GESELLSCHAFT
FÜR INFORMATIK

Anne Koziolek, Ina Schaefer,
Christoph Seidl (Hrsg.)

Software Engineering 2021

Fachtagung des GI-Fachbereichs Softwaretechnik

22. – 26. Februar 2021
Braunschweig/Virtuell

Gesellschaft für Informatik e.V. (GI)

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

Volume P-310

ISBN 978-3-88579-704-3
ISSN 1617-5468

Volume Editors
Prof. Dr.-Ing. Anne Koziolek
Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology
Am Fasanengarten 5, 76131 Karlsruhe, Germany
anne.koziolek@kit.edu
Prof. Dr.-Ing. Ina Schaefer
Institute of Software Engineering and Automotive Informatics
Technische Universität Braunschweig
Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
i.schaefer@tu-braunschweig.de
Prof. Dr.-Ing. Christoph Seidl
Department of Computer Science
IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark
chse@itu.dk

Seminars
Andreas Oberweis, Karlsruher Institut für Technologie (KIT), Germany

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Herzlich willkommen zur Tagung Software Engineering 2021 (SE 21) 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. Der Austausch erstreckt sich dabei sowohl auf neueste akademische Erkenntnisse als auch auf aktuelle industrielle Trends und Praktiken. Die Tagung richtet sich sowohl an Softwareentwicklerinnen und Softwareentwickler aus der Praxis, als auch an Forscherinnen und Forscher aus dem akademischen Umfeld. Software ist der wesentliche Bestandteil um aktuelle Herausforderungen in Wirtschaft und Gesellschaft zu meistern und auch weiterhin weltweit wettbewerbsfähige Produkte und Dienstleistungen anbieten zu können.

Die SE 21 bietet im wissenschaftlichen Hauptprogramm ein "Best-Of" der international in Fachzeitschriften und Konferenzen veröffentlichten Arbeiten deutschsprachiger Autoren. Sie umfasst eine große Bandbreite an Themen, die beispielsweise in der International Conference on Software Engineering (ICSE), den IEEE Transactions on Software Engineering (TSE), den ACM Transactions of Software Engineering and Methodology (TOSEM) und der ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) und vielen weiteren einschlägigen Fachzeitschriften und Konferenzen veröffentlicht wurden. Die angenommenen wissenschaftlichen Beiträge decken dabei ein weites Spektrum des Software Engineering ab, das sich in einem vielfältigen Programm widerspiegelt.

Das wissenschaftliche Hauptprogramm der SE 21 wird durch sechs Workshops ergänzt, in denen weitere Themen im kleineren Kreis intensiv diskutiert werden:

Requirement Management in Enterprise Systems Projects (AESP'21)

Automotive Software Engineering (ASE'21)

Avionics Systems and Software Engineering (AvioSE'21)

Evolution and Maintenance of Long-Living Systems (EMLS'21)

Software Engineering in Cyber-Physical Production Systems (SECPPS'21)

Software Engineering for E-Learning-Systems (SEELS'21)

Wir danken allen, die zum Gelingen der Konferenz beigetragen haben, insbesondere den Autoren und den Gutachtern, den Keynote-Speakern, den Organisatoren der Workshops und Tracks, den Teilnehmern, den Sponsoren (IAV Automotive Engineering und eck*cellent IT) und Unterstützern (Kenzler Conference Management, der Technischen Universität Braunschweig im Allgemeinen und dem Institut für Softwaretechnik und Fahrzeuginformatik im Speziellen sowie der GI e.V.), dem Local Organizer Michael Nieke, der Proceedings Chair Sofia Ananieva sowie allen Helferinnen und Helfern, die die Durchführung der Konferenz auch im virtuellen Format ermöglicht haben.

Braunschweig, im Februar 2021

Ina Schaefer, Christoph Seidl und Anne Koziolek

Sponsoren

Wir danken den folgenden Unternehmen für die Unterstützung der Konferenz.

eck*cellent IT GmbH

Theodor-Heuss-Straße 2
38122 Braunschweig

Webseite
https://eckcellent-it.de/

IAV GmbH Ingenieurgesellschaft
Auto und Verkehr

Carnotstraße 1
10587 Berlin

Webseite
https://www.iav.com/

Tagungsleitung

Gesamtleitung:	Ina Schaefer, Technische Universität Braunschweig
Leitung des Programmkomitees:	Anne Koziolek, Karlsruhe Institute of Technology Christoph Seidl, IT University of Copenhagen
Industrie:	Andreas Vogelsang, University of Cologne Elmar Juergens, CQSE GmbH
Workshops:	Sebastian Götz, Technische Universität Dresden Andreas Wortmann, RWTH Aachen University
Tools & Demos:	Lukas Linsbauer, Technische Universität Braunschweig
Lokale Organisation:	Michael Nieke, Technische Universität Braunschweig
Proceedings:	Sofia Ananieva, FZI Research Center for Information Technology
Publicity:	Adrian Hoff, IT University of Copenhagen

Programmkomitee

Sven Apel	Universität des Saarlandes
Stefan Biffl	TU Wien
Ruth Breu	Universität Innsbruck
Gordon Fraser	Universität Passau
Sabine Glesner	TU Berlin
Martin Glinz	Universität Zürich
Lars Grunske	Humboldt-Universität zu Berlin
Wilhelm Hasselbring	Christian-Albrechts-Universität zu Kiel
Barbara Paech	Universität Heidelberg
Martin Pinzger	Universität Klagenfurt
Klaus Pohl	Universität Duisburg-Essen
Rick Rabiser	Johannes Kepler Universität Linz
Ina Schaefer	TU Braunschweig
Klaus Schmid	Universität Hildesheim
André van Hoorn	Universität Stuttgart
Andreas Vogelsang	Universität zu Köln
Manuel Wimmer	Johannes Kepler Universität Linz
Uwe Zdun	Universität Wien

Inhaltsverzeichnis

Keynotes

Ralf S. Engelschall
Schönheit und Unzulänglichkeit von Software-Architektur 19

Wissenschaftliches Hauptprogramm

Manuel Benz, Erik K. Kristensen, Linghui Luo, Nataniel Borges Jr., Eric Bodden, Andreas Zeller
Heaps'n Leaks: How Heap Snapshots Improve Android Taint Analysis . . 23

Stefanie Beyer, Christian Macho, Massimiliano Di Penta and Martin Pinzger
What Kind of Questions Do Developers Ask on Stack Overflow? A Comparison of Automated Approaches to Classify Posts Into Question Categories 27

Andreas Dann, Ben Hermann, Eric Bodden
ModGuard: Identifying Integrity & Confidentiality Violations in Java Modules 29

Marian Daun, Jennifer Brings, Thorsten Weyer
Validierung von CPS-Spezifikationen 33

Wolfram Fenske, Jacob Krüger, Maria Kanyshkova, Sandro Schulze
#ifdef Directives and Program Comprehension: The Dilemma between Correctness and Preference 35

Jannik Fischbach, Andreas Vogelsang, Dominik Spies, Andreas Wehrle, Maximilian Junker, Dietmar Freudenstein
SPECMATE: Automated Creation of Test Cases from Acceptance Criteria . . 37

Stefan Fischer, Gabriela Karoline Michelon, Rudolf Ramler, Lukas Linsbauer, Alexander Egyed
Automated Reuse of Test Cases for Highly Configurable Software Systems 39

Sergio García, Daniel Strüber, Davide Brugali, Thorsten Berger, Patrizio Pelliccione
Robotics Software Engineering: A Perspective from the Service Robotics Domain 41

Christopher Gerking, David Schubert
Component-Based Refinement and Verification of Information-Flow Security Policies for Cyber-Physical Microservice Architectures 43

Stefan Götz, Matthias Tichy, Raffaela Groner
Claimed Advantages and Disadvantages of (dedicated) Model Transformation Languages: A Systematic Literature Review 45

Rahul Gopinath, Nikolas Havrikov, Alexander Kampmann, Ezekiel Soremekun, Andreas Zeller
Learning Circumstances of Software Failures 47

Rahul Gopinath, Björn Mathis, Andreas Zeller
Mining Input Grammars 49

Raffaela Groner, Luis Beaucamp, Matthias Tichy, Steffen Becker
An Exploratory Study on Performance Engineering in Model Transformations 51

Steffen Herbold
On the Cost and Profit of Software Defect Prediction 53

Steffen Herbold, Alexander Trautsch, Fabian Trautsch
On the Feasibility of Automated Prediction of Bug and Non-Bug Issues . . 55

Steffen Herbold, Aynur Amirfallah, Fabian Trautsch, Jens Grabowski
A Systematic Mapping Study of Developer Social Network Research 57

Jörg Holtmann, Jan-Philipp Steghöfer, Michael Rath, David Schmelter
Cutting through the Jungle: Disambiguating Model-based Traceability Terminology 59

Arno Kesper, Viola Wenz, Gabriele Taentzer
Detecting Quality Problems in Research Data; A Model-Driven Approach . 61

Lukas Kirschner, Ezekiel Soremekun, Andreas Zeller
Isolating Faults in Failure-Inducing Inputs 63

Jil Klünder, Dzejlana Karajic, Paolo Tell, Oliver Karras, Christian Münkel, Jürgen Münch, Stephen G. MacDonell, Regina Hebig, Marco Kuhrmann
Determining Context Factors for Hybrid Development Methods with Trained Models 65

Alexander Knüppel, Inga Jatzkowski, Marcus Nolte, Tobias Runge, Thomas Thüm, Ina Schaefer
Skill-Based Verification of Cyber-Physical Systems 67

Jacob Krüger, Thorsten Berger
An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse 69

Jacob Krüger, Regina Hebig
What Developers (Care to) Recall: An Interview Survey on Smaller System . 71

Dorian Leroy, Erwan Bousse, Manuel Wimmer, Tanja Mayerhofer, Benoit Combemale, Wieland Schwinger
Behavioral Interfaces for Executable DSLs 73

Andreas Metzger, Clément Quinton, Zoltán Mann, Luciano Baresi, Klaus Pohl
Feature-Modell-geführtes Online Reinforcement Learning 75

Malte Mues, Till Schallau, Falk Howar
Security Analysis with JAINT 77

Stefan Mühlbauer, Sven Apel, Norbert Siegmund
Accurate Modeling of Performance Histories for Evolving Software Systems 79

Hoang Lam Nguyen, Nebras Nassar, Timo Kehrer, Lars Grunske
MoFuzz: Fuzzing for MDSE Tools 81

Felix Pauck, Heike Wehrheim
Cooperative Android App Analysis with CoDiDroid 83

Nataniel Pereira Borges Jr., Nikolas Havrikov, Andreas Zeller
Generating Tests that Cover Input Structure 85

Florian Pudlitz, Florian Brokhausen, Andreas Vogelsang
Testing Procedures Based on Requirements Annotations 87

Rick Rabiser, Klaus Schmid, Holger Eichelberger, Michael Vierhauser, Paul Grünbacher
A Domain Analysis of Resource and Requirements Monitoring 91

Tobias Runge, Ina Schaefer, Loek Cleophas, Thomas Thüm, Derrick Kourie, Bruce W. Watson
Tool Support for Correctness-by-Construction 93

Aaron Schlutter, Andreas Vogelsang
Trace Link Recovery Using Semantic Relation Graphs and Spreading Activation 95

Ezekiel Soremekun, Esteban Pavese, Nikolas Havrikov, Lars Grunske, Andreas Zeller
Probabilistic Grammar-based Test Generation 97

Helge Spieker, Arnaud Gotlieb
Learning to Generate Fault-revealing Test Cases in Metamorphic Testing . 99

Patrick Stöckle, Bernd Grobauer, Alexander Pretschner
Automated Implementation of Windows-related Security-Configuration Guides 101

Daniel Strüber, Anthony Anjorin, Thorsten Berger
Variability Representations in Class Models: An Empirical Assessment . . 103

Cem Sürücü, Bianying Song, Jacob Krüger, Gunter Saake, Thomas Leich
Using Key Performance Indicators to Compare Software-Development Processes 105

Alexander Trautsch, Steffen Herbold, Jens Grabowski
Static Analysis Warning Evolution and the Effects of PMD 107

Fabian Trautsch, Steffen Herboldh, Jens Grabowski
Are Unit and Integration Test Definitions Still Valid for Modern Java Projects? An Empirical Study on Open-Source Projects 109

Alex Villazón, Haiyang Sun, Andrea Rosà, Eduardo Rosales, Daniele Bonetta, Isabella Defilippis, Sergio Oporto, Walter Binder
Automated Large-scale Multi-language Dynamic Program Analysis in the Wild 111

Andreas Vogelsang, Jonas Eckhardt, Daniel Mendez, Moritz Berger
Views on Quality Requirements in Academia and Practice: Commonalities, Differences, and Context-Dependent Grey Areas 113

Stefan Wagner, Daniel Méndez Fernández, Michael Felderer, Antonio Vetrò, Marcos Kalinowski, Roel Wieringa, Dietmar Pfahl et al.
Status Quo in Requirements Engineering 115

Sebastian Weigelt, Vanessa Steurer, Tobias Hey, Walter F. Tichy
Programming in Natural Language with fuSE 117

Florian Wiesweg, Andreas Vogelsang, Daniel Mendez
Data-driven Risk Management for Requirements Engineering: An Automated Approach based on Bayesian Networks 119

Franz Zieris, Lutz Prechelt
Explaining Pair Programming Session Dynamics from Knowledge Gaps . . 121

Workshops

Christoph Weiss, Johannes Keckeis
2nd Workshop on Requirement Management in Enterprise Systems Projects (AESP'21) 125

Patrick Ebel, Steffen Helke, Ina Schaefer, Andreas Vogelsang
18th Workshop on Automotive Software Engineering (ASE'21) 127

Björn Annighöfer, Andreas Schweiger, Marina Reich
3rd Workshop on Avionics Systems and Software Engineering (AvioSE'21) 129

Robert Heinrich, Reiner Jung, Marco Konersmann, Eric Schmieders
8th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS'21) 131

Rick Rabiser, Birgit Vogel-Heuser, Manuel Wimmer, Alois Zoitl
Workshop on Software Engineering in Cyber-Physical Production Systems (SECPPS'21) 133

Sven Strickroth, Michael Striewe
Workshop on Software Engineering for E-Learning Systems (SEELS'21) . 135

Keynotes

Schönheit und Unzulänglichkeit von Software-Architektur

Ralf S. Engelschall [1]

Software-Architektur ist die Königsdisziplin schlechthin im industriellen Software-Engineering. Sie zeigt sich aber oft von zwei gegensätzlichen Seiten: sie kann einerseits konzeptionell äußerst elegant und wunderschön sein, andererseits ist sie in der Praxis regelmäßig schwach und unzulänglich.

Was steckt dahinter? Wieso tun wir uns auch nach 50 Jahren Software-Engineering immer noch so schwer mit Software-Architektur? An welchen Stellen sollten wir erneut forschen und die Disziplin eventuell noch Mal überdenken? Wie können wir die kommenden Generationen von Software-Architekten noch besser ausbilden?

[1] msg systems ag, msg Research, Robert-Bürkle-Straße 1, 85737 Ismaning, Germany ralf.engelschall@msg.group

doi:10.18420/SE2021_01

Wissenschaftliches Hauptprogramm

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 23

Heaps'n Leaks: How Heap Snapshots Improve Android Taint Analysis

Manuel Benz[1], Erik Krogh Kristensen[2], Linghui Luo[3], Nataniel P. Borges Jr.[4], Eric Bodden[5], Andreas Zeller[6]

Abstract: The assessment of information flows is an essential part of analyzing Android apps, and is frequently supported by static taint analysis. Its precision, however, can suffer from the analysis not being able to precisely determine what elements a pointer can (and cannot) point to. Recent advances in static analysis suggest that incorporating dynamic heap snapshots, taken at one point at runtime, can significantly improve general static analysis. In this paper, we investigate to what extent this also holds for *taint* analysis, and how various design decisions, such as when and how many snapshots are collected during execution, and how exactly they are used, impact soundness and precision. We have extended FlowDroid to incorporate heap snapshots, yielding our prototype Heapster, and evaluated it on DroidMacroBench, a novel benchmark comprising real-world Android apps that we also make available as an artifact. The results show 1. the use of heap snapshots lowers analysis time and memory consumption while increasing precision; 2. a very good trade-off between precision and recall is achieved by a *mixed mode* in which the analysis falls back to static points-to relations for objects for which no dynamic data was recorded; and 3. while a *single* heap snapshot (ideally taken at the end of the execution) suffices to improve performance and precision, a better trade-off can be obtained by using multiple snapshots.

Keywords: points-to analysis; heap snapshot; taint analysis; Soot; Android

1 Introduction

Android is the world's most popular mobile operating system. Its official marketplace, Google Play Store, holds more than 3.3 million apps, which can be installed on billions of devices. To perform their tasks, apps frequently interact with sensitive information—from private images to banking details. Research shows that security-related bugs introduced by developers frequently put this sensitive information at risk [En11; En14; Gr12; Ra15].

To identify such sensitive information leaks, *taint analysis* detects potential leaks by determining if data acquired on a sensitive source reaches a sink, where the information would no longer be secure. Such taint flows can be detected statically or dynamically. A *static* taint analysis, which we focus on in this paper, reasons about all possible execution

[1] Paderborn University, Department of Computer Science manuel.benz@codeshield.de
[2] Aarhus University, Department of Computer Science erik@cs.au.dk
[3] Paderborn University, Department of Computer Science linghui.luo@upb.de
[4] CISPA Helmholtz Center for Information Security, Department of Computer Science nataniel.borges@cispa.saarland
[5] Paderborn University & Fraunhofer IEM, Department of Computer Science eric.bodden@upb.de
[6] CISPA Helmholtz Center for Information Security, Department of Computer Science zeller@cispa.saarland

doi:10.18420/SE2021_02

paths in a program and aims to achieve (close to) perfect recall, i.e., it seeks to identify virtually all potentially sensitive information leaks. Static analyses, though, often suffer from a trade-off between accuracy and scalability. Although existing taint analysis tools such as FlowDroid [Ar14] can be configured to conduct a relatively precise flow, context, and field-sensitive analysis, such configuration needs to be identified by possibly inexperienced users—and imprecise configuration causes the taint analysis to report substantial amounts of false positives [LBS18].

A recent approach by Grech et al. addresses this problem by extending static pointer analysis with information extracted from *heap snapshots*, collected at runtime. As the authors show, one can improve soundness [Gr17] by augmenting statically computed points-to information with *additional* data from the heap snapshots. Conversely, one can improve precision by *restricting* static points-to computation to such information present in the heap snapshots [Gr18].

