub, were removed from all user profiles without warning. This presented a unique opportunity to compare user behavior before and after the change, from which we can estimate the effect of streak counters on user behavior.

A large and growing literature indicates that gamification works: people respond to badges, points, and rankings in digital settings [HKS14]. In some contexts, achievements on gamified platforms have real labor market value, as they are thought to be credible signals of expertise and know-how [Ku21]. Yet although gamification is increasingly deployed on online platforms to steer individuals toward certain kinds of activity, there relatively little previous work on the potential negative effects of this phenomenon, especially in software development. What is known is that gamification can incentivize overwork and even dishonesty [WO14].

[1] RWTH Aachen University, Germany lukas.moldon@rwth-aachen.de markus.strohmaier@cssh.rwth-aachen.de
[2] Vienna University of Economics and Business, Austria johannes.wachs@wu.ac.at

We found evidence that the counters were influencing user behavior in a variety of ways. Users were significantly more likely to maintain long streaks of uninterrupted activity when these streaks were reported on their profiles. After the counters were removed, very long streaks of months of uninterrupted activity became much more rare. There was even a small but significant drop of weekend activity on the whole. We also found a significant drop-out rate among developers who had adopted a 100-days of code challenge, immediately following the change.

An additional observation suggests that individuals did care about their streaks for the sake of the signal: prior to the design change it was common for developers on long streaks to have many single contribution days. We interpret their statistical overrepresentation as evidence that developers were making a minimal effort just to keep their streaks alive. This kind of behavior is even more troublesome in light of our last finding: that developers tended to imitate the streaking behavior of their neighbors in GitHub's social network when counters were visible. This suggests that the behavioral consequences of gamification extend beyond the individual, effecting groups of people. We concluded our study with a reflection on these results and a recommendation that platform designers consider the use of gamification carefully.

Data availability

This work is based on the GHTorrent database[3] from June 2019. We explain the data processing in our GitHub repository[4] and provide all scripts to recreate our database. Processed data is available on Zenodo[5]. A video summary is available on YouTube[6].

Literaturverzeichnis

[HKS14] Hamari, Juho; Koivisto, Jonna; Sarsa, Harri: Does gamification work?–a literature review of empirical studies on gamification. In: 2014 47th Hawaii international conference on system sciences. Ieee, S. 3025–3034, 2014.

[Ku21] Kuttal, Sandeep Kaur; Chen, Xiaofan; Wang, Zhendong; Balali, Sogol; Sarma, Anita: Visual Resume: Exploring developers' online contributions for hiring. Information and Software Technology, S. 106633, 2021.

[MSW21] Moldon, Lukas; Strohmaier, Markus; Wachs, Johannes: How gamification affects software developers: Cautionary evidence from a natural experiment on github. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, S. 549–561, 2021.

[WO14] Welsh, David T; Ordóñez, Lisa D: The dark side of consecutive high performance goals: Linking goal setting, depletion, and unethical behavior. Organizational Behavior and Human Decision Processes, 123(2):79–89, 2014.

[3] https://ghtorrent.org/downloads.html
[4] https://github.com/lukasmoldon/GHStreaksThesis
[5] https://zenodo.org/record/4710603#.YbSOWr3MJD9
[6] https://www.youtube.com/watch?v=lR8DpCNQNBU

Data-Driven Design and Evaluation of SMT Meta-Solving Strategies

Malte Mues[1] Falk Howar[1,2]

Abstract: The 36th IEEE/ACM International Conference on Automated Software Engineering (2021) accepted the paper 'Data-Driven Design and Evaluation of SMT Meta-Solving Strategies: Balancing Performance, Accuracy, and Cost' [MH21a] and selected it for an ACM SIGSOFT Distinguished Paper Award. The paper presents four generally applicable patterns for the combination of multiple SMT decision procedures in a meta-solving strategy and demonstrates how a meta-solving strategy for string constraints can be developed in a data-driven approach based on these patterns: The paper cleans up and merges existing collections of SMT benchmarks in string theory solving to evaluate and compare derived meta-solving strategies. Notably, we can demonstrate on the available data that commonly used strategies as earliest returning SMT solver do not always return the most reliable result if all available SMT solvers are combined. Instead, cross-checking strategies work slightly better at moderate overhead.

Keywords: SMT Solving; Formal Methods; Software Verification

1 Integration Patterns for Formal Methods

Advances in formal methods are often implemented in research tools and libraries, which are then used in other analyses or commercial applications. The most prominent example of the past decade are constraint solvers that are used in many applications today. While the underlying decision procedures are usually sound, implementations optimize for different aspects of a problem and — just like other software — can contain bugs. One strategy for making formal methods robust in practice is redundancy: i.e., using multiple analyses of a similar type. Here, again, constraint solvers can serve as an example: so-called portfolios of solvers are used to optimize the number of conclusive verdicts or response times by analyzing a problem with multiple solvers. The paper [MH21a] identifies four basic patterns for integrating different formal analyses that have been proposed in the literature (c.f. [WHDM09, HJM19, Ri17, MH21b]).

The four patterns, sketched in Figure 1, are briefly described below. The *Majority Vote* (middle right) integrates different constraint solvers (or multiple instances of the same solver) A to N by analyzing a problem instance with all solvers. A majority vote is computed as a final verdict. The *Earliest Verdict* (middle left) integrates constraint solvers A to N by analyzing a problem instance with all solvers. The first obtained verdict is used as the final

[1] TU Dortmund University, 44227 Dortmund, Deutschland {malte.mues,falk.howar}@tu-dortmund.de
[2] Fraunhofer ISST, 44227 Dortmund, Deutschland

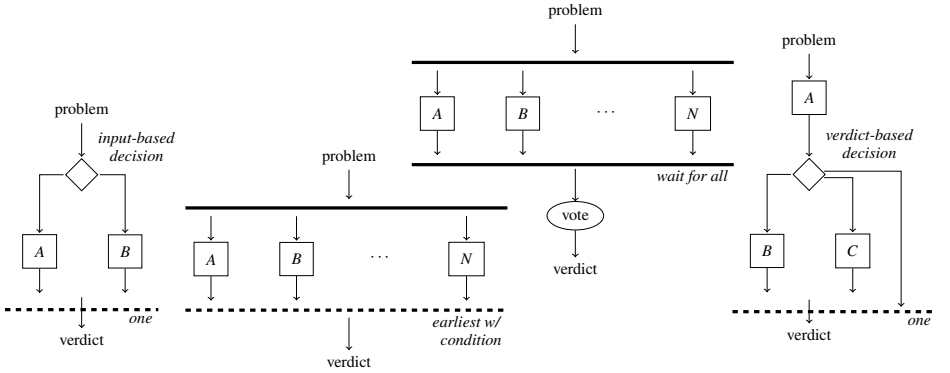

Fig. 1: Four Basic integration patterns for formal anayses [MH21a].

verdict. The *Verdict-based Second Attempt* (right) integrates multiple constraint solvers by first calling solver *A* and then deciding on a next step on the basis of the obtained verdict. Potential next steps are invoking another solver on the original problem or using another solver or tool to validate the result computed by *A*. The *Feature-based Attempt* (left) integrates multiple solvers by selecting one constraint solver for a problem instance based on features of the problem instance. In the paper, strengths and drawbacks of the patterns are discussed and performance of patterns is analyzed for integrating string solvers.

2 Data Availability

The reproduction package for the paper with all experimental data and scripts for computing results is publicly available at `https://zenodo.org/record/5102174`.

Bibliography

[HJM19] Howar, Falk; Jabbour, Fadi; Mues, Malte: JConstraints: a library for working with logic expressions in Java. In: Models, Mindsets, Meta: The What, the How, and the Why Not?, pp. 310–325. Springer, 2019.

[MH21a] Mues, Malte; Howar, Falk: Data-Driven Design and Evaluation of SMT Meta-Solving Strategies: Balancing Performance, Accuracy, and Cost. In: Proc. of ASE, 2021. ACM (to appear), 2021.

[MH21b] Mues, Malte; Howar, Falk: JDart: Portfolio solving, breadth-first search and smt-lib strings (competition contribution). Proc. of TACAS 2021, 12652:448–452, 2021.

[Ri17] Riener, Heinz; Haedicke, Finn; Frehse, Stefan; Soeken, Mathias; Große, Daniel; Drechsler, Rolf; Fey, Goerschwin: metaSMT: focus on your application and not on solver integration. STTT, 19(5):605–621, 2017.

[WHDM09] Wintersteiger, Christoph M; Hamadi, Youssef; De Moura, Leonardo: A concurrent portfolio approach to SMT solving. In: Proc. of CAV 2009. Springer, pp. 715–720, 2009.

Identifying Software Performance Changes Across Variants and Versions

Stefan Mühlbauer[1], Sven Apel[2], Norbert Siegmund[3]

Abstract: Performance changes of configurable software systems can occur and persist throughout their lifetime. Finding optimal configurations and configuration options that influence performance is already difficult, but in the light of software evolution, configuration-dependent performance changes may lurk in a potentially large number of different versions of the system. Building on previous work, we combine two perspectives—variability and time—and devise an approach to identify configuration-dependent performance changes *retrospectively* across the software variants and versions of a software system. In a nutshell, we iteratively sample pairs of configurations and versions and measure the respective performance, which we use to actively learn a model that estimates how likely a commit introduces a performance change. For such commits, we infer the configuration options that best explain observed performance changes. Pursuing a search strategy to measure selectively and incrementally further pairs, we increase the accuracy of identified change points related to configuration options and interactions. Our evaluation with both real-world software systems and synthesized data demonstrates that we can pinpoint performance shifts to individual configuration options and commits with high accuracy and at scale.

Keywords: Software Performance; Configurable Software Systems; Software Evolution

Modern software systems often provide configuration options to customize their functionality and non-functional characteristics, such as energy consumption and performance. Determining the influence of individual configuration options and their interactions on performance (henceforth called performance influences) follows a two-step process. First, a set of configurations is selected and then measured via a benchmark or an application-specific workload. Second, a machine learning technique, such as linear regression or classification and regression trees, is applied to learn a performance model using the measurements as a training set. With this model, we can estimate the performance of unseen configurations.

However, software is not a static entity, and as the code base changes so does performance. If the performance influences change, prediction models from older versions make inaccurate estimations for newer versions. Maintaining accurate prediction models throughout a software's lifetime vastly increases the cost of measurement since we would blindly select configurations throughout the version history of a system to find the relevant change points.

By contrast, information about which performance influences change at what revision could guide practitioners to re-learn models only upon such an occurrence or guide future testing efforts. We address the problem of detecting change points in space (configuration dimension)

[1] Universität Leipzig, Institut für Informatik, Leipzig, Deutschland, muehlbauer@informatik.uni-leipzig.de
[2] Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Deutschland, apel@cs.uni-saarland.de
[3] Universität Leipzig, Institut für Informatik, Leipzig, Deutschland, norbert.siegmund@informatik.uni-leipzig.de

and time (version dimension) and devise a novel approach that *retrospectively* identifies shifts in performance influences across a software system's development history [MAS20].

In previous work, we presented how one can identify performance shifts in a series of performance measurements of a single software configuration [MAS19]. Since configuration-specific performance changes often emerge from changes in performance influences, it is key to identify such causative options. To this end, we pinpoint performance change not only to a specific commit, but also a set of responsible configuration options. Starting from performance measurements of a small sample of random configurations and revisions, our active learning and sampling strategy first estimates the temporal location of likely change points (candidate commits). If a performance shift is observed at a candidate commit for multiple configurations, the shift is marked for further analysis (candidate shift). Second, we attribute each candidate shift to one or more configuration options based on the configurations for which it manifests. Last, to increase the reliability of these estimations, we iteratively add new measurements to the training set and repeat the previous steps until the set of identified change points does not change over a couple of iterations

The main challenge with this approach is the combinatorial complexity of the configuration-and-revision space. To wisely select configurations and revisions, we balance the budget of measurements between two objectives, *exploration* and *exploitation*. To capture undetected performance shifts, we select configurations and revisions randomly (exploration). We dedicate the remaining budget to re-evaluating the approach's candidate solutions in each iteration (exploitation). Throughout the iterations, we shift the ratio of measurements towards exploitation.

We show for three real-world software systems that our approach can precisely pinpoint performance changes to a single commit and set of configuration options. Further experiments with synthetic data demonstrate its scalability regarding the number of commits of a version history, the number of configuration options, and the degree of option interactions.

Data Availability We provide all measurement data via the original companion web site [4].

Literatur

[MAS19] Mühlbauer, S.; Apel, S.; Siegmund, N.: Accurate Modeling of Performance Histories for Evolving Software Systems. In: Proceedings of the International Conference on Automated Software Engineering (ASE). IEEE, S. 640–652, Nov. 2019.

[MAS20] Mühlbauer, S.; Apel, S.; Siegmund, N.: Identifying Software Performance Changes across Variants and Versions. In: Proceedings of the International Conference on Automated Software Engineering (ASE). ACM, S. 611–622, 2020.

[4] https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/AI-4-SE/Changepoints-Across-Variants-And-Versions

Challenges in the Development of Mobile Online Services in the Automotive Industry - A Case Study

Nils Prenner[1], Jil Klünder[2], Michael Nolting[3], Oliver Sniehotta[4], Kurt Schneider[5]

Abstract:

Automotive companies need to develop new innovations fast in order to stay ahead of their competitors. New sensor technologies and the connection between cars and smartphones open the development of different services, like the analysis of driving behavior. These services are called mobile online services and are a growing field in the automotive sector. We want to understand the challenges that slow down the development of mobile online services in automotive companies. Therefore, we conducted a case study with an interview study in a project that develops services to manage vehicle fleets. Our results show that the company considers mobile online services rather as a by-product and is still focused on the manufacturing of cars.

This summary refers to the paper *Challenges in the Development of Mobile Online Services in the Automotive Industry - A Case Study* [Pr21]. This paper was published as full research paper in the proceedings of the Joint International Conference on Software and System Processes and International Conference on Global Software Engineering 2021.

Keywords: Automotive Industry; Mobile Online Services; Agile Development; Case Study

1 Introduction

Most aspects of cars a steered by software and more and more sensors are integrated into today's cars. This enables the development of helpful services, e.g. for maintenance. These services are called mobile online services and define more and more the real value of a car. Automotive companies are under much pressure to develop fast, new innovative features with a high quality. Further, to follow new technologies and customer demands, automotive companies have to be flexible. However, automotive companies are often slow, have a rigid structure, and are reluctant to change. To understand the challenges of automotive companies when they develop mobile online services, we conducted a case study at Volkswagen

[1] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, nils.prenner@inf.uni-hannover.de
[2] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, jil.kluender@inf.uni-hannover.de
[3] Volkswagen Nutzfahrzeuge, Mecklenheidestraße 74, 30419 Hannover, michael.nolting@volkswagen.de
[4] Volkswagen Nutzfahrzeuge, Mecklenheidestraße 74, 30419 Hannover, oliver.sniehotta@volkswagen.de
[5] Leibniz Universität Hannover, Fachgebiet Software Engineering, Welfengarten 1, 30167 Hannover, kurt.schneider@inf.uni-hannover.de

Commercial Vehicles (VWN). The project team we describe in our publication creates a fleet management system for medium-sized companies. The system enables companies to easily connect, maintain, and monitor the vehicles of their fleet. We conducted interviews with six members of the project team.