In this work, we present an empirical study in which we seek to reproduce the original experiments revised by Grech et al. but also go significantly beyond them to address these open questions. We make the following original contributions:

Using heap snapshots for Android taint analysis. We investigate how heap snapshots impacts the soundness and precision, not just of simple pointer analysis, but of a concrete client analysis, a *static Android taint analysis.*

Assessment of design decisions. We investigate how various essential design decisions impact precision and soundness of the analysis. In particular, we evaluate the impact of two novel extensions:

- information not only from a *single* heap snapshot but *multiple ones,* e.g., collected at various times during the execution; and
- *dynamic* heap models collected at runtime (precise, but possibly unsound) versus pure *static* heap models (sound, but possibly imprecise) versus *mixed* models that seek to define a sensible middle ground between those two extremes by focusing on precision and enhancing a dynamic model with static information.

Implementation and Benchmark. To evaluate the above decisions, we implemented *Heapster*, an extension to FlowDroid that can incorporate heap dumps. Additionally, we created *DroidMacroBench*, a set of 12 real-world Android applications that we manually labeled with ground truth for taint analyses.

Evaluation. We explore the impact of different design decisions about *when to collect and how to consume heap snapshots*. In our evaluation we show that:

- adding heap snapshots can significantly improve the precision of taint analysis (from 50.3% to up to 94.7%);
- while restricting points-to information to that of the heap snapshots offers high precision it significantly harms recall. Our mixed mode solution, however, provides

both good precision (77.1%) and good recall (68.4%). Its F1 score is the highest among all configurations;

- in all evaluated scenarios, incorporating heap snapshots significantly lowers the amount of computational resources required by the taint analysis, moreover, in 90% of the scenarios it also improves the analysis performance; and
- while a single heap snapshot, taken at the end of the runtime, suffices to significantly increase the analysis precision, additional snapshots, taken at different times, are beneficial for the analysis recall, achieving the best overall F1 score.

For details please consider the full paper accessible at `https://dl.acm.org/doi/10.1145/3377811.3380438` and `https://www.hni.uni-paderborn.de/pub/10027`.

References

[Ar14] Arzt, S.; Rasthofer, S.; Fritz, C.; Bodden, E.; Bartel, A.; Klein, J.; Traon, Y. L.; Octeau, D.; McDaniel, P. D.: FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In: ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014. Pp. 259–269, 2014, URL: `http://doi.acm.org/10.1145/2594291.2594299`.

[En11] Enck, W.; Octeau, D.; McDaniel, P. D.; Chaudhuri, S.: A Study of Android Application Security. In: 20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings. USENIX Association, 2011, URL: `http://static.usenix.org/events/sec11/tech/full%5C_papers/Enck.pdf`.

[En14] Enck, W.; Gilbert, P.; Chun, B.-G.; Cox, L. P.; Jung, J.; McDaniel, P.; Sheth, A. N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones William. Communications of the ACM 57/3, pp. 99–106, 2014, ISSN: 00010782, arXiv: `1005.3014`, URL: `http://dl.acm.org/citation.cfm?doid=2566590.2494522`.

[Gr12] Grace, M. C.; Zhou, W.; Jiang, X.; Sadeghi, A.: Unsafe exposure analysis of mobile in-app advertisements. In (Krunz, M.; Lazos, L.; Pietro, R. D.; Trappe, W., eds.): Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WISEC 2012, Tucson, AZ, USA, April 16-18, 2012. ACM, pp. 101–112, 2012, URL: `https://doi.org/10.1145/2185448.2185464`.

[Gr17] Grech, N.; Fourtounis, G.; Francalanza, A.; Smaragdakis, Y.: Heaps don't lie: countering unsoundness with heap snapshots. PACMPL 1/OOPSLA, 68:1–68:27, 2017, URL: `https://doi.org/10.1145/3133892`.

[Gr18] Grech, N.; Fourtounis, G.; Francalanza, A.; Smaragdakis, Y.: Shooting from the heap: ultra-scalable static analysis with heap snapshots. Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2018/, pp. 198–208, 2018, URL: http://dl.acm.org/citation.cfm?doid=3213846.3213860.

[LBS18] Luo, L.; Bodden, E.; Späth, J.: A Qualitative Analysis of Taint-Analysis Results, tech. rep., Heinz Nixdorf Institute, Paderborn University, Aug. 2018.

[Ra15] Rasthofer, S.; Arzt, S.; Hahn, R.; Kohlhagen, M.; Bodden, E.: (In)Security of Backend-as-a-Service. In: blackhat europe 2015. Nov. 2015, URL: http://bodden.de/pubs/rah+15backend.pdf.

What Kind of Questions Do Developers Ask on Stack Overflow? A Comparison of Automated Approaches to Classify Posts Into Question Categories

Stefanie Beyer[1], Christian Macho[1], Massimiliano Di Penta[2], Martin Pinzger[1]

Abstract: Stack Overflow (SO) is among the most popular question and answers sites used by developers. Labeling posts with tags is one of the features to facilitate searching and browsing SO posts. However, existing tags mainly refer to technological aspects but not to the purpose of a question. In this paper, we argue that tagging posts with their purpose can facilitate developers to find the posts that provide an answer to their question. We first present a harmonization of existing taxonomies of question categories, that represent the purpose of a question, into seven categories. Next, we present two approaches to automate the classification of posts into the seven question categories, one using regular expressions and one using machine learning. Evaluating both approaches on an independent test set, we found that our regular expressions outperform machine learning. Applying the regular expressions on posts related to Android app development, showed that the categories API USAGE, CONCEPTUAL, and DISCREPANCY are most frequently assigned. By integrating our approach into SO, posts could be manually tagged with our categories which would allow developers to search posts by question category.

Keywords: Stack Overflow; Classification; Question Categories; Program Understanding

Many developers use question and answer forums, such as Stack Overflow (SO), to discuss and solve their development issues. To refine the search and describe the questions briefly, each question post on SO is labeled with 1 to 5 tags. These tags often describe technological aspects of the questions but lack to describe the motivation of the author which is necessary to understand the issue [Be17].

Given the number of questions that are newly posted each day, manually assigning such tags to the posts is considered no feasible. In this work, we set out to automate the process of labeling posts with tags that represent the *why* questions are asked (question categories). These tags are important to understand the most difficult aspects of software development and the usage of APIs [AS13].

We manually classified 1000 posts into seven question categories that we obtained by comparing taxonomies found by prior studies [AS13, Be17, BP14, RS15, TBS11]. Additionally, we marked 2.192 phrases that indicate a particular question category. Then, we used the manually created data set to supervise the automated classification of posts into question categories. First, we implemented a classifier based on regular expressions that we derived

[1] University of Klagenfurt, stefanie.beyer@aau.at
[2] University of Sannio, dipenta@unisannio.it

doi:10.18420/SE2021_03

by combining recurrent patterns in the phrases. Second, we trained machine learning (ML) models using Random Forest and Support Vector Machine on the phrases to classify the posts into the seven question categories.

We evaluated the performance of our approaches on an independent test set of 110 SO posts that were neither used to extract patterns for the regular expressions nor used to train and test the models before. The results showed that the regex approach achieves an average precision and recall of 0.90 and 0.90, respectively, which outperforms the ML approaches. Furthermore, the regex approach is much faster and easier to adapt. The application of the regex approach to all studied questions confirmed our findings that API USAGE, DISCREPANCY, and CONCEPTUAL are the most frequently occurring question categories. Furthermore, the results show that the majority of the posts is classified in one to three categories and that the categories are mostly not overlapping.

By integrating the regex classifier into SO, several improvements could be achieved. First, the automated nature of our approach can help to tag historical posts that still lack tags that describe non-technical aspects of the posts. Tagging existing posts automatically will increase the chances that historical posts will also be tagged with newly introduced tags. Second, we enable developers posting questions by applying our approach to their post draft and suggesting related question categories to improve the characterization of the posts through the assigned tags. Third, both of these applications consequently help to find appropriate posts when developers are searching for help on SO. Lastly, the question categories can be used to improve existing approaches, such as Seahawk and Prompter, that suggest suitable code snippets to provide more accurate postings.

Literaturverzeichnis

[AS13] Allamanis, M.; Sutton, C.: Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code. In: Proceedings of the Working Conference on Mining Software Repositories. IEEE, S. 53–56, May 2013.

[Be17] Beyer, S.; Macho, C.; Di Penta, M.; Pinzger, M.: Analyzing the Relationships between Android API Classes and their References on Stack Overflow. Technical report, University of Klagenfurt, University of Sannio, 2017.

[BP14] Beyer, S.; Pinzger, M.: A manual categorization of android app development issues on Stack Overflow. In: Proceedings of the International Conference on Software Maintenance and Evolution. IEEE, S. 531–535, 2014.

[RS15] Rosen, C.; Shihab, E.: What are mobile developers asking about? A large scale study using stack overflow. Empirical Software Engineering, 21:1–32, 2015.

[TBS11] Treude, C.; Barzilay, O.; Storey, M. A.: How Do Programmers Ask and Answer Questions on the Web? (NIER Track). In: Proceedings of the International Conference on Software Engineering. ACM, S. 804–807, 2011.

ModGuard: Identifying Integrity & Confidentiality Violations in Java Modules (Short Summary)

Andreas Dann[1] Ben Hermann[1] Eric Bodden[1,2]

Abstract: This short paper[3] presents a static analysis for the novel challenge of analyzing Java modules. Since modules have only been recently introduced with Java 9, we point out the impact of modules both from the security and the static code analysis perspective. In particular, we introduce a static analysis that allows developers to assess if a module successfully encapsulates internal data, along with a formal definition of a module's entrypoints.

Keywords: Static Code Analysis; Module System; Java 9

1 Overview

With the release of version 9, Java introduced the module system Jigsaw. It enables developers to explicitly declare which packages and types are exposed and which are internal [Or15].

Although modules can encapsulate internal types, they do not prevent the unintentional leak of security-sensitive data, e.g., secret keys, giving rise to integrity and confidentiality violations. Confining data by leveraging the module system requires reasoning about data flows between modules and which classes, methods, and fields are actually accessible from outside the module.

To complement Java's module system with means to identify unintended data leaks, we present ModGuard[4], a novel static analysis to identify escaping instances, fields, or methods. Further, we clarify if existing applications may benefit from the guarantees provided by the module system by conducting a case study on Apache Tomcat.

2 What are modules?

Like "traditional" JAR files, modules assemble related packages, native code, and resources. Additionally, modules further contain a static module descriptor file (`module-info.java`). A descriptor file specifies the module's unique name, the exported packages, and the other modules it requires.

Up to Java 8, every public class was visible to any other on the classpath. In Java 9, a class contained in a module (*java.desktop*) may only access another module's (*java.xml*) class if it requires that module, and the module exports the package [Or14] (cf. Figure 1).

[1] Heinz Nixdorf Institute, Paderborn University, Germany <firstname>.<lastname>@uni-paderborn.de
[2] Fraunhofer IEM, Germany
[3] The full-paper is available online [DHB19]
[4] `https://github.com/secure-software-engineering/modguard`

doi:10.18420/SE2021_04

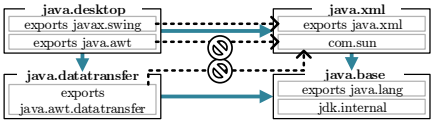


Fig. 1: Blue: Module Dep.; Dashed: Exported Pkg.

But *crucially instances* of internal types can still escape their module. Further, the methods and fields inherited from exported supertypes can be invoked, e.g., $java.desktop$ may invoke exported methods on instances escaping $com.sun$.

3 How to identify data leaks & escaping instances?

Identifying unintended data flows using static analyses on individual modules is challenging. The analysis must be conducted on open code much alike call-graph construction for libraries [Re16]: a module can be linked to any other, and the analysis must foresee all ways in which those other modules may invoke it.

To cope with this challenge, we define all potential interactions with a module in the form of a so-called entrypoint model using Datalog-based analysis rules extending Doop [SB11; SKB14]. The model distinguishes between *explicit* and *implicitly* entrypoints. Explicit entrypoints are methods whose declaring type is exported and can be invoked directly. Also, they may grant access to the so-called *implicit* entrypoints. Implicit entrypoints are methods that inherit, implement, or override methods of exported supertypes but are declared by an internal type whose instances may escape.

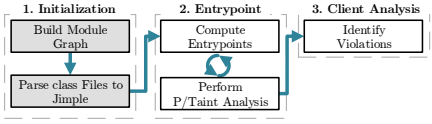


Fig. 2: ModGuard's Analysis Steps.

Based on the entrypoint model, we designed the analysis ModGuard (cf. Figure 2). After computing the entrypoints, ModGuard checks which fields, returned values, and classes became accessible as a result of invoking of the entrypoints by computing their points-to set. To identify violations, ModGuard intersects the points-to set with the point-to set of security-sensitive types and fields. If the intersection is non-empty, ModGuard reports a violation.

4 Can Modules help to confine data?

To clarify if applications may benefit from the guarantees provides by the module system, we exemplary studied Apache Tomcat 8.5.21. Since Tomcat not yet uses modules, we migrated every JAR to a module, following Corwin et al. [Co03]. Our case study shows that such a naïve migration fails to mitigate confidentiality and integrity violations, as ModGuard found violations in 12 out of 26 Tomcat modules.

5 Conclusion

ModGuard may help developers to leverage the module system security-wise by identifying the exposure of security-sensitive data or objects. The Apache Tomcat example shows that to confine sensitive data successfully, developers must introduce modules with care.

References

[Co03] Corwin, J.; Bacon, D. F.; Grove, D.; Murthy, C.: MJ: a rational module system for Java and its applications. In: OOPSLA '03 Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications. Vol. 38, ACM, pp. 241–254, 2003, ISBN: 1581137125, URL: http://doi.acm.org/10.1145/949343.9493.

[DHB19] Dann, A.; Hermann, B.; Bodden, E.: ModGuard: Identifying Integrity Confidentiality Violations in Java Modules. IEEE Transactions on Software Engineering/, pp. 1–1, 2019, URL: http://dx.doi.org/10.1109/TSE.2019.2931331.

[Or14] Oracle Corporation: JEP 261: Module System, 2014, URL: http://openjdk.java.net/jeps/261.

[Or15] Oracle Corporation: JEP 260: Encapsulate Most Internal APIs, 2015, URL: http://openjdk.java.net/jeps/260.

[Re16] Reif, M.; Eichberg, M.; Hermann, B.; Lerch, J.; Mezini, M.: Call Graph Construction for Java Libraries. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. FSE 2016, ACM, Seattle, WA, USA, pp. 474–486, 2016, ISBN: 978-1-4503-4218-6, URL: http://doi.acm.org/10.1145/2950290.2950312.

[SB11] Smaragdakis, Y.; Bravenboer, M.: Using Datalog for Fast and Easy Program Analysis. In: Proceedings of the First International Conference on Datalog Reloaded. Datalog'10, Springer-Verlag, Oxford, UK, pp. 245–251, 2011, ISBN: 978-3-642-24205-2, URL: http://dx.doi.org/10.1007/978-3-642-24206-9_14.

[SKB14] Smaragdakis, Y.; Kastrinis, G.; Balatsouras, G.: Introspective Analysis: Context-sensitivity, Across the Board. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI '14, ACM, Edinburgh, United Kingdom, pp. 485–495, 2014, ISBN: 978-1-4503-2784-8, URL: http://doi.acm.org/10.1145/2594291.2594320.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 33

Verbesserung manueller Validierungsprozesse von CPS-Spezifikationen durch Review-Modelle auf Instanzebene

Marian Daun[1], Jennifer Brings[2], Thorsten Weyer[3]

Abstract: Dieser Vortrag berichtet von dem Beitrag *Do Instance-level Review Diagrams Support Validation Processes of Cyber-Physical System Specifications: Results from a Controlled Experiment* [DBW20], der auf der IEEE/ACM International Conference on Software and System Processes (ICSSP) 2020 vorgestellt und in dem Konferenzband veröffentlicht wurde. Im Rahmen des Beitrags wurde untersucht, inwiefern manuelle Validierungsprozesse für cyber-physische Systeme durch die Erstellung von Review-Modellen auf Instanzebene verbessert werden können. Mit einem Experiment konnte gezeigt werden, dass die Wahl des Review-Modells (d.h., ob ein Modell auf Typebene oder auf Instanzebene untersucht wird) Auswirkungen auf die Qualität der Inspektion hat.

Keywords: Manuelle Validierung; Modellbasierte Entwicklung; Experiment

1 Einleitung

Inspektionen von Entwicklungsartefakten sind ein bedeutender Bestandteil der Qualitätssicherung für sicherheitskritische Systeme. In der Vergangenheit wurde hierzu untersucht, ob modellbasierte Spezifikationen von der Erzeugung dedizierter Review-Modelle profitieren können [DWP19, Da15b, Da19]. Hier haben sich insbesondere Spezifikationen in Notation von Message Sequence Charts als vorteilhaft erwiesen.

Bei der Validierung kollaborativer cyber-physische Systeme (CPS) ergeben sich neue Herausforderungen. Kollaborative CPS bilden Systemverbünde, um Ziele zu erreichen, die Einzelsysteme nicht erreichen können. Bspw. kann durch den Zusammenschluss mehrerer Fahrzeuge zu einem Platoon eine zusätzliche CO_2-Reduktion erzielt werden. Durch die damit verbundene Eigenschaft der Offenheit können sich zahlreiche unterschiedliche kollaborative CPS zur Laufzeit zusammenschließen. Dadurch entsteht eine Vielzahl von Konfigurationen des Systemverbunds, die bei der Entwicklung und insbesondere bei der Validierung zu berücksichtigen sind. Die Betrachtung des Interaktionsverhaltens auf Typebene wird erschwert, da die vielen möglichen Konfigurationen in Abhängigkeit von den jeweils beteiligten Instanzen unterschiedliches Verhalten aufweisen und auch aufweisen sollen [Da15a]. Hierdurch gestaltet sich die Validierung dieses Interaktionsverhaltens schwierig.

[1] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, 45127 Essen, Deutschland
marian.daun@paluno.uni-due.de
[2] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, 45127 Essen, Deutschland
jennifer.brings@paluno.uni-due.de
[3] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, 45127 Essen, Deutschland
thorsten.weyer@paluno.uni-due.de

doi:10.18420/SE2021_05

2 Beitrag

Hierzu haben wir untersucht, ob Review-Modelle auf Instanzebene oder auf Typebene besser geeignet sind, um die Validierung kollaborativer CPS zu unterstützen.