2 Results

The development of mobile online services depends strongly on the needs of their users. Design-driven approaches help to create applications that are aligned with users' needs. However, our results show that the project has difficulties applying a more design-driven approach. Often the problems and the domain of the users are insufficiently understood. A lack of staff exacerbates the situation further because the employees cannot focus on one problem and do not have the time to get a deeper understanding. Next to a development aligned with users' needs, the project wants a fast development but the development is often deferred. The discussion of the requirements takes a lot of time and a lot of different parties have to be aligned. Further, the dealings with the projects of the surrounding systems frequently interrupt the development. Also, an unclear distribution of tasks and responsibilities let the project struggle.

3 Conclusion

Automotive companies have to consider the development of mobile online services and the manufacturing of cars more as one entity. At the moment both are mostly separated and the development of these services only follows the manufacturing process. This leads to services that represent more the technological possibilities of the car, but not what the users really need. Automotive companies should invest more time in the comprehension of what their customers really need and which services they require. Consequently, the development of mobile online services and the manufacturing of cars should go hand in hand.

4 Data Availability

Due to the privacy concerns of our interviewees and the company's internal regulations, we can not make the interview transcripts publicly available.

Bibliography

[Pr21] Prenner, Nils; Klünder, Jil; Nolting, Michael; Sniehotta, Oliver; Schneider, Kurt: Challenges in the Development of Mobile Online Services in the Automotive Industry - A Case Study. In: 2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE). pp. 22–32, 2021.

Lars Grunske, Janet Siegmund, Andreas Vogelsang (Hrsg.): SE 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 81

CiFi: Versatile Analysis of Class and Field Immutability

Tobias Roth, Dominik Helm, Michael Reif, Mira Mezini[1]

Abstract: This paper was accepted in 2021 at the 36th IEEE/ACM International Conference on Automated Software Engineering and proposes a model for immutability analysis. Reasoning about immutability is important for preventing bugs, e.g., in multi-threaded software. Static analysis to infer immutability properties has mostly focused on individual objects and references. Reasoning about fields and entire classes, while significantly simpler, has gained less attention. A consistently used terminology is missing, which makes it difficult to implement analyses that rely on immutability information. We propose a model for class and field immutability that unifies terminology for immutability flavors considered by previous work and covers new levels of immutability to handle lazy initialization and immutability dependent on generic type parameters. Using the OPAL static analysis framework, we implement CiFi, a set of modular, collaborating analyses for different flavors of immutability, inferring the properties defined in our model. We propose a benchmark of representative test cases for class and field immutability. We use the benchmark to showcase CiFi's precision and recall in comparison to state of the art and use CiFi to study the prevalence of immutability in real-world libraries, showcasing the practical quality and relevance of our model.

Keywords: class and field immutability; static analysis; lattice; Java

1 Summary

Immutability brings important guarantees and is, e.g., recommended for secure coding [Or20]. Intuitively, immutability means that a program element is unchangeable or not changed after its creation [Po13]. While there are multiple flavors and levels of immutability, current approaches only focus on a restricted set of immutability levels and additionally cannot handle lazy initialization and immutability in combination with generic types properly. Furthermore, there is not only no common terminology yet for different immutability levels, but also existing terminology is used inconsistently [Co17, Po00, NPN12, Po13].

To solve this problem, we propose a unified terminology and present it in our lattice model that focuses on class and field immutability. Analyzing class immutability is much simpler than analyzing object immutability while remaining sound [Co16, Co17]. The model is divided into four different lattices for *field assignability*, *field immutability*, *class immutability*, and *type immutability*. Because current approaches cannot handle lazy initialization of fields and immutability in combination with Java generics precisely, we introduced the novel immutability levels *(unsafe) lazy initialization* and *dependent immutability*. (Unsafe) lazy

[1] Technische Universität Darmstadt, FG Softwaretechnik, Germany
{roth,helm,mezini}@cs.tu-darmstadt.de,mi.reif.mr@gmail.com

initialization describes whether all reads of a field always return the same value. Dependent immutability describes cases where the immutability of a type, class or field depends on the concretization of a generic type. This model allows for modular immutability analyses for field assignability and field, class, and type immutability.

Based on our model, we implemented CiFi that encompasses four different analyses for field assignability, field, class, and type immutability in the Java Bytecode analysis framework OPAL [He20]. OPAL's blackboard architecture supports the composition of multiple decoupled interdependent analyses. In our evaluation, we show the expressiveness of our model and challenge CiFi with our CiFi-Benchmark for field and class immutability, which we created based on our immutability research. We show that CiFi handles most of the test-cases precisely and over-approximates remaining corner-cases soundly. The results of analyzing several real-world libraries with CiFi show the applicability and relevance of our immutability model in the real world. Furthermore, CiFi outperforms the state of the art in class- and field-immutability enforcement, Glacier [Co17].

2 Data Availability

CiFi: `https://github.com/opalj/opal/tree/feature/classFieldImmutability`
CiFi-Benchmark: `https://github.com/opalj/CiFi-Benchmark`
Artifact: `https://doi.org/10.5281/zenodo.5227231`

Bibliography

[Co16] Coblenz, Michael; Sunshine, Joshua; Aldrich, Jonathan; Myers, Brad; Weber, Sam; Shull, Forrest: Exploring language support for immutability. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). pp. 736–747, 2016.

[Co17] Coblenz, Michael; Nelson, Whitney; Aldrich, Jonathan; Myers, Brad; Sunshine, Joshua: Glacier: Transitive class immutability for Java. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, pp. 496–506, 2017.

[He20] Helm, Dominik; Kübler, Florian; Reif, Michael; Eichberg, Michael; Mezini, Mira: Modular collaborative program analysis in OPAL. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE'20, pp. 184–196, 2020.

[NPN12] Nelson, Stephen; Pearce, David J; Noble, James: Profiling field initialisation in Java. In: International Conference on Runtime Verification. Springer, pp. 292–307, 2012.

[Or20] Oracle: Secure Coding Guidelines for Java SE. https://www.oracle.com/java/technologies/javase/seccodeguide.html, September 2020.

[Po00] Porat, Sara; Biberstein, Marina; Koved, Larry; Mendelson, Bilha: Automatic detection of immutable fields in Java. In: CASCON. p. 10, 2000.

[Po13] Potanin, Alex; Östlund, Johan; Zibin, Yoav; Ernst, Michael D.: Immutability. In: Aliasing in Object-Oriented Programming, pp. 233–269. Springer-Verlag, Berlin, Heidelberg, 2013.

Scalable N-Way Model Matching Using Multi-Dimensional Search Trees – Summary

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

Abstract: In this work, we report about recent research on n-way model matching, originally published at the International Conference on Model Driven Engineering Languages and Systems (MODELS) 2021 [Sc21]. Model matching algorithms are used to identify common elements in input models, which is a fundamental precondition for many software engineering tasks, such as merging software variants or views. If there are multiple input models, an n-way matching algorithm that simultaneously processes all models typically produces better results than the sequential application of two-way matching algorithms. However, existing algorithms for n-way matching do not scale well, as the computational effort grows fast in the number of models and their size. We propose a scalable n-way model matching algorithm, which uses multi-dimensional search trees for efficiently finding suitable match candidates through range queries. We implemented our generic algorithm named RaQuN (**Ra**nge **Qu**eries on **N** input models) in Java, and empirically evaluate the matching quality and runtime performance on several datasets of different origin and model type. Compared to the state-of-the-art, our experimental results show a performance improvement by an order of magnitude, while delivering matching results of better quality.