In unserem Beitrag haben wir mit einem Experiment gezeigt, dass in diesen Fällen die Inspektion der Typmodelle nicht ausreichend ist, da eben diese kombinatorischen Konfigurationen nicht abgebildet werden. Des Weiteren ist die systematische Erzeugung von Review-Modellen auf Instanzebene geeignet, um Validierungsprozesse adäquat zu unterstützen. Es hat sich herausgestellt, dass die Nutzung von Instanzmodellen ausdrucksstärker und effektiver ist als die Nutzung von Typmodellen. Darüber hinaus werden Instanzmodelle auch vom Validierer als unterstützender wahrgenommen. Unterschiede ergeben sich bezüglich der Größe des Instanzmodells, d.h., wie viele Instanzen eines Typs abgebildet werden.

Literaturverzeichnis

[Da15a] Daun, M.; Brings, J.; Bandyszak, T.; Bohn, P.; Weyer, T.: Collaborating Multiple System Instances of Smart Cyber-physical Systems: A Problem Situation, Solution Idea, and Remaining Research Challenges. In: 2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems. S. 48–51, 2015.

[Da15b] Daun, Marian; Salmon, Andrea; Weyer, Thorsten; Pohl, Klaus: The Impact of Students' Skills and Experiences on Empirical Results: A Controlled Experiment with Undergraduate and Graduate Students. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. EASE '15, ACM, New York, NY, USA, S. 29:1–29:6, 2015.

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

[DBW20] Daun, Marian; Brings, Jennifer; Weyer, Thorsten: Do Instance-level Review Diagrams Support Validation Processes of Cyber-Physical System Specifications: Results from a Controlled Experiment. In: ICSSP '20: International Conference on Software and System Processes, Seoul, Republic of Korea, 26-28 June, 2020. ACM, S. 11–20, 2020.

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

#ifdef Directives and Program Comprehension: The Dilemma between Correctness and Preference

Wolfram Fenske[1] Jacob Krüger[2] Maria Kanyshkova[2] Sandro Schulze[2]

Abstract: In this extended abstract, we summarize our paper with the homonymous title published at the International Conference on Software Maintenance and Evolution (ICSME) 2020 [Fe20].

Keywords: Configurable Systems; Preprocessors; Program Comprehension; Refactoring; Empirical Study

The C PreProcessor (CPP) is a simple, yet effective tool to implement configuration options in a software system. For this purpose, the CPP provides text-based directives to enable conditional compilation, following the annotate-and-remove paradigm. Each directive is associated with a macro (i.e., the configuration option), which controls the presence or absence of the source code surrounded by its opening (e.g., #ifdef) and closing (e.g., #endif) directives. Due to its simplicity, the CPP is widely used in industrial and open-source systems from various domains—prominent examples being the Linux Kernel with over 26 million lines of code and more than 15 thousand configuration options, Hewlett-Packard's printer firmware, and the Apache web server. The CPP allows developers to customize such systems to specific customer requirements, safety regulations, resource restrictions, or non-functional properties.

While the CPP is established in practice, it is also heavily criticized for several issues perceived as problematic. For instance, researchers suspect that the CPP impedes program comprehension, fosters code scattering as well as tangling, harms maintainability, and increases fault proneness. The most prominent issue that has been investigated in greater detail are *undisciplined CPP directives*, that is, directives that are not aligned with syntactic units in the source code. However, some studies on the CPP led to contradicting results and most studies are limited in their validity, for example, because they exclusively rely on automated repository mining or because controlled experiments involved mostly a smaller number of students. Only two previous experiments (one on undisciplined directives, one on faults) involved a larger number of experienced practitioners.

In our paper, we present a large-scale empirical study on the impact of refactoring CPP directives to be more comprehensible. We selected five real-world code example from `Emacs` and `Vim` that previous work indicates to be particularly "smelly" (i.e., hard to comprehend). Building on findings of previous studies, we employed three types of refactoring to improve

[1] pure-systems GmbH, Magdeburg, Germany, Email: wolfram.fenske@pure-systems.com
[2] Otto-von-Guericke-University Magdeburg, Germany, Email: jkrueger@ovgu.de, sanschul@ovgu.de

doi:10.18420/SE2021_06

the comprehensibility of the source code by reducing the complexity of the present CPP directives. We then designed an online study that comprised an experiment and a survey, combining objective and subjective empirical data for the same code examples—which has not been done in previous work that always focused on either experimental or survey data. Our study was split into two different versions, both sharing one code example to assure that we could compare between the versions. Moreover, each version comprised two original and two refactored code examples. For each example, our participants solved two program comprehension tasks (i.e., the experiment) during which we measured their objective correctness. Afterwards, we asked them to assess the quality of the code and CPP directives (i.e., the survey) to elicit their subjective opinion. We sent our study to 7,791 C developers of open-source projects hosted on GitHub who made their data publicly available. Overall, we received 521 responses with an almost even split between the two study versions (i.e., 260 to 261). Using this methodology, we considerably extend previous studies by combining objective and subjective measurements in a large-scale study.

The core findings we derive from our results are:

- Our participants *performed slightly worse on refactored code* in terms of correctly solving the defined program-comprehension tasks, despite existing evidence suggesting that the refactoring should have improved their program comprehension.
- Our participants *preferred the refactored CPP directives* over those in the original code examples, aligning with existing evidence.
- Most interestingly, our *participants' objective comprehension performance and their subjective preferences contradict each other and existing evidence* on the comprehensibility and refactoring of CPP directives.
- Refactoring CPP directives may result in developers perceiving the overall code quality as worse, indicating a *trade-off between the quality of the CPP directives and the quality of the underlying source code*.

Overall, our results imply a surprising dilemma not covered by previous studies, challenging common beliefs in the context of program comprehension of CPP directives. In our future work, we will investigate this dilemma between objective performance and subjective preference in more detail using further empirical research methods.

Bibliography

[Fe20] Fenske, Wolfram; Krüger, Jacob; Kanyshkova, Maria; Schulze, Sandro: #ifdef Directives and Program Comprehension: The Dilemma between Correctness and Preference. In: International Conference on Software Maintenance and Evolution. ICSME. IEEE, pp. 255–266, 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 37

SPECMATE: Automated Creation of Test Cases from Acceptance Criteria

Jannik Fischbach[1], Andreas Vogelsang[2], Dominik Spies[3], Andreas Wehrle[4], Maximilian Junker[5], Dietmar Freudenstein[6]

Abstract: We summarize the paper *Specmate: Automated Creation of Test Cases from Acceptance Criteria* [Fi20b], which was presented at the 2020 edition of the IEEE International Conference on Software Testing, Verification and Validation (ICST).

Keywords: test case creation; natural language processing; model-based testing; user stories; agile software development

1 Introduction

User stories are an established instrument for the notation of system requirements in agile software projects. A user story is fulfilled if all specified acceptance criteria (AC) are satisfied. This requires testing the defined AC by creating, executing, and maintaining both positive and negative test cases (*Acceptance Testing*). Test case design is a very laborious activity that easily accounts for 40-70 % of the total effort in the testing process: First, the right input-output combinations need to be determined to comprehensively test the requirement, which is not trivial, especially for complex system requirements [Fi20a]. Secondly, the number of test cases should be kept to a minimum to avoid unnecessary testing efforts. Furthermore, the creation of test cases has to be mostly done manually since there is a lack of tool support. Existing approaches support the test case generation from formal and semi-formal requirements, but are not suitable for informal requirement descriptions based on unrestricted natural language. Unrestricted natural language, however, is the dominant way of formulating AC, as we found in the analysis of 961 user stories from two projects together with our industry partner Allianz Deutschland. We address this research gap and present SPECMATE as an approach to reduce the manual effort of deriving test cases from AC by applying *Natural Language Processing* (NLP).

[1] Qualicen GmbH, Munich, Germany, jannik.fischbach@qualicen.de
[2] University of Cologne, Germany, vogelsang@cs.uni-koeln.de
[3] Qualicen GmbH, Munich, Germany, dominik.spies@qualicen.de
[4] Qualicen GmbH, Munich, Germany, andreas.wehrle@qualicen.de
[5] Qualicen GmbH, Munich, Germany, maximilian.junker@qualicen.de
[6] Allianz Deutschland AG, Munich, Germany, dietmar.freudenstein@allianz.de

doi:10.18420/SE2021_07

2 Our Approach

We argue that a valuable automated solution for generating test cases from user stories and their AC can only be created by understanding both their content and form. For this purpose, we analyze 961 user stories provided by our industrial partner to determine requirements for the automated approach. Based on these requirements, we design an approach based on NLP to generate corresponding test cases automatically. We follow the idea of *Model-Based-Testing* and introduce an intermediate layer between the user stories and the final test cases. We extract the AC from the user stories and transfer each into a test model. Since we found in our case study that the expected system behavior is usually described in the form of cause and effect relationships (e.g. *In the case of <cause>, the system shall <effect>*), we use *Cause-Effect-Graphs* (CEG) as test models. In order to transfer the AC into a CEG, the relevant causes and effects must be identified within the AC. For this purpose we apply *Dependency Parsing* and first convert each AC into a dependency tree. Subsequently, we traverse the dependency tree and generate the CEG. Finally, we derive the minimum number of test cases from the CEG by applying the *Basic Path Sensitization Technique*.

3 Our Results

Our case study demonstrates that not every user story provides functional information to generate test cases. Depending on the project, user stories are increasingly used as a means of communication. In contrast, about 31.1 % to 50.1 % of the observed user stories describe the functionality by AC. We hypothesize that there is a high automation potential in test case derivation from these *functional user stories*. In this context, a major challenge arises in processing the informal nature of the AC, which is the dominant type of notation. Despite the use of unstructured language, the majority of AC are characterized by recurring patterns, of which cause-effect-relationships have the broadest application. We evaluated SPECMATE based on 604 test cases that have been manually derived from 72 user stories by test designers from our industry partner. Our experiments underline the practical benefits of SPECMATE. 56 % of the manually created test cases could be generated automatically and missing negative test cases are added. The missing 44 % stems from required domain knowledge and poor data quality within the AC. We hypothesize that a full automation of the test creation from AC can hardly be achieved. Our approach should therefore be seen as a supplement to the existing manual process.

Bibliography

[Fi20a] Fischbach, J.; Femmer, H.; Mendez, D.; Fucci, D.; Vogelsang, A.: What Makes Agile Test Artifacts Useful? An Activity-Based Quality Model from a Practitioners' Perspective. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). 2020.

[Fi20b] Fischbach, J.; Vogelsang, A.; Spies, D.; Wehrle, A.; Junker, M.; Freudenstein, D.: SPECMATE: Automated Creation of Test Cases from Acceptance Criteria. In: IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 39

Automated Reuse of Test Cases for Highly Configurable Software Systems

Stefan Fischer[1], Gabriela Karoline Michelon[2], Rudolf Ramler[3], Lukas Linsbauer[4], Alexander Egyed[5]

Abstract: In this work, we report about our research results [Fi20] initially published in the journal *Empirical Software Engineering, volume 25, issue 6, pp. 5295–5332, November 2020*, https://doi.org/10.1007/s10664-020-09884-x. We performed experiments on test reusability across configurations of highly configurable software systems. First, we used manually written tests for specific configurations of three configurable systems and investigated how changing configuration options affects these tests. Subsequently, we applied an approach developed for automated reuse, ECCO (Extraction and Composition for Clone-and-Own), to automatically generate tests for new configurations from the existing, manually written tests. The experiments showed that our generated tests had a higher or equal success rate compared to direct reuse and they generally achieved a higher code coverage. It can be concluded that automating the reuse of tests for configurable software can substantially reduce the effort for adapting existing tests and it supports a rigorous testing process.

Keywords: Variability; Configurable software; Clone-and-own; Reuse; Testing

1 Summary

Many large software systems are designed to be highly configurable with hundreds or thousands of configuration options. They introduce the flexibility to adapt the system to specific customer needs. However, some combinations of configuration options may result in undesired interactions, i.e., one option having unintended negative side effects on the behavior associated with another configuration option. Testing configurations for interactions is inherently challenging, because a large number of configuration options means that there are often myriads of possible configurations that can be derived, each showing a potentially different behavior. Combinatorial Interaction Testing (CIT) selects configurations that cover combinations of n configuration options (often referred to as n-wise testing) to reduce the amount of configurations to test to a viable number [Lo15]. Nevertheless, the actual test cases still need to be adapted to reflect the individual behavior

[1] Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria, stefan.fischer@scch.at
[2] Johannes Kepler University, Institute for Software Systems Engineering and LIT Secure and Correct Systems Lab, Linz, Austria, gabriela.michelon@jku.at
[3] Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria, rudolf.ramler@scch.at
[4] Technische Universit¨at Braunschweig, Institute of Software Engineering and Automotive Informatics, Braunschweig, Germany, l.linsbauer@tu-braunschweig.de
[5] Johannes Kepler University, Institute for Software Systems Engineering, Linz, Austria, alexander.egyed@jku.at

doi:10.18420/SE2021_08

of each selected configuration, which can result in significant extra effort for testing. Thus, Krüger et al. discussed the need for automated test refactoring for the adoption of more systematic reuse approaches [Kr18]. In the presented work, we performed experiments on test reuse across configurations of configurable software systems (RQ1) and investigate how automated reuse affects the testing effort and the resulting coverage (RQ2).

The focus of our work is on reusing existing (e.g., manually written) test cases dedicated to specific configurations of the system. These specific test cases are reused for testing other, so far untested configurations combining previously tested configurations. We applied an automated approach for reuse, ECCO (Extraction and Composition for Clone-and-Own) [Fi14], to automatically generate new variants of tests from existing ones. ECCO dissects the existing tests according to the related configuration options and combines the obtained fragments to new test cases matching the options of the new configurations under test. For our experiments, we used three different configurable systems: *StackSPL*, *ArgoUML*, and *Bugzilla* (version 3.4 and 5.1). We investigated *direct reuse* of existing tests and *automated reuse* by generating tests for new configurations, which were obtained by pairwise and three-wise combination of configuration options. The measures used for evaluation were *success rate* (i.e., the number of test cases that passed on a new configuration without adaption) and *code coverage* (as well as mutation testing for one of the systems).

Our experiments showed that a large proportion of the existing test cases could be reused on new configurations. For our first two systems, nearly all individual test variants could be directly reused, and for the two versions of Bugzilla from 70% to even 100% of tests cases could be reused. Automatically reusing tests yielded better results in success rate (avg. improvement 27.8%) and code coverage (avg. improvement 5-10.1%). These results suggest a considerable advantage for applying automated test reuse over the direct reuse, which requires additional manual effort for adapting failing test cases.

Literaturverzeichnis

[Fi14] Fischer, Stefan; Linsbauer, Lukas; Lopez-Herrejon, Roberto Erick; Egyed, Alexander: Enhancing clone-and-own with systematic reuse for developing software variants. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, S. 391–400, 2014.

[Fi20] Fischer, Stefan; Michelon, Gabriela Karoline; Ramler, Rudolf; Linsbauer, Lukas; Egyed, Alexander: Automated test reuse for highly configurable software. Empirical Software Engineering, S. 1–38, 2020.

[Kr18] Krüger, Jacob; Al-Hajjaji, Mustafa; Schulze, Sandro; Saake, Gunter; Leich, Thomas: Towards automated test refactoring for software product lines. In: Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1. S. 143–148, 2018.

[Lo15] Lopez-Herrejon, Roberto E; Fischer, Stefan; Ramler, Rudolf; Egyed, Alexander: A first systematic mapping study on combinatorial interaction testing for software product lines. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, S. 1–10, 2015.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 41

Robotics Software Engineering: A Perspective from the Service Robotics Domain (Summary)

Sergio García [1], Daniel Strüber [2], Davide Brugali [3], Thorsten Berger [4], Patrizio Pelliccione [5]

Abstract: We present our paper published in the proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering 2020.
Robots that support humans by performing useful tasks (a.k.a., service robots) are booming worldwide. In contrast to industrial robots, the development of service robots comes with severe software engineering challenges, since they require high levels of robustness and autonomy to operate in highly heterogeneous environments. As a domain with critical safety implications, service robotics faces a need for sound software development practices. In this paper, we present the first large-scale empirical study to assess the state of the art and practice of robotics software engineering. We conducted 18 semi-structured interviews with industrial practitioners working in 15 companies from 9 different countries and a survey with 156 respondents (from 26 countries) from the robotics domain. Our results provide a comprehensive picture of (i) the practices applied by robotics industrial and academic practitioners, including processes, paradigms, languages, tools, frameworks, and reuse practices, (ii) the distinguishing characteristics of robotics software engineering, and (iii) recurrent challenges usually faced, together with adopted solutions. The paper concludes by discussing observations, derived hypotheses, and proposed actions for researchers and practitioners.

Keywords: robotics; software engineering; interview study; questionnaire study

1 Introduction

Service robots are a rising robotics domain with broad applications in many fields, such as logistics, healthcare, telepresence, maintenance, domestic tasks, education, and entertainment. Service robots are robots that performs useful tasks for humans or equipment (excluding industry automation robots). Compared to industrial robotics, the service robotics domain is more challenging, since these robots usually operate in unconstrained environments, often populated by humans, requiring high degrees of robustness and autonomy.

Despite software playing an ever-increasing role in robotics, the current software engineering (SE) practices are perceived as insufficient, often leading to error-prone and hardly maintainable and evolvable software. Robotic systems are an advanced type of cyber-physical sys-

[1] Chalmers | University of Gothenburg, Sweden sergio.garcia@gu.se
[2] Radboud University, Nijmegen, Netherlands d.strueber@cs.ru.nl
[3] University of Bergamo, Bergamo, Italy davide.brugali@unibg.it
[4] Chalmers | University of Gothenburg, Sweden thorsten.berger@gu.se
[5] Chalmers | University of Gothenburg, Sweden and University of L'Aquila, L'Aquila, Italy patrizio.pelliccione@gu.se

doi:10.18420/SE2021_09

tem (CPS), made up of an intricate blend of hardware, software, and environmental components. SE, despite its beneficial role in other CPS domains (e.g., automotive, aeronautics), has traditionally been considered an auxiliary concern of robotic system construction. A possible reason is that robots in factory automation have built-in proprietary controllers for repetitive tasks, therefore, allowing a simple programming style. The heavy lifting is in the domains of mechanics, electronics, and automatic control. In contrast, to achieve autonomy when interacting in highly heterogeneous environments, service robots are equipped with a large variability of functionalities for perception, control, planning, learning, and multimodal interaction with the human operator [Ga19]. The integration, customization, and evolution of these functionalities give rise to a large amount of complexity, the management of which is a challenging task. SE systematic practices could play a crucial role in the management of such complexity.

Systematic studies about the specific software development practices and tools applied in service robotics as well as the challenges faced by practitioners in this domain are currently lacking. Towards this goal, our paper [Ga20] assesses the current SE practices applied to the domain of service robotics, as well as its distinguishing characteristics and faced challenges. To collect data, we conducted 18 semi-structured interviews with industrial robotics experts working for 15 companies from 9 different countries. We accompany this study with an online survey, targeting industrial and academic practitioners in the robotics domain, from which we collect 156 responses. To the best of our knowledge, our study is the first with this ambition.

Highlights from our results include the following observations: We discovered that roboticists are predominantly focused on implementation and real-world testing, often favored over simulation. We learned that robotic control systems are typically developed as component-based systems, implemented by developers who may come from different backgrounds (e.g., mechanical, electrical, or software engineers). We also elicited the main characteristics of robotics SE, where the cyber-physical nature of robots and the variety of disciplines required to develop a complete robotic system were highlighted. These characteristics increase the complexity of robots' control software, calling for systematic practices as modeling and the usage of software architectures to improve the development process. Our respondents ranked challenges related to robustness and validation as most pressing, and typically address them by applying thorough testing processes. Based on our observations, we also identify research themes that deserve further investigation. We provide conjectures for why these themes are currently under-investigated and recommendations for both researchers and practitioners.

Literaturverzeichnis

[Ga19] García, Sergio; Strüber, Daniel; Brugali, Davide; Di Fava, Alessandro; Schillinger, Philipp; Pelliccione, Patrizio; Berger, Thorsten: Variability Modeling of Service Robots: Experiences and Challenges. In: VaMoS. ACM, S. 8, 2019.

[Ga20] García, Sergio; Strüber, Daniel; Brugali, Davide; Berger, Thorsten; Pelliccione, Patrizio: Robotics Software Engineering: A Perspective from the Service Robotics Domain. In: ESEC/FSE. S. 593–604, 2020.

Component-Based Refinement and Verification of Information-Flow Security Policies for Cyber-Physical Microservice Architectures

Christopher Gerking [1], David Schubert[2]

Abstract: This publication is based on our paper presented at the IEEE International Conference on Software Architecture 2019 [GS19].

Keywords: security policy; information flow; microservice architecture; cyber-physical systems

1 Composable Security for Cyber-Physical Systems

Cyber-physical systems (CPS) are closely interconnected with the outside world, exchanging information with different parties. From a security viewpoint, it is therefore crucial for software engineers to ensure that confidential information is never leaked to unauthorized third parties. To protect CPS against such security leaks, the flow of information must be regulated and analyzed in the early design phase. Formal methods for regulation and analysis are provided by the theory of *information-flow security*. Due to the popularity of component-based design principles (e.g., such as the *microservice* architectural style), the software of CPS is increasingly composed of multiple components. Thus, each component must be provided with an individual security policy that regulates the flow of information between the component's interfaces. To satisfy the security regulations of the composite system, these policies must be composable in a way that prevents unauthorized information flows from end to end.

However, the composability of properties like security is a problem that requires careful investigation by software engineers. Over the last decade, secure information flow has become known as a so-called *hyperproperty* [CS10]. Due to their formal characteristics, such properties are hard to compose in general. Therefore, a careless composition of secure components is at risk of leading to an insecure system [Ma02]. In our publication [GS19], we provided software engineers with means to ensure the composability of information-flow security in the presence of domain-specific CPS characteristics. Thereby, we enable microservice architectures of CPS to be composed securely. Our contributions address both regulation and analysis of the information flow.

[1] Karlsruhe Institute of Technology (KIT), Software Design and Quality, Am Fasanengarten 5, 76131 Karlsruhe, Germany, christopher.gerking@kit.edu, https://orcid.org/0000-0001-5531-9607
[2] Fraunhofer Institute for Mechatronic Systems Design (IEM), Software Engineering and IT Security, Zukunftsmeile 1, 33102 Paderborn, Germany, david.schubert@iem.fraunhofer.de

doi:10.18420/SE2021_10

2 Refining Information-Flow Regulations

As our first contribution, we address the refinement of security regulations during the decomposition of systems into components. To this end, we provide software engineers with a set of architectural well-formedness rules for the security policies of components. These rules are used to distinguish regular refinements from those that are irregular because they put the composite system at risk of unauthorized information flows. Our work is based on a well-founded theory of composability for information-flow properties [Ma02], but applies these theoretical foundations to the engineering practice at the architectural level. To ensure applicability in the domain of CPS, our rule set is tailored to the asynchronous communication between components, exchanging information by passing messages to each other. Applying our rules ensures that, as long as all constituent components adhere to their refined security policies, the composite system is free of any information leaks.

3 Timing-Sensitive Analysis of the Information Flow

Our second contribution is a tool-supported verification technique that enables engineers to analyze the message-passing behavior of a component for unauthorized information flows. To account for the fact that CPS are real-time systems, our technique is timing-sensitive. Thus, it detects so-called *timing channels*, which are information leaks that are exploitable by observing the instant of time at which messages are passed. In combination, our contributions enable a compositional security analysis, in which the analysis results are securely composable out of the box [Ge20]. Thereby, we assure software engineers that a composition of secure components is riskless because it will always lead to a secure system. To evaluate the accuracy of our contributions, we conducted a security-related extension of the community case study CoCoME for component-based systems.

References

[CS10] Clarkson, M. R.; Schneider, F. B.: Hyperproperties. Journal of Computer Security 18/6, pp. 1157–1210, 2010, DOI: 10.3233/JCS-2009-0393.

[Ge20] Gerking, C.: Model-driven information flow security engineering for cyber-physical systems, PhD thesis, Paderborn University, 2020, DOI: 10.17619/UNIPB/1-1033.

[GS19] Gerking, C.; Schubert, D.: Component-Based Refinement and Verification of Information-Flow Security Policies for Cyber-Physical Microservice Architectures. In: IEEE International Conference on Software Architecture, Proceedings. ICSA 2019. IEEE, pp. 61–70, 2019, DOI: 10.1109/ICSA.2019.00015.

[Ma02] Mantel, H.: On the Composition of Secure Systems. In: 2002 IEEE Symposium on Security and Privacy, Proceedings. IEEE Computer Society, pp. 88–101, 2002, DOI: 10.1109/SECPRI.2002.1004364.

Claimed Advantages and Disadvantages of (dedicated) Model Transformation Languages: A Systematic Literature Review

Stefan Götz[1], Matthias Tichy[1], Raffaela Groner[1]

Abstract: There exists a plethora of claims about the advantages and disadvantages of model transformation languages compared to general purpose programming languages. With our work, published at the Software and Systems Modelling Journal in 2020 [GTG20], we aim to create an overview over these claims in literature and systematize evidence thereof. For this purpose we conducted a systematic literature review by following a systematic process for searching and selecting relevant publications and extracting data. We selected a total of 58 publications, categorized claims about model transformation languages into 14 separate groups and conceived a representation to track claims and evidence through literature. From our results we conclude that: (i) current literature claims many advantages of model transformation languages but also points towards certain deficits and (ii) there is insufficient evidence for claimed advantages and disadvantages and (iii) a lack of research interest into the verification of claims.

Keywords: Model Transformation Language; DSL; Model Transformation; MDSE; advantages; disadvantages; SLR

Ever since the dawn of Model-Driven Engineering at the beginning of the century, model transformations, supported by dedicated transformation languages [Hi13], have been an integral part of model-driven development. Model transformation languages (MTLs) have ever since been associated with advantages for the development of model transformations compared to general purpose programming languages. Many of these advantages are reiterated time and time again throughout literature often without any actual evidence to support the statements. This makes it difficult for readers to grasp which properties of MTLs are verifiably true and which might still only be of visionary nature. The **goal** of our study is to identify and categorize claims about advantages and disadvantages of model transformation languages made throughout the literature and to gather available evidence thereof. For this purpose we performed a systematic review [BSP16] of claims about model transformation languages and evidence for those claims in literature. Lastly, we analysed and discussed the extracted claims and evidence to: (i) provide an overview over claimed advantages and disadvantages and their origin, (ii) the current state of evidence thereof and (iii) identify areas where further research is necessary.

Of the 58 selected publication *32* publications mention advantages and *36* publications mention disadvantages. Moreover *four* publications provide empirical evidence for either advantages or disadvantages while *12* publications use citations to support their claims and *14*

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

doi:10.18420/SE2021_11

publications use other means such as examples and experience of the authors to back up their statements. In total we were able to extract 127 claims about model transformation languages which we were able to sort into 14 different categories. The categories include properties that are expected to show up such as *performance*, *expressiveness* and *comprehensibility* but also less obvious groups such as *verification* or *versatility*. An effort by us to categorize the extracted claims along existing taxonomies of model transformation language features failed because ~70% of all claims are made broadly and without reference to specific features of MTLs that aid the advantage or disadvantage. Regarding the state of evidence we found that current literature exhibits a deficit in evidence (empirical or otherwise) for asserted properties of MTLs. We also identified several potential reasons and barriers for why current literature lacks evidence. First there is the fact that designing and conducting rigorous studies to examine model transformation languages requires a substantial amount of time and effort. A fact that is further hampered by the lack of easily available study subjects. Next is the fallacy that because MTLs are DSLs they bear the same advantages that other DSLs exhibit. We believe that the benefits attributed to DSLs can only point to potential advantages that MTLs may have, rather than being a certainty. Lastly there is the effect, that statements can become 'established' facts by virtue of being reiterated often enough or being cited multiple times without the cited source actually providing any evidence. This can lead to a distorted factual picture and makes it often impossible to find the origin of a claim.

We conclude that: (i) current literature claims many advantages of MTLs but also points towards deficits owed to the mostly experimental nature of the languages and its limited domain, (ii) there is insufficient evidence for and (iii) research about properties of model transformation languages. Our results suggest a lack of effort put into the evaluation of model transformation languages and their potential advantages or disadvantages. We believe that a significant portion of current research efforts that are being invested into the development of new features should instead be spent on evaluating the state of the art in hopes of ascertaining both what current MTLs are lacking most and where their strengths really lie. We also suggest that in future publications, claims on benefits and disadvantages of model transformation languages be made more specific and include mentions of the features that aid or hamper the benefits. This will allow a more nuanced and focused discussion of the issue and make statements less vulnerable to basic scrutiny.

Bibliography

[BSP16] Boot, Andrew; Sutton, Anthea; Papaioannou, Diana: Systematic Approaches to a Successful Literature Review. Sage, 2016.

[GTG20] Götz, Stefan; Tichy, Matthias; Groner, Raffaela: Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review. Software and Systems Modeling, 2020.

[Hi13] Hinkel, Georg: An approach to maintainable model transformations with an internal DSL. PhD thesis, National Research Center, 2013.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 47

Learning Circumstances of Software Failures

Rahul Gopinath[1], Nikolas Havrikov[2], Alexander Kampmann[3], Ezekiel Soremekun[4], Andreas Zeller[5]

Abstract: A program fails. Under which circumstances does the failure occur? Starting with a single failure-inducing input ("The input `((4))` fails") and an input grammar, this talk presents two techniques that use systematic tests to automatically determine the circumstances under which the failure occurs. The DDSET algorithm [Go20] generalizes the input to an _abstract failure-inducing input_ that contains both (concrete) terminal symbols and (abstract) nonterminal symbols from the grammar—for instance, "(($\langle$expr$\rangle$))", which represents any expression in double parentheses. The ALHAZEN technique [Ka20] takes this even further, using decision trees to learn *input properties* such as length or numerical values associated with failures: "The error occurs as soon as there are two parentheses or more." Such abstractions can be used as *debugging diagnostics*, characterizing the circumstances under which a failure occurs; and as *producers* of additional failure-inducing tests to help design and validate fixes and repair candidates. Both have the potential to significantly boost speed and quality of software debugging.

Keywords: debugging; grammar; error diagnosis

1 Introduction

A program may fail when processing certain inputs, and it is not often clear why. Not all parts of the input may contribute equally to the failure, and the developers need to understand the exact circumstances of program failure. We introduce two techniques to help the developers.

Our DDSET algorithm reduces a given failure inducing input to its abstract form, precisely identifying the failure causing parts of the input. Our ALHAZEN technique, on the other hand, analyses the input interpretations and automatically builds and refines *hypotheses* about why the program failed.

2 DDSET

Given a failure inducing input, and the program grammar, DDSET starts by minimizing the failure inducing input to its minimal form using hierarchical delta debugging. DDSET starts

[1] CISPA Helmholtz Center for Information Security, Saarbrücken rahul.gopinath@cispa.de
[2] CISPA Helmholtz Center for Information Security, Saarbrücken nikolas.havrikov@cispa.de
[3] CISPA Helmholtz Center for Information Security, Saarbrücken alexander.kampmann@cispa.de
[4] CISPA Helmholtz Center for Information Security, Saarbrücken ezekiel.soremekun@cispa.de
[5] CISPA Helmholtz Center for Information Security, Saarbrücken zeller@cispa.de

doi:10.18420/SE2021_12

from the root node of the input parse tree, and checks whether any node could be replaced by a randomly generated node of the same kind while reproducing the observed failure. If a node that can be thus replaced is marked abstract. The *abstract failure inducing inputs* are produced by collapsing the annotated parse tree to the corresponding string where abstract nodes are left as non-terminals. Such abstractions can be used to generate further failure inducing inputs by replacing non-terminal symbols in the string with random expansions of the same non-terminal.

In our evaluation the inputs generated from abstractions induced failures 99.9% of the time.

3 Alhazen

Imagine you have a calculator program that fails on the input "sqrt(-900)", and passes with "tan(9)". ALHAZEN uses a context-free grammar to decompose the inputs into individual parts, in the example "function == sqrt", "function == tan", "number == -900" and "number == 9". Those are used as *features* for a decision-tree learner. Such a tree learner generates a model which can decide between failing and non-failing inputs. In the example, the tree learner provides the hypothesis that the program fails if "number <= -445.5". This is not correct, but explains all available observations. ALHAZEN proceeds to generate inputs which challenge this hypothesis. We execute those tests and use the new observations to *refine* the model. After several iterations of this *feedback loop*, ALHAZEN comes up with the correct hypothesis: The program fails if "number <= 0".

Our evaluation shows, thatALHAZEN's final hypothesis classified 92% of all inputs correctly.

4 Conclusion

DDSET and ALHAZEN are techniques for explaining why a program fails, and their explanations are complementary. DDSET focuses on identifying syntactical properties directly related to structural elements. ALHAZEN's hypotheses identify concrete values or value ranges, arguing about semantic, rather than syntactic, input properties. Evaluation results show that both approaches can be used to generate more failure-inducing inputs, predict whether an input triggers a failure or provide insights into why a failure occurs.

Bibliography

[Go20] Gopinath, Rahul; Kampmann, Alexander; Havrikov, Nikolas; Soremekun, Ezekiel; Zeller, Andreas: Abstracting Failure-Inducing Inputs. In: ISSTA 2020. July 2020.

[Ka20] Kampmann, Alexander; Havrikov, Nikolas; Soremekun, Ezekiel; Zeller, Andreas: When does my Program do this? Learning Circumstances of Software Behavior. In: ESEC/FSE 2020. November 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 49

Mining Input Grammars

Rahul Gopinath[1] Björn Mathis[2] Andreas Zeller[3]

Abstract: To assess the behavior of a program, one needs to understand its *inputs*—their sources, their structure, and how they lead to individual behavior. But as syntax and semantics of inputs are almost never completely specified, humans and computers constantly have to figure out how to produce a particular behavior.

In this abstract, we show how to automatically extract accurate, well-structured *input grammars* from existing programs. Such input grammars are useful for *software testing,* as they can serve as *producers of valid, high-quality inputs for software testing* that easily pass through parsing and validation to reliably trigger the desired program behavior. Moreover, they allow testers to *control* which inputs are to be produced—in contrast to the majority of fuzzers, that operate as black boxes.

Our Mimid prototype [GMZ20] uses dynamic tainting to *extract input grammars* from given programs—grammars that are well-structured and highly readable, even for complex recursive input formats such as JavaScript or JSON. Specific *parser-directed test generators* [Ma19; MGZ20] systematically explore the input syntax, such that grammars can be mined even without any given inputs.

Keywords: grammar; grammar mining; automated testing; fuzzing; input generation

1 Introduction

Understanding the input specification is key to understanding the program behavior, but few come with an input specification. Even if an input specification is given, it may be obsolete, incomplete, or incorrect. Given that inaccuracy in the input specification is a blind spot, it is important to obtain accurate input specifications. We describe three key techniques for recovering accurate input specifications from a given program. Our techniques work on all styles of handwritten recursive descent programs. We start by generating unbiased input samples [Ma19] that explore the input space of even complex parsers with tokenisation [MGZ20]. The execution traces of these input samples are then used to decompose the given input into parse trees, which are then combined and abstracted into a general grammar [GMZ20]. The grammars obtained are well readable and accurate.

[1] CISPA Helmholtz Center for Information Security, Saarbrücken rahul.gopinath@cispa.de
[2] CISPA Helmholtz Center for Information Security, Saarbrücken bjoern.mathis@cispa.de
[3] CISPA Helmholtz Center for Information Security, Saarbrücken zeller@cispa.de

doi:10.18420/SE2021_13

2 Grammar Mining

Mimid relies on input decomposition to construct the grammar, and uses three key insights. The first insight is that the *dynamic program dependence tree* of the input processing is structurally equivalent to the parse tree. Next, Mimid limits itself to the context-free decomposition of the input. Hence, it only tracks the input decomposition in the first level parser. Finally, any input character is consumed by the node that accesses it last. Each character is attached to the node in the dynamic control dependence tree that consumed it. The parse trees resulting from the application of these techniques are then collapsed. Each node of a parse tree corresponds to a particular expansion in the corresponding grammar where the node name corresponds to the non-terminal symbol of the expansion, and the names of the child nodes correspond to the non-terminal symbols in the expansion. Any attached characters become the terminal symbols in the expansion. All such expansions are collected, and repetition of node sequences is abstracted out to produce the final grammar.