Keywords: Model-driven engineering; n-way model matching; clone-and-own development; software product lines; multi-view integration; variability mining

1 Summary

Matching algorithms are an essential requirement for detecting common parts of development artifacts in many software engineering activities. In domains where model-driven development has been adopted in practice, such as automotive, avionics, and automation engineering, numerous model variants emerge from cloning existing models. Integrating such autonomous variants into a centrally managed software product line in extractive software product-line engineering requires to detect similarities and differences between them, which in turn requires to match the corresponding model elements of the variants.

Currently, almost all existing matching algorithms can only process *two* development artifacts, whereas it is typically required to identify corresponding elements in *multiple* (i.e., $n > 2$) input models. A few approaches calculate an n-way matching by repeated

[1] Humboldt-University of 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] Humboldt-University of Berlin, Germany, timo.kehrer@hu-berlin.de

two-way matching of the input artifacts. In each step, the resulting two-way correspondences are simply linked together. However, sequential two-way matching of models may yield sub-optimal results because not all input artifacts are considered at the same time and order in which input models are processed influences the quality of the matching.

The only matching approach which simultaneously processes n input models is a heuristic algorithm called NwM by Rubin and Chechik. Yet, we faced scalability problems when applying NwM to models of realistic size, comprising thousands of elements. The most likely reason for this is the huge number of model element comparisons, which often leads to performance problems even in the case of few input models if these models are large.

We propose RaQuN (**Ra**nge **Qu**eries on **N** input models), a generic, heuristic n-way model matching algorithm. The key idea behind RaQuN is to map the elements of all input models to points in a numerical vector space. RaQuN embeds a multi-dimensional search tree into this vector space to efficiently find nearest neighbors of elements. By comparing an element only with its nearest neighbors serving as match candidates, RaQuN can reduce the number of required comparisons considerably.

For our empirical assessment, we use datasets from different domains and development scenarios. Next to academic and synthetic models, we investigate variants generated from model-based product lines, and reverse-engineered models from clone-and-own development. Moreover, we are the first to provide a thorough investigation of n-way model matching on large-scale subjects and a real-world clone-and-own subject. Our evaluation shows that RaQuN reduces the number of required comparisons by more than 90% for most experimental subjects, making it possible to match models of realistic size simultaneously.

2 Data Availability

The original publication is accessible under the DOI 10.1109/MODELS50736.2021.00010. A preprint is also available online `https://github.com/SoftVarE-Group/Papers/blob/master/2021/2021-MODELS-Schultheiss.pdf`. Our artifact is available on Github (`https://github.com/AlexanderSchultheiss/RaQuN`) and Zenodo (DOI: 10.5281/zenodo.5150388).

3 Acknowledgments

We thank Lars Grunske for his ongoing support and initiative towards our original publication.

Bibliography

[Sc21] Schultheiß, Alexander; Bittner, Paul Maximilian; Grunske, Lars; Thüm, Thomas; Kehrer, Timo: Scalable N-Way Model Matching Using Multi-Dimensional Search Trees. In: Proc. Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS). IEEE, Washington, DC, USA, pp. 1–12, October 2021.

Exploring Web Search Engines to Find Architectural Knowledge

Mohamed Soliman[1] Marion Wiese[2] Yikun Li[3] Matthias Riebisch[4] Paris Avgeriou[5]

Abstract:

Software engineers need relevant and up-to-date architectural knowledge (AK), in order to make well-founded design decisions. However, finding such AK is quite challenging. One pragmatic approach is to search for AK on the web using traditional search engines (e.g. Google); this is common practice among software engineers. Still, we know very little about what AK is retrieved, from where, and how useful it is. In this paper, we conduct an empirical study with 53 software engineers, who used Google to make design decisions using the Attribute-Driven-Design method. Based on how the subjects assessed the nature and relevance of the retrieved results, we determined how effective web search engines are to find relevant architectural information. Moreover, we identified the different sources of AK on the web and their associated AK concepts.

The full paper of this extended Abstract has been published in [So21].

Keywords: software architecture; architecture knowledge; search approach; Google

1 Summary

Architectural knowledge (AK) is crucial for software engineers to make architectural design decisions [BCK12]. However, finding architectural knowledge (AK) is a challenging task, because AK resides in multiple heterogeneous *AK sources*, such as developer communities. Moreover, each source of AK contains different *AK concepts* (e.g. design decisions). A pragmatic and common way to search for AK from different sources is to use web search engines (e.g. Google). However, there is little to no empirical evidence about which AK sources and AK concepts can actually be found by web search engines. Our main **goal** is to *explore which AK sources and AK concepts are retrieved by web search engines, and to gauge the effectiveness of web search engines to find relevant AK during the architectural design process*. To this end, we conducted an empirical study with 53 software engineers. The subjects used the most popular web search engine (i.e. Google) to find relevant AK concepts when conducting the steps of the Attribute-Driven Design (ADD) method [KC16].

[1] University of Groningen, The Netherlands m.a.m.soliman@rug.nl
[2] Universität Hamburg, SWK, Hamburg, Deutschland marion.wiese@uni-hamburg.de
[3] University of Groningen, The Netherlands yikun.li@rug.nl
[4] Universität Hamburg, SWK, Hamburg, Deutschland matthias.riebisch@uni-hamburg.de
[5] University of Groningen, The Netherlands p.avgeriou@rug.nl

We provide below an overview on the results of each research question:

(RQ1) *Which AK can web search engines support to find?* The results suggest that google retrieve the following AK sources: blogs (39%), technology vendor documentations (23%), scientific contents (13%), and forums like Stack Overflow (7%), source code repositories like Github (4.5%) and knowledge repositories (4%). Furthermore, issue tracking systems are not retrieved at all by Google. On the other hand, the results showed that blogs, forums, knowledge repositories, and scientific contents have the most AK concepts.

(RQ2) *Which AK concepts co-occur on the web?* Our results show that benefits and drawbacks have the highest correlation (0.651) to co-occur with each other. It is common to find web pages with benefits and no drawbacks, while it is rare to find web pages with drawbacks and no benefits. Moreover, lists of alternative solutions, are usually (0.355) accompanied with a comparison between them regarding their benefits and drawbacks.

(RQ3) *How well do web search engines support software engineers in following the ADD steps?* Our results show that Google supports software engineers to perform the ADD step "Select design concepts" better than the other steps like "Instantiate architecture elements". Moreover, we found that Google returns many distantly relevant web pages (e.g. general concepts on components design) to support the "Instantiate architecture elements" step.

(RQ4) *Which AK concepts make web pages more relevant for design decisions?* Our results show that web pages that contain solution benefits, solution drawbacks and made design decisions are most relevant for finding AK. Web pages with solution description' and alternatives have a lower probability to be highly relevant.

2 Data Availability

We provide our corpus online (`github.com/m-a-m-s/ICSA2021`) to facilitate replicating the study. Moreover, the corpus contains empirically classified and evaluated web pages and their respective AK sources and AK concepts.

Literatur

[BCK12] Bass, L.; Clements, P.; Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional, 2012, ISBN: 0321815734, 9780321815736.

[KC16] Kazman, R.; Cervantes, H.: Designing Software Architectures: A Practical Approach. Addison-Wesley Professional, 2016.

[So21] Soliman, M.; Wiese, M.; Li, Y.; Riebisch, M.; Avgeriou, P.: Exploring Web Search Engines to Find Architectural Knowledge. In: 2021 IEEE 18th International Conference on Software Architecture (ICSA). IEEE, S. 162–172, März 2021, ISBN: 978-1-7281-6260-7.