In our evaluation, the mined grammars had an accuracy of approximately 90% as producers and as recognizers.

3 Parser Directed Input Generation

For Mimid to be effective, it needs input samples that cover every input feature. One can't rely on existing input corpus as such a corpus may not exist, and even if one exists, it may not cover all features. *Parser directed test generation* provides a solution. Our technique [Ma19] injects instrumentation to differentiate between *incorrect* and *invalid* inputs. We also track the comparisons made to the last index of the input. We start with an empty input, and extend the input with a random character if the input was found to be *incomplete*. If on the other hand, the input was found to be *incorrect*, the last character added is removed, and is replaced by one of the characters it was compared against. Proceeding in this fashion, complete valid inputs are achieved. While tokenisation of inputs can reduce the effectiveness, we show [GMZ20] that tokenisation specific instrumentation can overcome this issue.

References

[GMZ20] Gopinath, R.; Mathis, B.; Zeller, A.: Mining Input Grammars from Dynamic Control Flow. In: ESEC/FSE. artifact: https://github.com/vrtrha/mimid, Nov. 2020, URL: https://publications.cispa.saarland/3101/.

[Ma19] Mathis, B.; Gopinath, R.; Mera, M.; Kampmann, A.; Höschele, M.; Zeller, A.: Parser-Directed Fuzzing. In: PLDI 2019. June 2019, URL: https://publications.cispa.saarland/2823/.

[MGZ20] Mathis, B.; Gopinath, R.; Zeller, A.: Learning Input Tokens for Effective Fuzzing. In: ISSTA 2020. Pp. 1–11, July 2020, URL: https://publications.cispa.saarland/3135/.

An Exploratory Study on Performance Engineering in Model Transformations

Raffaela Groner[1], Luis Beaucamp[1], Matthias Tichy[1], Steffen Becker[2]

Abstract: Model–Driven Software Engineering is used to deal with the increasing complexity of software, but this trend also leads to larger and more complex models and model transformations. While improving the performance of transformation engines has been a focus, there does not exist any empirical study on how transformation developers deal with performance issues. We used a quantitative questionnaire to investigate whether the performance of transformations is actually important for transformation developers. Based on the answers to the questionnaire, we conducted qualitative semi-structured interviews. The results of the online survey show that 43 of 81 participants have already tried to improve the performance of a transformation and 34 participants are sometimes or only rarely satisfied with the execution performance. Based on the answers from our 13 interviews, we identified different strategies to prevent or find performance issues in model transformations as well as different types of causes of performance issues and solutions to resolve them. We compiled a collection of tool features perceived helpful by the interviewees for finding causes. Overall, our results show that performance of transformations is relevant and that there is a lack of support for transformation developers without detailed knowledge of the engine to solve performance issues. This summary refers to our work, which was accepted for the Foundation Track of the ACM / IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS) in 2020 [Gr20].

Keywords: Mixed Method Study; Model Transformation; ATL; Henshin; QVTo; Viatra; Performance Engineering

Due to the increasing size and complexity of software systems to be developed, the size and complexity of models used in Model–Driven Software Engineering (MDSE) is also increasing. Since these models serve as input for model transformations, this trend can lead to an increasing execution time of transformations until a performance issue occurs. Currently, there is a lot of research that deals with the performance of the transformation engine, for example by improving the engine [Fr17] or developing approaches that speed up the transformation execution by parallel execution [BWV16]. However, there is still a lack of research focusing on whether transformation developers have to deal with performance issues and, if they do, how they try to solve them. Therefore, we conducted an explorative mixed method study, consisting of a quantitative online survey and qualitative interviews.

The results show that 43 out of 81 survey respondents have already tried to analyze or improve the performance of their transformations. The issues mentioned are long execution

[1] Ulm University, Institute for Software Engineering and Programming Languages, James–Franck–Ring, D–89069 Ulm, Germany raffaela.groner@uni-ulm.de, luis.beaucamp@uni-ulm.de, matthias.tichy@uni-ulm.de
[2] University of Stuttgart, Institute for Software Engineering, Universitätsstraße, D–70569 Stuttgart, Germany steffen.becker@iste.uni-stuttgart.de

doi:10.18420/SE2021_14

times and high memory consumption. The transformation developers have developed techniques to prevent performance issues or to find causes. For example, they carry out performance tests to prevent issues. The strategies used to find causes vary depending on the knowledge of the engine and the available information. Since most of the languages do not provide any support for such an analysis at the transformation level, pure users can only try to find possible causes through trial–and–error. Developers with more knowledge of the engine, use tools, e.g. a Java profiler, at the engine level to analyze the performance. The developers have discovered different causes for performance issues that occur either at the engine level, the transformation level, or in the input model. Causes on the engine level include, e.g., properties of the engine, for instance in some engines, lists are copied multiple times to avoid side effects. In case of transformations, e.g., expensive OCL expressions can be the cause of performance issues. On the model level, certain structures, such as deep inheritance hierarchies, can be the cause. With regard to the solutions used, there are only a few that require adjustment of the engine. Instead, the developers try to improve the execution time, e.g., by defining their transformations more unambiguous through stricter preconditions or by using language concepts like caching. It also becomes clear that the developers want more information about a transformation execution at the transformation level. For example, they want not only information like execution time or memory consumption, but also tracing information about how a transformation is executed. They also want more tool support, e.g., profilers at the transformation level or analyses to detect expensive operations.

Some transformation developers have already developed techniques for analyzing and improving the performance, but these work mainly at the engine level. Therefore, these techniques are not applicable for pure users, although many solutions could be applied by them, since they do not require any engine customization. In order to address this problem, we want to develop an approach in our future work that will also help pure users to identify causes for performance issues and to fix them.

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) - Ti 803/4-1.

Bibliography

[BWV16] Burgueño, Loli; Wimmer, Manuel; Vallecillo, Antonio: A Linda-based platform for the parallel execution of out-place model transformations. Information and Software Technology, 79:17 – 35, 2016.

[Fr17] Fritsche, Lars; Leblebici, Erhan; Anjorin, Anthony; Schürr, Andy: A Look-Ahead Strategy for Rule-Based Model Transformations. In: MODELS (Satellite Events). volume 2019 of CEUR Workshop Proceedings. CEUR-WS.org, pp. 45–53, 2017.

[Gr20] Groner, Raffaela; Beaucamp, Luis; Tichy, Matthias; Becker, Steffen: An Exploratory Study on Performance Engineering in Model Transformations. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. MODELS '20, Association for Computing Machinery, New York, NY, USA, p. 308–319, 2020.

On the Cost and Profit of Software Defect Prediction

Steffen Herbold[1]

Abstract: We summarize the article *On the cost and profit of software defect prediction* [He19], which was published in the IEEE Transactions on Software Engineering in 2019.

Keywords: Defect Prediction; Costs; Return On Investment

1 Overview

The article "On the cost and profit of software defect prediction" published in the IEEE Transactions on Software Engineering in 2019 propose a cost model to enable the estimation of the expected profit when using machine learning models for defect prediction [He19]. Defect prediction can be a powerful tool to guide the use of quality assurance resources. However, while lots of research covered methods for defect prediction as well as methodological aspects of defect prediction research, the actual cost saving potential of defect prediction is still unclear. We close this research gap and formulate a cost model for software defect prediction. We derive mathematically provable boundary conditions that must be fulfilled by defect prediction models such that there is a positive profit when the defect prediction model is used. Our cost model includes aspects like the costs for quality assurance, the costs of post-release defects, the possibility that quality assurance fails to reveal predicted defects.

2 Results

Our results show that the unrealistic assumption that defects only affect a single software artifact leads to inaccurate cost estimations. Moreover, the results indicate that thresholds for machine learning metrics like precision and recall are also not suited to define success criteria for software defect prediction.

Bibliography

[He19] Herbold, Steffen: On the costs and profit of software defect prediction. IEEE Transactions on Software Engineering, (01):1–1, dec 2019.

[1] Karlsruher Institut für Technologie, Institute AIFB, Kaiserstr. 89, 76133 Karlsruhe, Deutschland steffen.herbold@kit.edu

doi:10.18420/SE2021_15

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 55

On the Feasibility of Automated Prediction of Bug and Non-Bug Issues

Steffen Herbold[1], Alexander Trautsch[2], Fabian Trautsch[3]

Abstract: We summarize the article *On the feasibility of automated prediction of bug and non-bug issues* [HTT20], which was published in the Empirical Software Engineering in 2020.

Keywords: issue type prediction; bug issues; recommendation systems; empirical software engineering

1 Overview

The article "On the feasibility of automated prediction of bug and non-bug issues" published in Empirical Software Engineering in 2020 considers the application of machine learning for the automated classification of issue types, e.g., for research purposes or as a recommendation system [HTT20]. Issue tracking systems are used to track and describe tasks in the development process, e.g., requested feature improvements or reported bugs. However, past research has shown that the reported issue types often do not match the description of the issue. Within our work, we evaluate the state of the art of issue type prediction system can accurately identify bugs. We also investigate if manually specified knowledge can improve such systems.

2 Results

While we found that manually specified knowledge about contents is not useful, respecting structural aspects can be valuable. Our experiments show that issue type prediction system can be trained based on large amounts of unvalidated data and still be sufficiently accurate to be useful. Overall, the misclassifications of the automated system are comparable to the misclassifications made by developers.

[1] Karlsruher Institut für Technologie, Institute AIFB, Kaiserstr. 89, 76133 Karlsruhe, Deutschland steffen.herbold@kit.edu
[2] Georg-August-Universität Göttingen, Institute für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland alexander.trautsch@cs.uni-goettingen.de
[3] Georg-August-Universität Göttingen, Institute für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland fabian.trautsch@cs.uni-goettingen.de

doi:10.18420/SE2021_16

Literatur

[HTT20] Herbold, S.; Trautsch, A.; Trautsch, F.: On the feasibility of automated prediction of bug and non-bug issues. Empirical Software Engineering 25/6, S. 5333–5369, Sep. 2020, URL: https://doi.org/10.1007/s10664-020-09885-w.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 57

A Systematic Mapping Study Of Developer Social Network Research

Steffen Herbold[1], Aynur Amirfallah[2], Fabian Trautsch[3], Jens Grabowski[4]

Abstract: We summarize the article *A systematic mapping study of developer social network research* [He21], which was published in the Journal of Systems and Software in 2020.

Keywords: Developer social networks; Mapping study; Literature survey

1 Overview

The article "A systematic mapping study of developer social network research" published in the Journal of Systems and Software in 2020 presents the results of a systematic mapping study of the state of the art of developer social network research [He21]. Developer social networks (DSNs) are a tool for the analysis of community structures and collaborations between developers in software projects and software ecosystems. We identified 255 primary studies on DSNs. We mapped the primary studies to research directions, collected information about the data sources and the size of the studies, and conducted a bibliometric assessment.

2 Results

We found that nearly half of the research investigates the structure of developer communities. Other frequent topics are prediction systems build using DSNs, collaboration behavior between developers, and the roles of developers. Moreover, we determined that many publications use a small sample size regarding the number of projects, which could be problematic for the external validity of the research. Our study uncovered several open issues in the state of the art, e.g., studying inter-company collaborations, using multiple information sources for DSN research, as well as general lack of reporting guidelines or replication studies.

[1] Karlsruher Institut für Technologie, Institute AIFB, Kaiserstr. 89, 76133 Karlsruhe, Deutschland steffen.herbold@kit.edu
[2] Georg-August-Universität Göttingen, Institut für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland aynur.amirfallah@stud.uni-goettingen.de
[3] Georg-August-Universität Göttingen, Institut für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland fabian.trautsch@cs.uni-goettingen.de
[4] Georg-August-Universität Göttingen, Institut für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland jens.grabowski@cs.uni-goettingen.de

doi:10.18420/SE2021_17

Literatur

[He21] Herbold, S.; Amirfallah, A.; Trautsch, F.; Grabowski, J.: A systematic mapping study of developer social network research. Journal of Systems and Software 171/, S. 110802, 2021.

Cutting through the Jungle: Disambiguating Model-based Traceability Terminology (Extended Abstract)

Jörg Holtmann[1] Jan-Philipp Steghöfer[2] Michael Rath[3] David Schmelter[4]

Abstract: This extended abstract summarizes our distinguished paper [Ho20], published and presented in 2020 at the 28^{th} IEEE International Requirements Engineering Conference (RE'20).

Keywords: Traceability; Requirements Engineering; Model-based Engineering; Terminology

Natural language remains the dominant documentation format for requirements specifications for software-intensive systems. At the same time, models are increasingly applied in the engineering of such systems: Embedded systems development heavily relies on model-based systems engineering for interdisciplinary communication and model-driven software development for early automated analyses and code generation. In combination, the development processes for software-intensive systems include intertwined development phases producing, among others, informal system requirements, semi-formal system design models, informal software requirements, formal software design models, code, and tests.

At the same time, traceability is demanded by many development and safety standards for software-intensive systems and has to be established throughout the development lifecycle. The term traceability and its definition originate in the Requirements Engineering (RE) research community, which also provides a terminology for this area. However, existing definitions even only within the RE research community partially contradict each other.

Looking moreover into the modeling research community, its literature on model-based traceability uses the same terms with different meanings. Furthermore, additional aspects that go beyond the traceability terminology shaped by the RE community are relevant when working with models. For example, when models are transformed during development to yield other models with more fine-grained granularity and additional details, trace links between the source models and the target models can be established automatically. In addition, trace links are often stored in external trace models or are included in models in languages such as the Unified Modeling Language or the Systems Modeling Language. The traceability definitions from the RE community do not yet cover such cases. Thus, there is the need to extend these definitions to cover cases from model-based engineering.

[1] Software Engineering & IT Security, Fraunhofer IEM, Paderborn, Germany, joerg.holtmann@iem.fraunhofer.de
[2] Chalmers | University of Gothenburg, Gothenburg, Sweden jan-philipp.steghofer@gu.se
[3] Technische Universität Ilmenau, Ilmenau, Germany michael.rath@tu-ilmenau.de
[4] Software Engineering & IT Security, Fraunhofer IEM, Paderborn, Gemany, david.schmelter@iem.fraunhofer.de

doi:10.18420/SE2021_18

Whereas existing secondary studies list the terminology from both communities, they do not resolve conflicts or add missing aspects. They also circumscribe concepts rather than naming them concretely while reusing existing terminology with additional identifiers. However, a domain terminology is the common ground for a set of people communicating with each other. Thus, a scientific community has to define its own terminology as the basis for an efficient communication and conduct of research. Consequently, with the inclusion of the modeling research community it becomes necessary to extend the RE terminology, include new terms, dissolve term conflicts, and dismiss terms that cannot be defined unambiguously.

In order to disambiguate the terminology on traceability of both the RE and the modeling research communities, we conducted a multi-stage literature review. Figure 1 sketches our research method, which consists of several iterative refinements of the yielded terminology and an accompanying taxonomy. Our results are based on and validated with a tertiary literature review and samples from primary literature. We include a mapping to how the secondary and primary studies in the review use the concepts in our terminology.

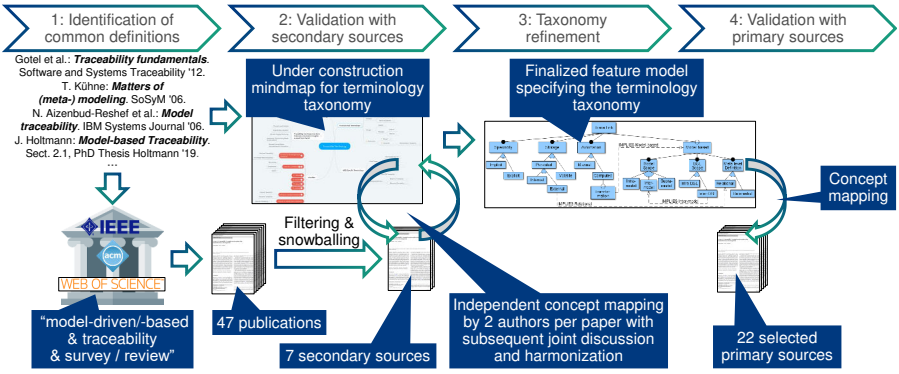


Fig. 1: Overview of our research method—iterative refinement of the terminology and its taxonomy

Thus, the contribution of our paper is a terminology that disambiguates the different terms used in model-based traceability. Furthermore, we provide a taxonomy specified by means of a feature model, which conceptually visualizes the structure of our terminology and formalizes the relationships between certain terms. As a side effect, this terminology serves as a classification scheme for how trace links are used in model-based environments. Since primary studies rarely define the properties of the traceability approach they propose, our work helps comparing and contrasting different approaches. Summarizing, we believe our work will simplify discussions between requirements engineers and engineers working with models since they can now use a set of unambiguous terms to discuss traceability concepts.

Bibliography

[Ho20] Holtmann, Jörg; Steghöfer, Jan-Philipp; Rath, Michael; Schmelter, David: Cutting through the Jungle: Disambiguating Model-based Traceability Terminology. In: 28^{th} IEEE International Requirements Engineering Conference (RE'20). IEEE, pp. 8–19, 2020. Distinguished Paper.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 61

Detecting Quality Problems in Research Data: A Model-Driven Approach[1]

Arno Kesper[2], Viola Wenz[2], Gabriele Taentzer[2]

Abstract: The quality of research data is essential for scientific progress. A major challenge in data quality assurance is the localisation of quality problems that are inherent to data. Based on the observation of a dynamic shift in the database technologies employed, we present a model-driven approach to analyse the quality of research data. It allows a data engineer to formulate anti-patterns that are generic concerning the database format and technology. A domain expert chooses a pattern that has been adapted to a specific database technology and concretises it for a domain-specific database format. The resulting concrete pattern is used by a data analyst to locate quality problems in the database. As a proof of concept, we implemented tool support that realises this approach for XML databases. We evaluated our approach concerning expressiveness and performance.

The original paper has been published at the International Conference on Model Driven Engineering Languages and Systems 2020 [KWT20].

Keywords: Data quality; Model-driven development; Pattern matching

1 Introduction

As scientific progress highly depends on the quality of research data, there are strict requirements for data quality coming from the scientific community. Relevant quality dimensions include consistency, completeness and precision. In a qualitative study on cultural heritage data we observed a variety of data quality problems. Examples include imprecise, redundant and semantically incorrect data. A major step to data quality assurance is to analyse the inherent quality problems. Due to the dynamic digitalisation in specific scientific fields, different database technologies and data formats may be used. In the digital humanities, for example, a shift from relational to XML and further to graph databases can be observed. To cope with this challenge, we present a model-driven approach to data quality analysis. Given a large variety of data quality problems, which may have various concrete forms, a *model-driven approach* is promising to develop technology- and format-independent concepts and tooling for data quality analysis.