Exploring Architectural Design Decisions in Industry 4.0: A Literature Review and Taxonomy

Tarik Terzimehić[1], Kirill Dorofeev[2], Sebastian Voss[3]

Abstract: The full version of this paper is published at the Foundation Track of ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2021. Architectural design decisions, such as service deployment and composition, plant layout synthesis, or production planning, are an indispensable and overarching part of an industrial manufacturing system design. In the fourth industrial revolution (Industry 4.0), frequent production changes trigger their synthesis, and preferably optimization. Yet, knowledge on architecture synthesis and optimization has been scattered around other domains, such as generic software engineering. We take a step towards synthesizing current knowledge on architectural design decisions in Industry 4.0. We developed a taxonomy describing architectural models, design decisions, and optimization possibilities. The developed taxonomy serves as a guideline for comparing different possibilities (e.g., application of different optimization algorithms) and selecting appropriate ones for a given context. Furthermore, we reviewed and mapped 30 relevant research works to the taxonomy, identifying research trends and gaps. We discuss interesting, and yet uncovered topics that emerged from our review.

Keywords: architecture synthesis; optimization; taxonomy; design space exploration; model-based development; Industry 4.0

1 Summary

The design of industrial manufacturing system comprises different architectural design decisions (ADDs), such as the deployment of different software components (SWCs) to hardware components (HWCs), production services composition, plant layout determination, or production planning (i.e., the decision of which station should perform which production step) [KKZ14]. Contrary to current manufacturing systems, the fourth industrial revolution, known as Industry 4.0 (I40), envisions an unplanned changeability of optimized production processes without significant downtime [WCB17]. Different I40 scenarios, such as lot-size-one production and plug-and-produce, frequently influence and trigger the synthesis of different ADDs, making the ADDs synthesis *an indispensable* and *overarching* part of the I40.

This increased importance of architecture synthesis and optimization (ASO) in I40, and numerous established ASO solutions from other domains [Al13], drive *the need for an*

[1] fortiss – Research Institute of the Free State of Bavaria, Munich, Germany terzimehic@fortiss.org
[2] fortiss – Research Institute of the Free State of Bavaria, Munich, Germany dorofeev@fortiss.org
[3] Aachen University of Applied Sciences, Aachen, Germany s.voss@fh-aachen.de

overview of ASO. Such an overview would list and clarify existing problems in the domain, and at the same time, present fitting approaches and alternatives according to specific user's needs.

We take a step towards synthesizing current knowledge on ASO. We differ from existing works (e.g., [Al13]) by providing a review and analysis of the ASO in the I40 context – considering domain-specific modeling approaches, design decisions, use cases, etc. We characterize the state of the art of ASO in I40 by reviewing and analyzing 30 research papers.

We develop a taxonomy that characterizes architectural models, ADDs, and optimization possibilities. The resulting taxonomy serves as a guideline for a comparison of different possibilities (e.g., application of different optimization algorithms) and a subsequent selection of the appropriate possibility for the given context. Furthermore, we identify the following research trends and gaps:

- Deployment and production planning are the most often conducted ADDs, usually done together with scheduling and matching ADDs, respectively.
- Multi-dimensional ADDs are mostly conducted sequentially.
- Compositionally and incrementally searching ADDs are neglected, as well as application of meta-heuristics and multi-objective optimization.
- Timing aspects are the most often considered in different I40 ASO problems, whereas the reconfigurability and reliability constraints and objectives are almost neglected.
- The majority of papers use the benefits of Model-based Design (MbD) and apply different models as input for ASO.
- Domain-specific models are the most popular, especially for modeling software and hardware architectures.

2 Data Availability

The full version of this paper [TDV21] is published at the Foundation Track of ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2021. Private preprint paper and accompanying material can be requested for personal use at the ResearchGate website[4].

[4] https://www.researchgate.net/publication/354473962_Exploring_Architectural_Design_Decisions_in_Industry_40_A_Literature_Review_and_Taxonomy

Literaturverzeichnis

[Al13] Aleti, Aldeida; Buhnova, Barbora; Grunske, Lars; Koziolek, Anne; Meedeniya, Indika: Software architecture optimization methods: A systematic literature review. IEEE Transactions on Software Engineering, 2013.

[KKZ14] Keddis, Nadine; Kainz, Gerd; Zoitl, Alois: Capability-based planning and scheduling for adaptable manufacturing systems. In: 19th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA). 2014.

[TDV21] Terzimehić, Tarik; Dorofeev, Kirill; Voss, Sebastian: Exploring Architectural Design Decisions in Industry 4 . 0 : A Literature Review and Taxonomy. In: 24th Int. Conf. on Model Driven Eng. Languages and Systems (MODELS). 2021.

[WCB17] Wortmann, Andreas; Combemale, Benoit; Barais, Olivier: A Systematic Mapping Study on Modeling for Industry 4.0. In: ACM/IEEE 20th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS). 2017.

Assessing the Usefulness of a Visual Programming IDE for Large-Scale Automation Software

Bianca Wiesmayr[1], Alois Zoitl[2], Rick Rabiser[3]

Abstract: This is a summary of a paper (with the same title) that we published at the ACM/IEEE 24th International Conference on Model-Driven Engineering Languages and Systems (MODELS 2021) describing a study centered on a visual programming IDE for large-scale automation software development and maintenance.

Keywords: Usefulness study; open source software; IEC 61499; modeling tools; model-driven engineering

1 Summary

Industrial control software is more commonly developed and maintained by domain experts than by software engineers. These control engineers frequently use visual programming languages based on industrial standards such as IEC 61131-3 and IEC 61499. The latter standard applies model-based engineering concepts to abstract from hardware and low-level communication. Industrial control systems are usually complex, one-of-a-kind systems that are maintained for many years. Therefore, very usable IDEs for visual programming languages are required. However, not much empirical research exists on the practical usefulness of such IDEs, i.e., usability and utility.

We have conducted a multi-stage usability study to evaluate the usefulness of an open source modeling tool for IEC 61499, the 4diac IDE [Ec21]. In our paper [WZR21a], we discuss common control software maintenance tasks and tool capabilities derived from existing research and show their realization in the 4diac IDE. In the first stage, we performed a walkthrough of the capabilities using the cognitive dimensions of notations framework, where developers evaluated the tool usability. After improving the tool, we conducted a remote user study with ten industrial automation engineers. We asked each participant to perform maintenance tasks on provided automation software for a capping station. The study leader instructed the participant, while one scribe collected think-aloud statements and another scribe noted observations. After the practical work with the tool, we conducted a semi-structured utility interview and asked the subjects to fill in a usability questionnaire.

[1] LIT CPS Lab, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, bianca.wiesmayr@jku.at
[2] CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz, Austria, alois.zoitl@jku.at
[3] CDL VaSiCS, LIT CPS Lab, Johannes Kepler University Linz, Austria, rick.rabiser@jku.at

We grouped all results based on the cognitive dimensions. As expected, the understandability and maintainability of models depends on the provided IDE features. For example, we learned that the layout quality affects understanding visual models, but manually adjusting the graph layout is tedious. Users therefore benefit from an automated layout. However, they also need capabilities for customizing the layout algorithms to match their mental model. Orienting in a large application could be further facilitated with search features and reference lists. Finally, maintaining existing software may require complex editing operations. A modeling IDE should support these operations by handling inconsistencies gracefully. In general, we conclude that advanced tools with sophisticated editing support can simplify working with large models and thus increase the benefits of the applied modeling language.