[1] This work was partially funded by the German Federal Ministry of Education and Research (BMBF) grant 16QK06A.

[2] Philipps-Universität Marburg, Marburg, Germany, {arno.kesper, viola.wenz, taentzer}@uni-marburg.de

doi:10.18420/SE2021_19

2 Approach

Our model-driven approach to data quality analysis is based on the observation that many quality problems show anti-patterns. In contrast to related approaches, it allows data engineers to specify parameterised anti-patterns for data quality problems that are *generic* concerning the underlying database technology and format. Such a generic pattern can be adapted to several database technologies, resulting in *abstract* patterns. Depending on the database technology this can be done (semi-)automatically by a data engineer. A domain expert chooses an abstract pattern as template and concretises it to a domain-specific database format and to a concrete quality problem. The resulting *concrete* pattern is then automatically translated into a corresponding query language. A data analyst can apply the concrete pattern to localise occurrences of the quality problem in a database. The data analyst can then initiate an improvement process.

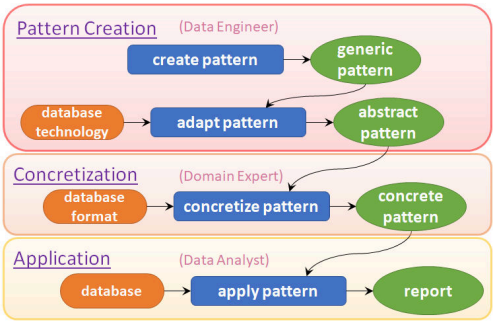


Fig. 1: Workflow of pattern creation and application

The core of the approach is a metamodel for patterns. It currently supports generic, XML-adapted abstract and concrete patterns. The metamodel is designed to allow extensions for further database technologies. Patterns are defined as first-order logic conditions over graphs. We use directed graphs to interpret data independently of the database technology. Our implementation includes a mapping between graphs and XML data as well as a translation of XML-adapted concrete patterns to XQuery. Mappings for other database technologies are outlined in our paper.

The evaluation of our approach based on cultural heritage data revealed that its strength lies in detecting structural problems. The expressiveness can be expanded by integrating further techniques for quality analysis, such as similarity metrics.

Our overall goal is to develop a framework for quality assurance of research data, where the detection of quality problems is the first essential step. In general, there is a need for powerful tools that analyse the quality of data. Data quality assurance is also of particular importance in the context of data-intensive software systems, which have gained interest in recent years. Hence, this topic affects not only data engineering but also software engineering.

Bibliography

[KWT20] Kesper, Arno; Wenz, Viola; Taentzer, Gabriele: Detecting Quality Problems in Research Data: A Model-Driven Approach. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. MODELS '20, Association for Computing Machinery, New York, NY, USA, p. 354–364, 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 63

Isolating Faults in Failure-Inducing Inputs

Lukas Kirschner[1] Ezekiel Soremekun[2,3] Andreas Zeller[4]

Abstract: Program failures are often caused by faulty inputs (e.g. due to data corruption). When an *input induces failure*, one needs to *debug the input data*, i.e. isolate faults to obtain valid input data. Typically, debuggers focus on diagnosing faults in the program, rather than the input. This talk instead presents an approach that automatically repairs faults in the input data, without requiring program analysis. In addition, we present empirical data on the causes and prevalence of invalid inputs in practice, we found that four percent of inputs in the wild are invalid.

We present a general-purpose algorithm called *ddmax* that automatically isolates faults in invalid inputs and recovers the maximal valid input data. The aim of *ddmax* is to (1) identify which parts of the input data prevent processing by the program, and (2) recover as much of the (valuable) input data as possible.

Given a program and an invalid input, through experiments, *ddmax* recovers and repairs as much data as possible. The difference between the original failing input and the "maximized" passing input includes all input fragments that could not be processed, i.e. *the fault*. This approach is useful for automatically debugging and repairing invalid inputs.

Keywords: Program Inputs; Debugging; Input Repair; Fault Localisation

1 Summary

This article is an abridged version of our reseach paper titled "Debugging Inputs" which is published in the proceedings of the fourty-second IEEE/ACM International Conference on Software Engineering (ICSE) [KSZ20].

When a program fails, the failure can be caused by a faulty input (e.g. corrupt data): Which portion of the input data is faulty? Which part of the input contains valid data? Identifying faults in program inputs is challenging because input files often have complex structures and are required to conform to specific syntactic and semantic rules.

General-purpose automated debugging techniques specialize in diagnosing faults in program code [Bö17; Wo16], rather than the input data. Notably, input reduction techniques simplify failure-inducing inputs to aid developers during debugging. For instance, delta debugging (*ddmin*) [ZH02] simplifies inputs to a smaller subset that reproduces a failure. However,

[1] CISPA Helmholtz Center for Information Security, Saarbrücken s8lukirs@stud.uni-saarland.de
[2] SnT, University of Luxembourg, Luxembourg ezekiel.soremekun@uni.lu
[3] This work was done while working at CISPA Helmholtz Center for Information Security, Saarbrücken
[4] CISPA Helmholtz Center for Information Security, Saarbrücken zeller@cispa.de

doi:10.18420/SE2021_20

ddmin is not a good fit for input debugging: It produces the smallest subset of the input that also produces an input error – typically a single character – which is neither helpful for diagnosis nor data recovery. In contrast to input reduction, input debugging requires isolating faults in the input data, i.e., (1) identifying which parts of the input data prevent processing, and (2) recovering as much of the (valuable) input data as possible.

We present a general-purpose algorithm called *ddmax* that automatically isolate faults in inputs [KSZ20]. We obtain *ddmax* by inverting the original delta debugging (*ddmin*) algorithm [ZH02], such that it maximizes the subset of the input that can still be processed by the program through test experiments. It repeatedly adds data from the originally failing input (first larger pieces, then smaller pieces) as long as the failure does not occur. Eventually, it isolates faults by providing all input fragments that could not be processed, i.e., the difference between the failing input and the "maximized" passing input. In this work, we present two variants of *ddmax*: (1) *lexical ddmax* which repairs arbitrary invalid inputs at the character level, and (2) *syntactic ddmax* which performs input repair on the parse tree.

In our evaluation of thousands of input files, we found that four percent of inputs in the wild are invalid, they could not be processed either by their grammar or program(s). In our evaluation of the effectiveness of *ddmax*, *ddmax* repaired about 69% of input files and recovered about 78% of valid data, within one minute per input. A comparison of the fault diagnosis of *ddmax* to that of *ddmin* showed that, on average, only 12% of a *ddmin* diagnosis contains the failure cause, while a third of the diagnosis contains noise, i.e., valid data.

We have presented *ddmax*– the first generic technique for automatically repairing failure-inducing inputs, it recovers a maximal subset of the input that can still be processed by the program at hand. Our work opens the door for a number of exciting research opportunities in the following areas: synthesizing input structures, semantic input repair and fuzzing.

References

[Bö17] Böhme, M.; Soremekun, E. O.; Chattopadhyay, S.; Ugherughe, E.; Zeller, A.: Where is the bug and how is it fixed? An experiment with practitioners. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Pp. 117–128, 2017.

[KSZ20] Kirschner, L.; Soremekun, E.; Zeller, A.: Debugging Inputs. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). IEEE, 2020, URL: https://publications.cispa.saarland/3060/.

[Wo16] Wong, W. E.; Gao, R.; Li, Y.; Abreu, R.; Wotawa, F.: A survey on software fault localization. IEEE Transactions on Software Engineering 42/8, pp. 707–740, 2016.

[ZH02] Zeller, A.; Hildebrandt, R.: Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering 28/2, pp. 183–200, 2002.

Determining Context Factors for Hybrid Development Methods with Trained Models

Jil Klünder[1], Dzejlana Karajic[2], Paolo Tell[3], Oliver Karras[4], Christian Münkel[5], Jürgen Münch[6], Stephen G. MacDonell[7], Regina Hebig[8], Marco Kuhrmann[9]

Abstract: Selecting a suitable development method for a specific project context is one of the most challenging activities in process design. To extend the so far statistical construction of hybrid development methods, we analyze 829 data points to investigate which context factors influence the choice of methods or practices. Using exploratory factor analysis, we derive five base clusters consisting of up to 10 methods. Logistic regression analysis then reveals which context factors have an influence on the integration of methods from these clusters in the development process. Our results indicate that only a few context factors including project/product size and target application domain significantly influence the choice.

This summary refers to the paper *"Determining Context Factors for Hybrid Development Methods with Trained Models"*. This paper was published in the proceedings of the International Conference on Software and System Process in 2020.

Keywords: Software Process; Hybrid Development Method; Trained Models

1 Introduction

Nowadays, most software development processes do not consist of a single method and/or practice. Research has shown that the combination of different methods and practices into a so-called hybrid method is state-of-the-practice. Research has shown, how a "typical" hybrid method looks like, and how a hybrid method can be statistically constructed [Te20].

Problem Statement & Objective So far, hybrid methods have been devised based on experience and over time [Kl19]. However, it remains unclear which factors need to be considered when devising such a hybrid method. Therefore, in this paper, we analyze which (context) factors can help to derive a suitable hybrid method.

[1] Leibniz Universität Hannover, Germany, jil.kluender@inf.uni-hannover.de
[2] University of Passau, Germany, dzejlana.karajic@gmail.com
[3] IT University Copenhagen, Denmark, pate@itu.dk
[4] Leibniz Universität Hannover, Germany, oliver.karras@inf.uni-hannover.de
[5] Leibniz Universität Hannover, Germany, christian@muenkel.cc
[6] Reutlingen University, Germany, j.muench@computer.org
[7] Auckland University of Technology, New Zealand, stephen.macdonell@aut.ac.nz
[8] Chalmers | University of Gothenburg, Sweden, regina.hebig@cse.gu.se
[9] University of Passau, Germany, kuhrmann@acm.org

doi:10.18420/SE2021_21

Contribution We analyze a subset of the HELENA dataset consisting of 829 data points using exploratory factor analysis and logistic regression analysis. We cluster the set of methods and investigate which context factors influence the likelihood of using methods from a specific cluster.
Our results reveal that only a few context factors including the company size and some target application domains influence the likelihood of using methods from one of the clusters.

2 Results

The exploratory factor analysis on 829 data points from the HELENA dataset revealed five clusters consisting of up to ten methods. These clusters can be described as *mainly agile*, *mainly traditional* or *a mixture of agile and traditional methods*. The logistic regression analysis then reveals for each of the clusters, which context factors influence the likelihood of using methods from the respective cluster. Our results indicate that (1) companies working distributed across one continent tend to use agile methods, (2) the target application domain has an influence in some cases (defense systems, space systems, telecommunication, web applications), and (3) small projects tend to use traditional methods. In all other cases, our analysis did not reveal significant influences. In the last step, we extend the method clusters with frequently used practices following a construction process similar to Tell et al. [Te20].

3 Conclusion & Future Work

Our paper [Kl20] documents context factors that influence the likelihood of using methods and practices from a specific set. As, according to our results, context factors seem to be not as relevant as they are supposed to be, future work should investigate which other factors decide on the shape hybrid methods.

Bibliography

[Kl19] Klünder, Jil; Hebig, Regina; Tell, Paolo; Kuhrmann, Marco; Nakatumba-Nabende, Joyce; Heldal, Rogardt; Krusche, Stephan; Fazal-Baqaie, Masud; Felderer, Michael; Bocco, Marcela Fabiana Genero et al.: Catching up with method and process practice: An industry-informed baseline for researchers. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, pp. 255–264, 2019.

[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. In: Proceedings of the International Conference on Software and System Processes. pp. 61–70, 2020.

[Te20] Tell, Paolo; Klünder, Jil; Küpper, Steffen; Raffo, David; MacDonell, Stephen; Münch, Jürgen; Pfahl, Dietmar; Linssen, Oliver; Kuhrmann, Marco: Towards the statistical construction of hybrid development methods. Journal of Software: Evolution and Process, p. e2315, 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 67

Skill-Based Verification of Cyber-Physical Systems

Alexander Knüppel[1] Inga Jatzkowski[2] Marcus Nolte[3] Tobias Runge Knüppel[4] Thomas Thüm[5] Ina Schaefer[6]

Abstract: This work has been accepted at the 23rd International Conference on Fundamental Approaches to Software Engineering, held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020 [Kn20].

The increase of complexity in modeling cyber-physical systems poses a challenge for formally ensuring their functional correctness. Lack of expert knowledge and scalability are two limiting factors that prohibit a seamless integration into today's software engineering processes. To address this challenge, we propose to adopt and formalize the notion of *skill graphs*, an abstract and easy-to-use modeling notion for representing automated vehicle driving maneuvers. For formally verifying that skill graphs are well-formed and comply with a given set of safety requirements, we incorporate *hybrid programs* into our formalization. Hybrid programs constitute a program notion for cyber-physical systems on the basis of differential dynamic logic, which enables deductive and compositional verification following the idea of Hoare-style reasoning. That is, simpler verified skill graphs can be combined to exhibit complex maneuvers while validity is retained (i.e., without the need of re-verification). To showcase the benefits of our theoretical considerations, we implemented our framework in an open-source tool named Skeditor and conducted a case study exhibiting an automatic *vehicle follow mode*.

Keywords: Deductive verification; design by contract; formal methods; theorem proving; hybrid program; cyber-physical systems

Nowadays, cyber-physical systems are ubiquitously present in our lives and have a direct impact on most humans. As the complexity of modern cyber-physical systems increases while also being applied for safety-critical tasks, formal verification methods are required. This poses a major challenge on developers and engineering processes in the early development stages to ensure functional correctness. Hence, an important challenge is to derive modeling and verification techniques that (1) are easy to integrate into the software development cycle by reducing modeling complexity and (2) allow to identify severe requirement and modeling mistakes at design time. To address this challenge, we propose a model-based verification framework called Skeditor and an accompanying IDE that allows to prototype driving maneuvers (e.g., *following a leading a vehicle*) and verify their adherence to a set of formal requirements.

[1] TU Braunschweig a.knueppel@tu-bs.de
[2] TU Braunschweig jatzkowski@ifr.ing.tu-bs.de
[3] TU Braunschweig nolte@ifr.ing.tu-bs.de
[4] TU Braunschweig tobias.runge@tu-bs.de
[5] University of Ulm thomas.thuem@uni-ulm.de
[6] TU Braunschweig i.schaefer@tu-bs.de

doi:10.18420/SE2021_22

To achieve this goal, SKEDITOR combines two concepts, namely *skill graphs* and *hybrid programms*. Skill graphs were introduced by Reschka et al. [Re15] and serve as modeling notation, following the principle of *separation of concerns* by enabling the decomposition of more complex maneuvers into modular, reusable, and inter-related building blocks (i.e., skills). To model the behavior of skills that interact with the physical environment (i.e., controllers), we rely on *hybrid programs* [Pl18], which comprise a program notation for cyber-physical systems together with a deductive calculus to prove controller correctness. One of the most important properties of our formalization is *compositionality*, for which we defined a composition operator for skill graphs. That is, smaller provably correct skill graphs can be combined while correctness is retained without the need of re-verification.

In our evaluation, we modeled and verified a skill graph representing a vehicle follow mode combining two simpler maneuvers, namely *following a leading vehicle* and *following hard shoulder* (i.e. the lateral and longitudinal control tasks). We modeled each of these maneuvers individually and measured the verification effort with KEYMAERA X [Fu15]. Due to the typical overlap in skills for both maneuvers, our results show that the composition reduces the total verification effort by 53% compared to monolithic modeling.

In summary, we provide the first formalization of skill graphs including tool support and show how they can be combined with hybrid programming as a formal underpinning. As demonstrated in our work, SKEDITOR allows developers to prototype driving maneuvers and verify their safety as part of the early software development life cycle, while – due to compositionality – costly re-verification can be minimized.

Bibliography

[Fu15] Fulton, Nathan; Mitsch, Stefan; Quesel, Jan-David; Völp, Marcus; Platzer, André: KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In: International Conference on Automated Deduction. Springer, pp. 527–538, 2015.

[Kn20] Knüppel, Alexander; Jatzkowski, Inga; Nolte, Marcus; Thüm, Thomas; Runge, Tobias; Schaefer, Ina: Skill-Based Verification of Cyber-Physical Systems. In: International Conference on Fundamental Approaches to Software Engineering. Springer, Cham, pp. 203–223, 2020.

[Pl18] Platzer, André: Logical Foundations of Cyber-Physical Systems, volume 662. Springer, 2018.

[Re15] Reschka, Andreas; Bagschik, Gerrit; Ulbrich, Simon; Nolte, Marcus; Maurer, Markus: Ability and Skill Graphs for System Modeling, Online Monitoring, and Decision Support for Vehicle Guidance Systems. In: 2015 IEEE Intelligent Vehicles Symposium (Vol. IV). IEEE, pp. 933–939, 2015.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 69

An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse

Jacob Krüger[1], Thorsten Berger[2]

Abstract: In this extended abstract, we summarize our paper with the homonymous title published at the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) 2020 [KB20].

Keywords: Economics; Software reuse; Empirical study; Clone & own; Software product line; Platform engineering

Software reuse is a core practice to reduce development costs and improve the quality of software systems. Organizations typically employ one of two strategies to reuse software:

Clone & Own. When employing this strategy, developers create an independent clone of an existing system and adapt it to new customer requirements.

Platform Orientation. When employing this strategy, developers implement a code base that allows them to derive customized variants based on concepts from product-line engineering, such as variability mechanisms (e.g., preprocessors) and feature models.

Either strategy has its pros and cons, enforcing different software architectures and development activities. Clone & own is cheap and readily available, for instance, in the form of branches in version-control systems—which is why most organization start reusing software based on this strategy. However, particularly when maintaining cloned variants becomes a burden, most organizations migrate these variants towards a platform—which requires high upfront investments, but promises to substantially reduce development and maintenance costs as well as the time-to-market of new variants.

Deciding which strategy to employ is a core decision for any organization and has long-term impact in its software development. Despite this importance, the costs resulting from either strategy are not well-understood and, in fact, there is a lack of systematically elicited data to provide guidance for organizations. In our paper, we describe an empirical study with which we elicited such data, investigating the development activities, costs, cost factors, and benefits of both reuse strategies. For this purpose, we combined a systematic literature

[1] Otto-von-Guericke-University Magdeburg, Germany
jkrueger@ovgu.de
[2] Ruhr-University Bochum, Germany & Chalmers | University of Gothenburg, Sweden
bergert@chalmers.se

doi:10.18420/SE2021_23

review of 57 publications that report quantified costs on software reuse with 26 interviews at a large organization. During the interviews, we elicited the software-reuse process employed at the organization. Furthermore, we asked interviewees to do informed, judgment-based cost estimations for a concrete variant they developed, and to assess the impact of cost factors on developing variants. We triangulated the data from both sources to confirm and refute common hypotheses on software reuse, providing concrete empirical data.

Among others, the core findings of our study are:

- The organization employs a development process that integrates clone & own with platform orientation. Such processes have recently also been identified for other organizations and open-source projects, but are still rarely considered in research.
- The success of reuse heavily depends on establishing platform orientation, with our data suggesting total median savings of 52 %. Particularly the reduced costs of variant development (-67 %) and quality assurance (-60 %) are major pros of a platform.
- A critical success factor for platform orientation is the quality of the code base, which is why fewer bugs are found (-70 %) in established platforms. Our data confirms additional benefits, such as a substantially reduced time-to-market (-63 %), and highlights the impact of various cost factors, particularly developers' knowledge regarding the software.

In total, we identified 18 pieces of confirmatory evidence for established hypotheses on software reuse, also indicating that a platform is preferable over clone & own—which does promise benefits, but on a lower level. Moreover, we identified seven inconclusive and three refuting pieces of evidence, suggesting that (1) clone & own and platform orientation are not strictly separated in practice; (2) change propagation can be more problematic in a platform compared to clone & own; and (3) platforms are essential to successfully establish innovative variants in new markets.

Our contribution is the first evidence-based body-of-knowledge on the costs of two established reuse strategies, providing important insights for research and practice. For instance, this body-of-knowledge helps practitioners understand both reuse strategies and improve their decision making, while we highlight open problems and particularly costly activities to provide guidance for further research. In future work, we will investigate our inconclusive and refuting evidence in more detail. Also, we will conduct additional studies to verify our data and derive a decision model that helps organizations decide for a reuse strategy.

Bibliography

[KB20] Krüger, Jacob; Berger, Thorsten: An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE. ACM, pp. 432–444, 2020.

What Developers (Care to) Recall: An Interview Survey on Smaller Systems

Jacob Krüger[1], Regina Hebig[2]

Abstract: This extended abstract summarizes our paper with the homonymous title published at the International Conference on Software Maintenance and Evolution (ICSME) 2020 [KH20].

Keywords: Knowledge; Information needs; Developers' memory

Developers have to understand the behavior and properties of the software in their system in order to extend and maintain it, which is referred to as *program comprehension*. While studying program comprehension, researchers have conducted empirical studies aiming to analyze the readability of source code (e.g., based on different identifier names), investigated developers' information needs (e.g., what questions come to a developer's mind during their tasks), and designed techniques to support knowledge recovery (e.g., by reverse engineering information). Interestingly, researchers have rarely investigated developers' memory decay or what knowledge they consider important to remember, and thus keep in their mind. Understanding what knowledge developers memorize helps to scope tools, practices, and research. For instance, experts may require more light-weight code searching capabilities, due to their memorized knowledge. In contrast, novices or new team members may require extensive documentation or the help of experts to understand the system architecture, which may be tacit knowledge experts aim to memorize. Moreover, such needs may vary depending on the type of knowledge, for example, the system's architecture versus meta information about the team or the system's evolution.

Motivated by our previous work, we investigated the connection between developers' memory decay, types of knowledge, and what knowledge developers consider important to remember. To this end, we started with a systematic literature review of 14 papers that are concerned with a total of 456 questions developers ask during maintenance and development tasks. We analyzed the 420 questions that the authors classified into 81 classes to gain a first understanding of developers' knowledge needs. Then, we re-classified all questions to unify the classes and get a consolidated overview. Building on our insights, we derived a semi-structured interview guide, which we used to conduct 17 interviews with developers from different areas (i.e., academia, industry, open-source), domains (e.g., web services, machine learning, static code analysis), and countries (e.g., Germany, Sweden, France). We

[1] Otto-von-Guericke-University Magdeburg, Germany
jkrueger@ovgu.de
[2] Chalmers | University of Gothenburg, Sweden
regina.hebig@cse.gu.se

doi:10.18420/SE2021_24

remark that most of the systems we asked our interviewees about were comparably small (i.e., 1–6 developers, four with more than 100 k lines of code). During each interview, we started with a self-assessment of the interviewee's familiarity (i.e., remaining knowledge) with the system overall and with respect to three types of knowledge (i.e., architecture, meta, code), which we repeated after each section related to these types. We then asked our interviewee to recall answers to six questions from the systematic literature review on each of the three different types of knowledge from their memory (i.e., a total of 18 different questions). Afterwards, but before checking for correctness, we asked each interviewee to rate the importance of the three knowledge types and individual questions.

Triangulating from the results of our systematic literature review and our interview survey, our core findings are:

- Developers consider more abstract knowledge about their system (e.g., the system architecture and intentions of the code) more important to remember.
- Developers can recall knowledge for questions they consider important more often correctly than for those they consider less important.
- Developers may be reliable in doing self-assessments of their familiarity, but, interestingly, these self-assessments usually decreased after answering questions about their system from memory.

In our paper, we report various additional insights that have important implications for practice and motivate new research directions. For instance, our results support our aforementioned assumptions that developers aim to remember a system's architecture. Consequently, practitioners have to think about how to document and maintain the corresponding information explicitly, to avoid that the tacit knowledge is lost over time. Moreover, our findings provide guidance on how to structure teams or onboard new developers, while researchers may explore new techniques for reverse engineering information. As direct future work, we are working on extensions of our study to overcome its limitations (e.g., small system sizes, number of participants). For this purpose, we are planning and conducting additional empirical studies (e.g., surveys, controlled experiments) to reinforce our findings.

Bibliography

[KH20] Krüger, Jacob; Hebig, Regina: What Developers (Care to) Recall: An Interview Survey on Smaller Systems. In: International Conference on Software Maintenance and Evolution. ICSME. IEEE, pp. 46–57, 2020.

Behavioral Interfaces for Executable DSLs

Dorian Leroy[1], Erwan Bousse[2], Manuel Wimmer[3], Tanja Mayerhofer[4], Benoit Combemale[5], Wieland Schwinger[6]

Abstract: This work summarizes our paper [Le20] originally published in the Journal of Software and Systems Modeling in 2020 about a novel language engineering approach.

Keywords: Domain Specific Languages, Metamodeling, Language Workbenches

A large amount of domain-specific languages (DSLs) are used to represent behavioral aspects of systems in the form of behavioral models [BCW17]. Executable domain-specific languages (xDSLs) enable the execution of behavioral models [Ma13]. While an execution is mostly driven by the model's content (e.g., control structures, conditionals, transitions, method calls), many use cases require interacting with the running model, such as simulating scenarios in an automated or interactive way or coupling system models with environment models. The management of these interactions is usually hard-coded into the semantics of xDSLs, which prevents its reuse for other xDSLs and the provision of generic interaction tools.

To tackle these issues, we propose a novel metalanguage for complementing the definition of xDSLs with explicit behavioral interfaces to enable external tools to interact with executable models in a unified way. A behavioral interface defines a set of events specifying how external tools can interact with models that conform to xDSLs implementing the interface. Additionally, we define two types of relationships involving behavioral interfaces: the implementation relationship and the subtyping relationship. An implementation relationship ties a behavioral interface to a given operational semantics implementation. Subtyping relationships allow to build event abstraction hierarchies, indicating that events from one interface can be abstracted or refined as events from another interface.

We implemented the proposed metalanguage in the GEMOC Studio, an Eclipse-based language and modeling workbench for xDSLs, and evaluate the approach with three

[1] Institute of Business Informatics - Software Engineering, JKU Linz, Altenbergerstr. 69, 4040 Linz, Austria dorian.leroy@jku.at
[2] Nantes Software Modeling Group, University of Nantes, Chemin de la Houssiniere, 44322 Nantes, France erwan.bousse@ls2n.fr
[3] Institute of Business Informatics - Software Engineering, JKU Linz, Altenbergerstr. 69, 4040 Linz, Austria manuel.wimmer@jku.at
[4] Business Informatics Group, TU Wien, Favoritenstr. 9-11, 1040 Wien, Austria mayerhofer@big.tuwien.ac.at
[5] DiverSE Team, University of Rennes 1, 263 Avenue du General Leclerc, 35042 Rennes, France benoit.combemale@irisa.fr
[6] Institute of Telecooperation, JKU Linz, Altenbergerstr. 69, 4040 Linz, Austria wieland.schwinger@jku.at

doi:10.18420/SE2021_25

demonstration cases: (*i*) we show that the proposed metalanguage can be used to define the behavioral interface of xDSLs; (*ii*) we show that behavioral interfaces enable the definition of generic tools and their reuse across several xDSLs; (*iii*) we show that a single behavioral interface can be subtyped by several xDSLs, allowing to interact with and reason about the execution of models through a common behavioral interface.

References

[BCW17] Brambilla, M.; Cabot, J.; Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool, 2017.

[Le20] Leroy, D.; Bousse, E.; Wimmer, M.; Mayerhofer, T.; Combemale, B.; Schwinger, W.: Behavioral interfaces for executable DSLs. Softw. Syst. Model. 19/4, pp. 1015–1043, 2020, URL: https://doi.org/10.1007/s10270-020-00798-2.

[Ma13] Mayerhofer, T.; Langer, P.; Wimmer, M.; Kappel, G.: xMOF: Executable DSMLs Based on fUML. In (Erwig, M.; Paige, R. F.; Wyk, E. V., eds.): Proceedings of the 6th International Conference on Software Language Engineering (SLE). Vol. 8225. Lecture Notes in Computer Science, Springer, pp. 56–75, 2013, URL: https://doi.org/10.1007/978-3-319-02654-1%5C_4.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 75

Feature-Modell-geführtes Online Reinforcement Learning für Selbst-adaptive Systeme

Andreas Metzger[1], Clément Quinton[2], Zoltán Mann[3], Luciano Baresi[4], Klaus Pohl[5]

Abstract: Wir stellen Lernstrategien für selbst-adaptive Systeme vor, welche Feature-Modelle aus der Software-Produktentwicklung nutzen, um den Lernprozess zur Laufzeit zu beschleunigen.

Keywords: Adaptation; Reinforcement Learning; Feature Modell; Cloud Service

1 Überblick

Motivation. Ein selbst-adaptives System ist in der Lage sich zur Laufzeit anzupassen und somit auf Veränderungen in seiner Umgebung zu reagieren. Eine Herausforderung bei der Entwicklung selbst-adaptiver Systeme ist festzulegen, wann und wie sich das System zur Laufzeit anpassen soll. Dies erfordert die Antizipation der möglichen Umgebungssituationen sowie genaue Kenntnis der Effekte der jeweiligen Adaptionen. Aufgrund von unvollständigem Wissen zur Design-Zeit ist dies im Allgemeinen nicht vollständig möglich [We19]. Eine Lösung ist Online Reinforcement Learning, also bestärkendes Lernen, das auf Basis von Laufzeit-Feedback geeignete Adaptionen lernt.

Problemstellung. Unser als [Me20] vorgestellter Beitrag adressiert zwei wesentliche Probleme bisheriger Online Reinforcement Learning Ansätze für selbst-adaptive Systeme: (1) Bisherige Ansätze probieren zufällig neue Adaptionen aus, um Feedback über die Wirksamkeit der Adaptation zur Laufzeit sammeln. Wird die Menge der Adaptionsmöglichkeiten groß, dann kann diese Zufallsauswahl zu einem langsamen Lernprozess führen. (2) Bisherige Ansätze können mit Änderungen der Umgebung während des Lernprozesses, sog. Nicht-Stationarität, umgehen (siehe z.B. [PMP20]). Mit Veränderungen der Systemlogik und somit einer Veränderung der möglichen Adaptationen können diese Verfahren jedoch nicht umgehen. Eine solche Veränderung der Systemlogik erfolgt typischerweise im Rahmen der Software-Evolution, durch welche neue Adaptionsmöglichkeiten hinzugefügt sowie bestehende verändert oder entfernt werden können.

[1] paluno, University of Duisburg-Essen, Essen, Germany, andreas.metzger@paluno.uni-due.de
[2] University of Lille, Inria, Lille, France,clement.quinton@univ-lille.fr
[3] paluno, University of Duisburg-Essen, Essen, Germany, zoltan.mann@paluno.uni-due.de
[4] Politecnico di Milano, Milan, Italy, luciano.baresi@polimi.it
[5] paluno, University of Duisburg-Essen, Essen, Germany, klaus.pohl@paluno.uni-due.de

doi:10.18420/SE2021_26

Lösungsansatz. Als Lösung schlagen wir Feature-Modell-geführte Lernstrategien vor. Wir nutzen Feature-Modelle aus der Software-Produktlinienentwicklung zur Spezifikation der Adaptionsmöglichkeiten in kompakter Form [MP14], wie in Abb. 1 illustriert.

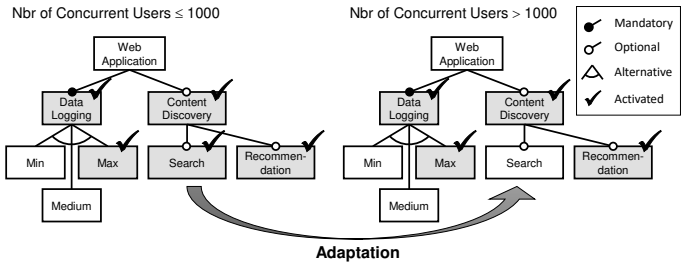


Abb. 1: Beispiel für die Spezifikation von Adaptionsmöglichkeiten mittels Feature-Modellen

Unsere Lernstrategien identifizieren z.B. mittels der Deltas zwischen den Variabilitätsmodellen vor und nach der Evolution die Veränderungen der Adaptionsmöglichkeiten. Diese Deltas werden genutzt, um den Lernprozess des Reinforcement-Learning-Agenten geeignet zu steuern. So können neu hinzugefügte Adaptionen z.B. mit einer höheren Wahrscheinlichkeit ausgeführt werden als bereits bekannte Adaptionen.

Experimentergebnisse. In unseren Experimenten erweitern wir den weit verbreiteten Q-Learning-Algorithmus um unsere Lernstrategien. An einem adaptiven Cloud Service konnten wir eine Beschleunigung des Lernprozesses von bis zu ca. 60% messen.[6]

Danksagung. Die Ergebnisse entstanden im Rahmen der europäischen H2020-Projekte ENACT (Förderkennzeichen 780351) und FogProtect (Förderkennzeichen 871525).

Literaturverzeichnis

[Me20] Metzger, Andreas; Quinton, Clément; Mann, Zoltán Ádám; Baresi, Luciano; Pohl, Klaus: Feature Model-Guided Online Reinforcement Learning for Self-Adaptive Services. In (Kafeza, Eleana; Benatallah, Boualem; Martinelli, Fabio; Hacid, Hakim; Bouguettaya, Athman; Motahari, Hamid, Hrsg.): 18th Int'l Conf. on Service-Oriented Computing (ICSOC 2020). Springer, LNCS 12571, 2020.

[MP14] Metzger, Andreas; Pohl, Klaus: Software product line engineering and variability management: Achievements and challenges. In (Herbsleb, James D.; Dwyer, Matthew B., Hrsg.): ICSE Future of Software Engineering Track (FOSE 2014). ACM, S. 70–84, 2014.

[PMP20] Palm, Alexander; Metzger, Andreas; Pohl, Klaus: Online Reinforcement Learning for Self-Adaptive Information Systems. In (Dustdar, Schahram; Yu, Eric; Salinesi, Camille; Rieu, Dominique; Pant, Vik, Hrsg.): 32nd Int'l Conf. on Advanced Information Systems Engineering (CAiSE 2020). Springer, LNCS 12127, S. 169–184, 2020.

[We19] Weyns, Danny: Software Engineering of Self-adaptive Systems. In (Cha, Sungdeok; Taylor, Richard N.; Kang, Kyo Chul, Hrsg.): Handbook of Software Engineering, S. 399–443. Springer, 2019.

[6] Code und Daten: `https://git.uni-due.de/online-reinforcement-learning/icsoc-2020-artefacts`

JAINT: A Framework for User-Defined Dynamic Taint-Analyses based on Dynamic Symbolic Execution of Java Programs

Malte Mues[1], Till Schallau[1], Falk Howar[1]

Abstract: We summarize the paper „Jaint: A Framework for User-Defined Dynamic Taint-Analyses Based on Dynamic Symbolic Execution of Java Programs“, published at the sixteenth international conference on integrated formal methods in November 2020 [MSH20]. Reliable and scalable methods for security analyses of JAVA applications are an important enabler for a secure digital infrastructure. In this paper, we present a security analysis that integrates dynamic symbolic execution and dynamic multi-colored taint analysis of JAVA programs, combining the precision of dynamic analysis with the exhaustive exploration of symbolic execution. We implement the approach in the JAINT tool, based on JDART [Lu16], a dynamic symbolic execution engine for JAVA PathFinder, and evaluate its performance by comparing precision and runtimes to other research tools on the OWASP benchmark set. The paper also presents a domain-specific language for taint analyses that is more expressive than the source and sink specifications found in publicly available tools and enables precise, CWE-specific specification of undesired data flows. This summary presents JAINT's language and the evaluation.