Our findings demonstrate how the usefulness of IDEs can be successfully investigated using a multi-phase approach combining a walkthrough and a user study. Our results and lessons learnt from the study are relevant for developers of visual modeling and programming tools. The identified capabilities for software maintenance are often not sufficiently supported in such tools. Furthermore, our results with the Eclipse-based modeling tool 4diac IDE are potentially applicable to other modeling tools using the same technology. As our improvements are included in the latest open-source release of the 4diac IDE, they can serve as a good practice example for other project teams, together with this paper.

2 Data Availability

The tool 4diac IDE is part of the Eclipse 4diac™ project and therefore available open source, including all extensions that were developed as part of the presented study. Furthermore, also all documents from the study (interview questions, questionnaire, study system) are available online [WZR21b].

Literatur

[Ec21] Eclipse 4diac: Eclipse 4diac - The Open Source Environment for Distributed Industrial Automation and Control Systems, 2021, URL: www.eclipse.org/4diac.

[WZR21a] Wiesmayr, B.; Zoitl, A.; Rabiser, R.: Assessing the Usefulness of a Visual Programming IDE for Large-Scale Automation Software. In: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, 2021.

[WZR21b] Wiesmayr, B.; Zoitl, A.; Rabiser, R.: Assessing the Usefulness of a Visual Programming IDE for Large-Scale Automation Software: Documents for performing a user study on maintaining hierarchical control applications with 4diac IDE. 2021, URL: doi.org/10.5281/ZENODO.4758816.

Collaborative Software Modeling in Virtual Reality

Enes Yigitbas[1] Simon Gorissen[2] Nils Weidmann[3] Gregor Engels[4]

Abstract: This work has been published as a full paper at the MODELS'21 conference [Yi21]. Through collaborative modeling, end-users and experts can create a shared understanding of a system representation. While the Unified Modeling Language (UML) is one of the major conceptual modeling languages in object-oriented software engineering, more and more concerns arise from the modeling quality of UML and its tool-support. Among them, the limitation of the two-dimensional presentation of its notations and lack of natural collaborative modeling tools are reported to be significant. In this paper, we explore the potential of using Virtual Reality (VR) technology for collaborative UML software design by comparing it with classical collaborative software design using conventional devices (Desktop PC / Laptop). For this purpose, we have developed a VR modeling environment that offers a natural collaborative modeling experience for UML Class Diagrams. Based on a user study with 24 participants, we have compared collaborative VR modeling with conventional modeling with regard to efficiency, effectiveness, and user satisfaction. Results show that the use of VR has some disadvantages concerning efficiency and effectiveness, but the user's fun, the feeling of being in the same room with a remote collaborator, and the naturalness of collaboration were increased.

Keywords: Collaborative Modeling; Virtual Reality; UML

1 Introduction

While the Unified Modeling Language (UML) is one of the major conceptual modeling languages for software engineers, more and more concerns arise from the modeling quality of UML and its tool-support. Among them, the limitation of the two-dimensional presentation of its notations and lack of natural collaborative modeling tools are reported to be significant. In this work, we have explored the potential of using Virtual Reality (VR) technology for collaborative UML software design by comparing it with classical collaborative software design using conventional devices (Desktop PC / Laptop). For this purpose, we have developed a VR modeling environment, called *VmodlR*, that offers a natural collaborative modeling experience for UML Class Diagrams. Based on a user study with 24 participants, we have compared collaborative VR modeling with a conventional web modeling tool with regard to efficiency, effectiveness, and user satisfaction.

[1] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany enes@mail.upb.de
[2] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany scg@mail.upb.de
[3] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany nilsweid@mail.upb.de
[4] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany engels@upb.de

2 Main Findings

Based on previous research and the participants' familiarity with conventional web modeling tools, we expected one downside of VR to be that users' task executions are slower and more error-prone in VR. The data from our study shows that this was indeed the case. These measures could have been influenced by the universal familiarity of participants with existing web modeling tools. It is therefore possible that speed and error rates in VR improve as users get more experience with a specific VR UML tool. The results of the System Usability Scale (SUS) evaluation additionally showed that our VR implementation is already quite usable even though it still lacks many features that users expect from such an application like automatic aligning of model elements and copy & paste functionalities. The various data points gathered about the naturalness of different aspects of the application gave a clearer insight on what concrete advantages such a VR application can have compared to traditional tools: On average, users found the interactions and especially the collaboration related aspects of the VR application significantly more natural than in the web application. This is especially important with respect to this work's focus on remote collaboration settings as the study showed that the feelings of being together and collaborating face to face with a co-worker were much higher in VR compared to the traditional PC alternative. Another aspect that we expected VR to be beneficial for is the motivation and fun users are having while using it. Our study shows that it is indeed true that users were a lot more motivated and had a lot more fun using the VR application compared to the web application. It is important to note that this could be influenced by the fact that VR is a relatively new and therefore possibly more interesting technology, so these values might align more over time when a user regularly uses a VR application for modeling. Summarizing, it can be said that VR can offer a more natural collaborative modeling experience compared to PC-based tools but that both techniques have certain advantages and drawbacks. These make the use of both tool-types, depending on the concrete situation, most sensible instead of using only one of them exclusively.

3 Data Availability

The code repository of our developed collaborative VR-based UML modeling tool VmodlR as well as details about the user study and corresponding evaluation data can be found at IRB Git[5].

Literaturverzeichnis

[Yi21] Yigitbas, Enes; Gorissen, Simon; Weidmann, Nils; Engels, Gregor: Collaborative Software Modeling in Virtual Reality. In: 24th International Conference on Model Driven Engineering Languages and Systems, MODELS 2021, Fukuoka, Japan, October 10-15, 2021. IEEE, S. 261–272, 2021.

[5] https://git.cs.uni-paderborn.de/scg/vmodlr

Enhancing Human-in-the-Loop Adaptive Systems through Digital Twins and VR Interfaces

Enes Yigitbas[1] Kadiray Karakaya[2] Ivan Jovanovikj[3] Gregor Engels[4]

Abstract: This work has been published as a full paper at SEAMS'21 [Yi21]. In the context of self-adaptive systems, there are situations where human involvement in the adaptation process is beneficial or even necessary. For such "human-in-the-loopädaptive systems, two major challenges, namely transparency, and controllability must be addressed to include the human in the self-adaptation loop. Transparency covers the context information about the adaptive system and its context while controllability targets the decision-making and adaptation operations. As existing human-in-the-loop adaptation approaches do not fully cover these aspects, we investigate alternative human-in-the-loop strategies by using a combination of digital twins and virtual reality (VR) interfaces. Based on the concept of the digital twin, we represent a self-adaptive system and its respective context in a virtual environment. For integrating the human in the decision-making and adaptation process, we have implemented and analyzed two different human-in-the-loop strategies in VR: a procedural control where the human can control the decision making-process and adaptations through VR interactions and a declarative control where the human specifies the goal state and the configuration is delegated to an AI planner. We evaluate our approach based on an autonomic robot system that is accessible through a VR interface.

Keywords: Self-Adaptive Systems; Human-in-the-loop; Virtual Reality

1 Introduction

In this paper, we discuss the benefit of human involvement in self-adaptive systems and the need for increased transparency and controllability in such human-in-the-loop adaptive systems. To address these issues, we have presented a solution approach for enhancing human-in-the-loop adaptive systems through digital twins and VR interfaces. For supporting the aspect of transparency, based on the concept of the digital twin, we represent a self-adaptive system and its respective context in a virtual environment. With the help of a virtual reality (VR) interface, we support an immersive and realistic human involvement in the self-adaptation loop by mirroring the physical entities of the real world to the VR interface. For supporting the aspect of controllability and integrating the human in the decision-making and adaptation process, we have implemented and analyzed two different human-in-the-loop strategies in VR: a procedural control where the human can control

[1] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany enes@mail.upb.de
[2] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany kadiray.karakaya@upb.de
[3] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany ivanj@mail.upb.de
[4] Paderborn University, Zukunftsmeile 2, 33102 Paderborn, Germany engels@upb.de

the decision making-process and adaptations through VR interactions (human-controlled) and a declarative control where the human specifies the goal state and the configuration is delegated to an AI planner (mixed-initiative). The solution approach which combines digital twin and VR interfaces has been evaluated regarding efficiency, effectiveness, and user satisfaction and shows great potential in increasing the transparency and control of human-in-the-loop adaptive systems.

2 Main Findings

Concerning efficiency, the results show that declarative control provides 35.2% faster task completion. This is an expected result because the number of interactions is less compared with declarative control. The effectiveness of the VR interface is the same with each control strategy. Differences can be seen regarding the effectiveness of the VR interface and the execution on the real Dobot Magician system. This is expected, due to potentially imprecise sensor data. In general, the user satisfaction data show promising results for the practical usage of the VR interface for human-in-the-loop involvement.

Our results show that using a graphical representation, such as a VR interface, it is possible to provide transparency about the system state. This approach does not require users to have extensive domain knowledge to reason about the system. Moreover, with our approach, people who used the system for the first time, were able to intervene in the system to complete a task by involving in its self-adaptation loop. We report that with procedural control, the users need more time to complete the task, but they have more freedom to define what exactly the system should perform. On the other hand, with declarative control, the users save a significant amount of time to complete the task, gain a slight precision benefit due to automated motion calculation, but they have less freedom to express their intentions.

3 Data Availability

The code repository of our developed solution for enhancing human-in-the-loop adaptive systems and corresponding evaluation data can be found on GitHub[5].

Literaturverzeichnis

[Yi21] Yigitbas, Enes; Karakaya, Kadiray; Jovanovikj, Ivan; Engels, Gregor: Enhancing Human-in-the-Loop Adaptive Systems through Digital Twins and VR Interfaces. In: 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2021, Madrid, Spain, May 18-24, 2021. IEEE, S. 30–40, 2021.

[5] https://github.com/kadirayk/HiL-VR-DT

Workshops

Requirement Management in Enterprise Systems Projects (AESP'22)

Christoph Weiss[1] Johannes Keckeis[2]

Abstract: ERP systems and other enterprise systems are the backbone of any company in a digitized world. Besides enterprise systems driven requirements companies often face additional organizational requirements and challenges during enterprise systems projects like evaluating, implementing or improving.

Keywords: enterprise systems; requirement management; enterprise systems driven requirements; organizational requirements; enterprise systems sucess factors; enterprise systems consulting

1 Motivation

ERP systems and other enterprise systems are the backbone of any company in a digitized world. In almost every company Enterprise Systems are adapted to the needs of the customers within the scope of parameterization, modifications (changes to existing functions and logics) or even extensions (new developments of existing functions and logics). However, many of such Enterprise Systems projects fail due to missing, incorrect, inadequate or incomplete requirements there are "incorrect" expectations, divergents in definition and attitudes on requirements management between customers and suppliers.

Besides enterprise systems driven requirements companies often face additional organizational requirements and challenges during enterprise systems projects like evaluating, implementing or improving.

- New or changed business processes
- New or changed business organization
- demand, capacity and availability of the relevant personal resources
- enterprise systems and process competency and knowledge of the involved personal resources
- funding and budgeting
- operationalization of lean and agile methods

[1] SIS Consulting GmbH, Kaiserjägerstraße 1, 6020 Innsbruck, Austria; Christoph.Weiss@sis-consulting.com
[2] SIS Consulting GmbH, Kaiserjägerstraße 1, 6020 Innsbruck, Austria; Johannes.Keckeis@sis-consulting.com

2 Expected results (work objectives) of the workshop

The enterprise systems driven and organizational driven requirements and challenges will be highlighted, talked over and discussed during this workshop. Presentation of points of view, ways of thinking, processes and perceptions of enterprise systems customers or prospective customers and providers in selection, implementation and further development of Enterprise Systems in the context of requirement and challange management.

3 Workshop Agenda

- Key Note
- Paper presentation
- Discussions and Lessons Learned
- Summary

4 Program Comitee

- Wolfgang Ahammer, VFI
- Christian Büll, FH Burgenland
- Fritz Fahringer and Franz Unterluggauer, Standort Tirol, Cluster IT Tirol
- Günther Glawar, EVVA
- Helmut Gollner, FH Technikum Wien
- Felix Piazolo, Universität Innsbruck
- Kurt Promberger, Universität Innsbruck
- Ludwig Rupp, Rupp Käse

19th Workshop on Automotive Software Engineering (ASE'22)

Heiko Dörr[1], Steffen Helke[2]

Abstract: Software based systems play an increasingly important role and enable most of the innovations in modern cars. This workshop deals with various topics related to the development of automotive software and discusses suitable methods, techniques, and tools necessary to master the most current challenges researchers and practitioners are facing.

Keywords: Automotive Software Engineering; Autonomous Driving; Driver Assistance Systems; Software Development

The 19th Workshop on Automotive Software Engineering (ASE'22) addresses the challenges of software development in the automotive sector as well as suitable methods, techniques, and tools for this specific area. With the increasing amount of connected vehicles, modern driver assistance systems, and the challenges of fully automated driving, automotive software plays an important role today more than ever.

Furthermore, the distraction-free and intuitive operation of vehicle applications via multimodal interfaces play an increasingly important role. In addition, innovative technologies like voice control, cloud computing, or 5G connectivity found their way into the car. These technological advances have changed the experience of driving a car: In the near future services such as WhatsApp, Skype or even Facebook will be integrated into the car and can then be operated by users while driving.

The main objectives of the workshop are the exchange and discussion of how current challenges in automotive software engineering can be mastered. The thematic orientation offers many cross-references to the Software Engineering (SE) conference to which the workshop is colocated. The workshop addresses researchers, developers, and users from the automotive industry as well as scientists from research institutes and universities working in the field of automotive software engineering. Traditionally, the focus is less on theory than on applied research.

To ensure that only high-quality submissions are selected for publication and presentation, two reviewers were selected for each of the contributions submitted to this year's workshop. Many thanks to all the reviewers who contributed with great commitment to the review process.

[1] Method Park by UL, heiko.doerr@methodpark.de
[2] Fachhochschule Südwestfalen, helke.steffen@fh-swf.de

Program Committee

Dr. Christian Allmann	Audi AG
Prof. Dr. Marcel Baunach	Technische Universität Graz
Dr. Mirko Conrad	samoconsult GmbH
Prof. Dr. Sabine Glesner	Technische Universität Berlin
Dr. Kerstin Hartig	Expleo Germany GmbH
Prof. Dr. Paula Herber	Universität Münster
Prof. Dr. Thomas Kropf	Robert Bosch GmbH
Prof. Dr. Stefan Kugele	Technische Hochschule Ingolstadt
Apl. Prof. Dr. Müller	Universität Paderborn
Dr. Thomas Noack	Datendeuter GmbH
Prof. Dr. Ralf Reißing	Hochschule Coburg
Prof. Dr. Eric Sax	Karlsruhe Institute of Technology (KIT)
Prof. Dr. Ina Schaefer	Technische Universität Braunschweig
Prof. Dr. Holger Schlingloff	Humboldt Universität und Fraunhofer FOKUS, Berlin
Prof. Dr. Jörn Schneider	Hochschule Trier
Prof. Dr. Ramin Tavakoli	Technische Hochschule Nürnberg
Prof. Dr. Thomas Thüm	Universität Ulm
Dr. Thomas Vogel	Humboldt Universität zu Berlin
Prof. Dr. Andreas Vogelsang	Universität zu Köln
Dr. Rebekka Wohlrab	Carnegie Mellon University

Organization

Dr. Heiko Dörr	Method Park by UL
Prof. Dr. Steffen Helke	Fachhochschule Südwestfalen

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] http://fg-ase.gi.de/

4th Workshop on Avionics Systems and Software Engineering (AvioSE'22)

Björn Annighöfer[1], Andreas Schweiger[2], Marina Reich[3]

Abstract: Software and systems engineering in aerospace is subject to special challenges. The AvioSE'22 workshop connects academia and industry with selected scientific presentations, motivating keynote talks, and an interactive panel discussion.

Keywords: avionics; systems engineering; software engineering; formal methods; model-based; requirements; 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, the 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 aerospace requirements. These challenges are to be addressed in the AvioSE workshop. The previous workshops AvioSE'19[4] dealt with general challenges and AvioSE'20[5] addressed development tools. AvioSE'21[6] tackled the application and issues of AI in avionics.

The main topic of AvioSE'22 is *Safe and secure avionics architectures (e.g. IMA, platforms, multi-core, networks, clouds, middleware)*. Generic avionics platforms seem to be the obvious choice for air vehicles developed today. A low number of standardized avionics hardware shares its resources with several other system functions. While benefits in size, weight, power, and cost are agreed, generic platforms bring along challenges caused by a concurrent development process, distributed safety analysis, generic security vulnerabilities, configuration engineering, and software segregation.

[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 Defence and Space GmbH, University of Chemnitz, Manching, Germany, marina.reich@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.

2 Workshop Objectives

The main objective of the workshop is to accelerate the bidirectional transfer of knowledge between academia and industry. This workshop provides the enabling platform for the stakeholders to discuss technical, but also process, and educational topics. The objectives of AvioSE'22 are three-fold: (1) It provides a forum for researchers from both academia and industry to present new methods, tools, and technologies from avionics systems and software engineering, e.g. model-based development, requirements engineering, formal methods, model-based methods, and virtual methods. These contributions are presented in a scientific format, but the small character of the workshop allows detailed discussions. (2) AvioSE'22 shall motivate researchers by keynote talks from three invited speakers. The keynotes each highlight a dedicated topic from the avionics context, summarize its state-of-the-art, and emphasize urgent challenges. (3) The main topic is addressed interactively by inviting all participants to discuss aspects and needs of modern avionics architectures. 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 the current and future challenges and needs for avionics platforms as well as ideas how to address these challenges. The panel members' statements can be challenged at any time 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 and presenters of the accepted papers. Second, high appreciation goes to our keynote speakers Maarten Uitj de Haag from TU Berlin, Stephane Poulain from Airbus Operations GmbH and Christoph Diesch from Airbus Defence and Space GmbH. Third, sincere thanks go to the panelists for sharing their knowledge and their will to answer our critical questions. Additionally, we want to express our gratitude to the SE 2022 organizers for supporting and hosting our workshop. Finally, we are happy for all people that served as members in the program committee, 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), María Ángeles Martín Prats (University of Seville), 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).

Workshop on Software Engineering in Cyber-Physical Production Systems (SECPPS), 2nd Edition

Rick Rabiser[1], Birgit Vogel-Heuser[2], Manuel Wimmer[3], Andreas Wortmann[4], Alois Zoitl[1]

Abstract: This workshop focuses on Software Engineering in Cyber-Physical Production Systems (SECPPS). SECPPS is an interactive workshop opened by keynotes and lightning talks, followed by project showcase presentations, and concluded by extensive discussions in break-out groups. The main output of SECPPS 2022 is an updated research roadmap as well as concrete networking activities to further grow the community in this interdisciplinary field.

Keywords: Software Engineering; Cyber-Physical Production Systems; Workshop

1 Context and Motivation

Software is nowadays the innovation driver in assuring effective and efficient engineering and operation of industrial automation systems. However, the traditional software engineering methods applied in this field are challenged by emerging methods where tremendous progress has been made in the last years in the general field of engineering and operating software-intensive systems. At the same time, the community is currently facing a dramatically increasing complexity in the engineering and operation of systems with the emergence of Cyber-Physical Production Systems (CPPS). To tackle this complexity increase, concepts, techniques, methods, and technologies are required to build systematic and comprehensive views of all aspects of CPPS, e.g., mechanics, electronics, software, and network, not only in the engineering phase, but in the operation phase as well. Moreover, more flexible methodologies are needed to adapt existing systems to ever-changing requirements and tasks, unexpected conditions, as well as structural transformations [Mo14].

The aim of this workshop is to discuss new emerging trends for the engineering and operation of software in the field of CPPS. A particular focus is on the current challenges [Vo15] the community is facing in adopting these emerging software engineering concepts, techniques, and technologies and best practices to tackle the open challenges.

[1] CDL VaSiCS, LIT CPS, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, firstname.lastname@jku.at
[2] AIS, TU Munich, Boltzmannstr. 15, 85748 Garching bei München, Germany vogel-heuser@tum.de
[3] WIN-SE, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, manuel.wimmer@jku.at
[4] ISW, University of Stuttgart, Seidenstr. 36, 70174 Stuttgart, Germany andreas.wortmann@isw.uni-stuttgart.de

2 Program, Format, and Topics

SECPPS employs an interactive format to stimulate group discussions which potentially lead to further inter-disciplinary research activities such as joint publications, projects, demonstrators, and research networks.

To reach this goal, two keynote speakers, one from the SE community as well as one from the CPPS community, present current research efforts in the two fields in order to set the stage for the workshop. The keynotes are followed by lightning talks and project showcase presentations to complement the presented viewpoints. For each presentation, questions, challenges, and provocative statements are collected to be potentially discussed in break-out groups in the afternoon sessions. The collected data from the morning sessions is clustered in common topics which are discussed in break-out groups in the afternoon. The results of the break-out groups are presented by the group leaders and reflected in the large audience to come up with an update of the collaborative research roadmap established in the workshop's first edition in 2021 and networks to identify further potential collaborations are initiated.

We foresee the following initial list of topics for discussions at the workshop:

- Engineering Process (Requirements, Design, Implementation, Verification & Validation, ...)
- Operation and Evolution (Data-driven, Continuous Integration, DevOps, Digital Twins, Agile, ...)
- Languages (DSLs, GPLs, standards, ...)
- Modeling (MDD, MDE, Transformations, Interoperability, Code generation, ...)
- Teaching (How to train SE in other disciplines, open courseware, ...)
- Management (Variability, Modularization, Configuration, Versioning, ...)
- Usability and SE Tools (Adoption, User Interactions, Needs, ...)
- Emerging Technologies (Cloud, AI, IoT, ...)
- Intelligent Organization (Multi-Agent Systems, Flexible Architectures, Adaptive Systems, ...)
- Interdisciplinary Collaboration (Interfaces, Conflict Management, Optimization, Distributed Work, ...)

3 Website and Further Information

See `https://rickrabiser.github.io/secpps-ws/` for more information.

Bibliography

[Mo14] Monostori, László: Cyber-physical production systems: Roots, expectations and R&D challenges. Procedia CIRP, 17:9–13, 2014.

[Vo15] Vogel-Heuser, Birgit; Fay, Alexander; Schaefer, Ina; Tichy, Matthias: Evolution of software in automated production systems: Challenges and research directions. J. Syst. Softw., 110:54–84, 2015.

Autorenverzeichnis