Keywords: Dynamic Symbolic Execution; Domain Specific Languages; Java Bytecode Analysis; Dynamic Taint Analysis

Specification of Taint Analyses

JAINT provides a domain-specific language for specifying undesired data flows from tainted sources to protected sinks that may be interrupted by flow through sanitization methods. The paper presents the grammar of the language along with usage examples. Here, we only present one small example: command injection attacks (CWE 78[2]) use parameters of a HTTP request as executable commands in a shell, i.e., in a command that is executed as a new process. Methods that match patterns `Runtime.exec(*)` and `ProcessBuilder.*(command)` are considered protected sinks:

$$Src ::= cmdi \quad \leftarrow (\_ : *\texttt{HttpServletRequest}).\texttt{get}*()$$
$$Sink ::= cmdi \quad \rightarrow (\_ : \texttt{java.lang.Runtime}).\texttt{exec}(*), (\_ : \texttt{java.lang.ProcessBuilder}).*(\texttt{command})$$

The zenodo archive accompanying the paper contains a version of JAINT with taint specifications for the eleven classes of CWEs in the OWASP benchmark set.[3]

[1] TU Dortmund, LS XIV Software Engineering - Automated Quality Assurance Group, Otto-Hahn-Str. 12, 44227 Dortmund, Deutschland {malte.mues,till.schallau,falk.howar}@tu-dortmund.de

[2] https://cwe.mitre.org/data/definitions/78.html

[3] `http://doi.org/10.5281/zenodo.4060244`

doi:10.18420/SE2021_27

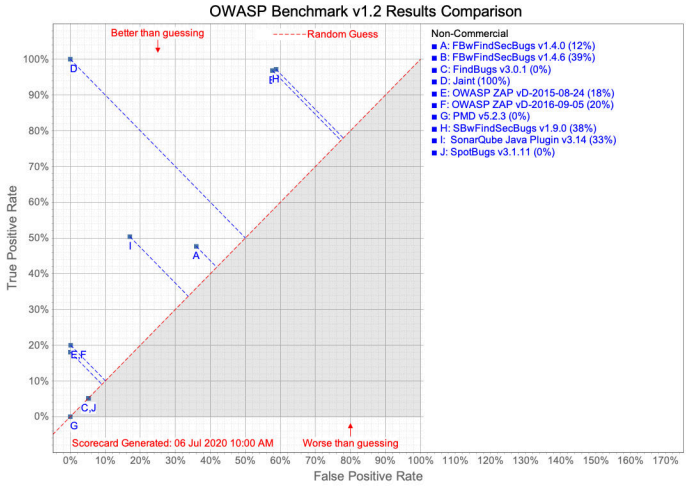


Fig. 1: Precision of JAINT on the OWASP benchmark set [MSH20].

Performance of JAINT on the OWASP benchmark set

Figure 1 shows the precision of JAINT on the OWASP benchmark set. The paper contains a detailed discussion and more obtained results, including runtimes. Most of the compared tools fall in one of two categories: dynamic analyses are precise but miss many vulnerabilities (lower left corner in the plot). Static analyses discover many vulnerabilities but suffer from high false positive rates (upper right corner of the plot). JAINT, in contrast, combines the exhaustive exploration of symbolic execution with the precision of dynamic analysis (upper left corner of the plot).

In the paper, we showed that JAINT beats the OWASP benchmark. This, of course, can only serve as initial validation of the approach: most benchmark instances consist only of few, easy to hit execution paths. We plan future work in two directions: (1) validation of JAINT on real-world examples, and (2) development of a more realistic benchmark set.

Bibliography

[Lu16] Luckow, Kasper Søe; Dimjasevic, Marko; Giannakopoulou, Dimitra; Howar, Falk; Isberner, Malte; Kahsai, Temesghen; Rakamaric, Zvonimir; Raman, Vishwanath: JDart: A Dynamic Symbolic Analysis Framework. In (Chechik, Marsha; Raskin, Jean-François, eds): Proceedings of TACAS 2016. volume 9636 of LNCS. Springer, pp. 442–459, 2016.

[MSH20] Mues, Malte; Schallau, Till; Howar, Falk: Jaint: A Framework for User-Defined Dynamic Taint-Analyses Based on Dynamic Symbolic Execution of Java Programs. In (Dongol, Brijesh; Troubitsyna, Elena, eds): Proceedings of IFM 2020. volume 12546 of LNCS. Springer, pp. 123–140, 2020.

Accurate Modeling of Performance Histories for Evolving Software Systems

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

Abstract: Learning from the history of a software system's performance behavior does not only help discovering and locating performance bugs, but also supports identifying evolutionary performance patterns and general trends. Exhaustive regression testing is usually impractical, because rigorous performance benchmarking requires executing realistic workloads per revision, resulting in large execution times. We devise a novel active revision sampling approach that aims at tracking and understanding a system's performance history by approximating the performance behavior of a software system across all of its revisions. In short, we iteratively sample and measure the performance of specific revisions to learn a performance-evolution model. We select revisions based on how uncertainty our models predicts their correspondent performance values. Technically, we use Gaussian Process models that not only estimates performance for each revision, but also provides an uncertainty value alongside. This way, we iteratively improve our model with only few measurements. Our evaluation with six real-world configurable software system demonstrates that Gaussian Process models are able to accurately estimate the performance-evolution histories with only few measurements and to reveal interesting behaviors and trends, such as change points.

Keywords: Software Performance; Software Evolution; Test Prioritization

Summary

Throughout a software system‘s development history, its non-functional properties, such as performance, evolve alongside. Individual modifications of the code base (*revisions*) or batches thereof can entail changes in performance. Unless identified and addressed, detrimental performance changes can add up to performance degrading over time. The retrospective analysis of existing histories can unveil causative revisions and, subsequently, help prioritize revisions for future performance regression testing. As performance measurements come at a considerable cost, it is intractable to assess all revisions. Instead, the challenge is to find a trade-off between measurement effort and accuracy of estimating performance.

We devise a novel probabilistic *active learning* algorithm to *accurately* approximate the performance history of a software system based on measurements of a specific workload *with few measurements* [MAS19]. Our approach is not only able to provide performance

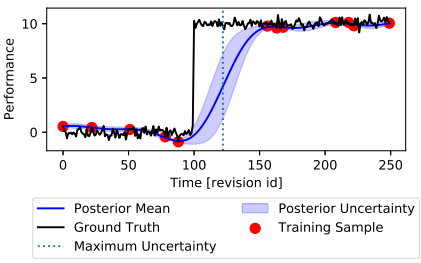
[1] Leipzig University, Institute of Computer Science, muehlbauer@informatik.uni-leipzig.de
[2] Saarland University, Saarland Informatics Campus, apel@cs.uni-saarland.de
[3] Leipzig University, Institute of Computer Science, norbert.siegmund@uni-leipzig.de

doi:10.18420/SE2021_28

estimations for all revisions, but also reports an uncertainty measure alongside. We use this uncertainty measure to decide for each revision whether our estimation is sufficiently accurate or whether we need to refine the approximation by including more measurements. To increase reliability where necessary, the algorithm selects and prioritizes new revisions for performance measurement based on the reported uncertainty and relearns the underlying Gaussian Process model.

We use *Gaussian Processes* (GPs) for time series data as a framework to model the performance history of a software system and obtain respective estimations. In a nutshell, a GP can be conceived as a distribution over functions (here: performance as a function of revisions). Evaluating the GP for a revision will yield a Gaussian $\mathcal{N}(\mu, \sigma)$ with a mean performance estimate μ and a variance measure σ. The variance σ is lower around revisions for which we have actual performance measurements at hand and can be interpreted as a measure of prediction uncertainty. The shape of an approximated performance history is determined by the GP's covariance function – a hyper parameter often called *kernel*. The kernel encodes further properties of the modeled performance histories, such as whether to expect a continuous estimation. At large, we evaluate the GP for all revisions to obtain an approximation as in Fig. 1 with regions of low and high uncertainty, the latter indicates the need for further measurements. The key idea of our approach is the following: We let the uncertainty measures *guide* the selection of new revisions to measure performance for. That is, we interpret the prediction uncertainty as a measure of how much we expect this measurement to improve the overall prediction accuracy. We repeatedly estimate performance across all revisions and add new measurements until the minimum uncertainty falls below a user-specified threshold.

Fig. 1: GP approximation of a performance history with a single change point (at revision 120).



We perform a series of experiments with the six real-world subject system from a variety of domains (file compression, scientific computing, image processing). Across different kernels evaluated, we obtained the most accurate approximations of performance histories in setups with the Brownian kernel. From such approximations, we are able to identify and pinpoint change points to individual revisions.

Bibliography

[MAS19] Mühlbauer, Stefan; Apel, Sven; Siegmund, Norbert: Accurate Modeling of Performance Histories for Evolving Software Systems. In: Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 640–652, 2019.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 81

MoFuzz: A Fuzzer Suite for Testing Model-Driven Software Engineering Tools – Summary

Hoang Lam Nguyen[1] Nebras Nassar[2] Timo Kehrer[1] Lars Grunske[1]

Abstract: Fuzzing or fuzz testing is an established technique that aims to discover unexpected program behavior (e. g., bugs, vulnerabilities, or crashes) by feeding automatically generated data into a program under test. However, the application of fuzzing to test Model-Driven Software Engineering (MDSE) tools is still limited because of the difficulty of existing fuzzers to provide structured, well-typed inputs, namely models that conform to typing and consistency constraints induced by a given meta-model and underlying modeling framework. We present three different approaches for fuzzing MDSE tools: A graph grammar-based fuzzer and two variants of a coverage-guided mutation-based fuzzer working with different sets of model mutation operators. Our evaluation on a set of real-world MDSE tools shows that our approaches can outperform both standard fuzzers and model generators w.r.t. their fuzzing capabilities. Moreover, we found that each of our approaches comes with its own strengths and weaknesses in terms of code coverage and fault finding capabilities, thus complementing each other and forming a fuzzer suite for testing MDSE tools.

Keywords: Model-Driven Software Engineering; Modeling Tools; Fuzzing; Automated Model Generation; Eclipse Modeling Framework

1 Summary

Fuzzing (also known as *fuzz testing*) automatically generates a large number of inputs and feeds them to the program under test to discover unexpected program behavior and evaluate the program's reliability. In our work, we investigate the fuzzing of Model-Driven Software Engineering (MDSE) tools which are based on the Eclipse Modeling Framework (EMF). Fuzzing MDSE tools is a challenging task, since (i) the test inputs must adhere to complex input constraints (e. g., well-typedness and valid multiplicities w.r.t. the input meta model) in order to pass the initial syntactic and semantic validation stages of the input processing pipeline, and (ii) the generated input models must be interesting/diverse enough to exercise a variety of code paths.

Building upon recent advances in automated model generation and structure-aware fuzzing, we propose three different fuzzers as part of our fuzzer suite MoFuzz [Ng20]: a graph-grammar based fuzzer and two mutation-based approaches. The graph grammar-based fuzzer is based on the recently introduced EMF Model Generator [Na20], which is able to efficiently generate large, properly-typed EMF models with valid multiplicities. Conceptually,

[1] Humboldt-Universität zu Berlin, Germany, {nguyehoa,kehrer,grunske}@informatik.hu-berlin.de
[2] Philipps-Universität Marburg, Germany, nassarn@informatik.uni-marburg.de

doi:10.18420/SE2021_29

the fuzzer first translates the meta-model into a constructive language specification (i. e., the grammar), which is then leveraged to generate models in a two-phased approach. First, the *model increase* phase creates model elements without violating upper multiplicity bounds. Then, the *model completion* phase completes the intermediate model to a valid EMF model. Overall, the graph-grammar based fuzzer attempts to *broadly* explore the space of valid instance models in an efficient manner. The mutation-based fuzzers are based on a widely used technique in automated fault detection, namely coverage-guided fuzzing (CGF) [LZZ18], which we adapt to the domain of MDSE as follows. First, a random set of seed models is generated using automated model generation techniques to initialize the input queue. Afterwards, both approaches continuously select an input model from the queue, apply model-based mutations on it, and retain the mutated input only if it increases coverage. The goal is to incrementally evolve the inputs in the queue to exercise deep paths. While both fuzzers essentially employ mutations that add, delete, or change model elements, one uses generic mutation operators based on the EMF Edit API, whereas the other automatically derives consistency-preserving mutation operators from the meta-model [Ke16].

Our implementation of MoFuzz builds upon JQF [PLS19], a feedback-directed fuzz testing framework for Java. We have evaluated MoFuzz gainst the Zest algorithm implemented by JQF, and the EMF random instantiator [At15] on a set of real-world MDSE tools. The results of our evaluation indicate that MoFuzz can improve code coverage as well as the number of exposed crashes when fuzzing MDSE tools. In terms of coverage, the graph grammar-based fuzzer of MoFuzz performed the best, while the mutation-based fuzzer using EMF Edit API mutations triggered the most crashes.

Bibliography

[At15] AtlanMod: EMF Random Instantiator. `https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator/`, 2015. Accessed: November 24, 2020.

[Ke16] Kehrer, Timo; Taentzer, Gabriele; Rindt, Michaela; Kelter, Udo: Automatically Deriving the Specification of Model Editing Operations from Meta-Models. In: 9th International Conference on Theory and Practice of Model Transformations (ICMT). pp. 173–188, 2016.

[LZZ18] Li, Jun; Zhao, Bodong; Zhang, Chao: Fuzzing: a survey. Cybersecurity, 1(1):6, 2018.

[Na20] Nassar, Nebras; Kosiol, Jens; Kehrer, Timo; Taentzer, Gabriele: Generating Large EMF Models Efficiently - A Rule-Based, Configurable Approach. In: 23rd International Conference on Fundamental Approaches to Software Engineering (FASE). Springer, pp. 224–244, 2020.

[Ng20] Nguyen, Hoang Lam; Nassar, Nebras; Kehrer, Timo; Grunske, Lars: MoFuzz: A Fuzzer Suite for Testing Model-Driven Software Engineering Tools. In: 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020.

[PLS19] Padhye, Rohan; Lemieux, Caroline; Sen, Koushik: JQF: Coverage-Guided Property-Based Testing in Java. In: 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). pp. 398–401, 2019.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 83

Cooperative Android App Analysis with CoDiDroid

Felix Pauck[1], Heike Wehrheim[2]

Abstract: Novel Android app analysis tools as well as improved versions of available tools are frequently proposed. These proposed tools often tackle a specific single issue that cannot be handled with existing tools. Consequently, the best analysis possible should use the advantages of each and every tool. With CoDiDroid we present an analysis framework that allows to combine analysis tools such that the best out of each tool is used for a more comprehensive and more precise cooperative analysis. Our experimental results show indeed that CoDiDroid allows to setup cooperative analyses which are beneficial with respect to effectiveness, accuracy and scalability.

Keywords: Android Taint Analysis; Tools; Cooperation; Precision

1 Cooperative Analysis

Combinations of different Android app analysis tools have been used before to enhance existing analysis. One well-known example is IccTA, a tool that beneficially combines IC3 with FlowDroid. Thereby the intra-component analysis of FlowDroid is lifted-up to inter-component level. However, this combination is hard-coded in the tool itself. Its components cannot be swapped out and the analysis cannot be extended by including another tool without adapting IccTA. In a *cooperative analysis* as proposed by Pauck and Wehrheim [PW19] it is possible to combine arbitrary analyses. The following example illustrates what a cooperative analysis is capable of. The section thereafter introduces the framework required to compose a cooperative Android app analysis, namely CoDiDroid [Pa19].

Example
The example depicted in Figure 1 shows two *taint flows* that leak sensitive information. A taint flow describes the connection of a *source* and a *sink*. A source extracts sensitive information. In the example the device identification number represents the extracted sensitive data which is read in statement s_1 (see Figure 1). Two sinks (s_4, s_7) may leak this information via logging or sending an SMS. In order to detect both leaks an analysis tool must be (1) lifecycle-aware, (2) able to resolve reflection, (3) analyze native code and (4) successfully handle Inter-App-Communication (IAC). Sadly, there exists no such analysis tool. However, FlowDroid is able to perform an intra-component, lifecycle-aware taint analysis, DroidRA is able to resolve reflection, NOAH can discover sources and sinks in native code, IC3 allows to gather information about an app's exit and entry points which can be used by PIM to find connections between those that are realized via intents (IAC).

[1] Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany fpauck@mail.uni-paderborn.de
[2] Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany wehrheim@uni-paderborn.de

doi:10.18420/SE2021_30

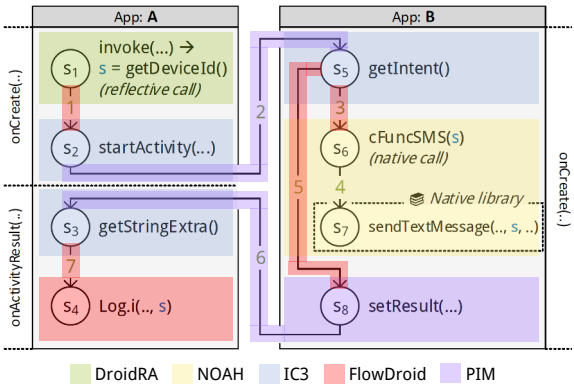


Fig. 1: Example: Two taint flows that include reflection, IAC and a native method call

Most of these tools were designed as standalone tools without having cooperation in mind. Nonetheless, CoDiDroid allows to compose a single analysis that not least successfully analyzes the described example by detecting both taint flows. The figure visualizes which parts of these taint flows are detected by which tool.

2 The CoDiDroid framework

The cooperative (and distributed) analysis framework CoDiDroid takes (1) a task in form of a query as input, (2) generates subtasks to answer each part of the query, (3) distributes these subtasks onto tools available in its configuration and in the end (4) merges tool answers to respond to the initial query. The Android App Analysis Query Language (AQL) and its execution system (AQL-System) is extensively used to do so. Along with CoDiDroid we thus proposed AQL-WebServices that allow to execute tools in different environments and exchange results. Also by employing the AQL, ReproDroid [PBW18] could be used to evaluate CoDiDroid against the state-of-the-art, revealing that cooperative analysis pays off by outperforming standalone tools with respect to effectiveness, scalability and accuracy in terms of precision, recall and F-measure.

Note, detailed evaluation results and all the reference of the mentioned tools can be found in the original paper [PW19].

Bibliography

[Pa19] Pauck, Felix: , CoDiDroid, 2019. `https://FoelliX.github.io/CoDiDroid` last accessed 11/16/2020.

[PBW18] Pauck, Felix; Bodden, Eric; Wehrheim, Heike: Do Android taint analysis tools keep their promises? In: Proceedings of ESEC/FSE 2018, Lake Buena Vista, FL, USA. ACM, 2018.

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

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 85

Generating Tests that Cover Input Structure

Nataniel Pereira Borges Jr.[1] Nikolas Havrikov[2] Andreas Zeller[3]

Abstract: To systematically test a program, one needs good *inputs*—inputs that are *valid* such that they are not rejected by the program, and inputs that *cover* as much of the input space as possible in order to reach a maximum of functionality.

We present *recent techniques to systematically cover input structure.* Our k-path algorithm for grammar production [HZ19] systematically covers syntactic elements of the input as well as their combinations. We show how to *learn* such input structures from graphical user interfaces, notably their *interaction language* [DBJZ19]. Finally, we demonstrate that knowledge bases such as DBPedia can be a reliable source of semantically coherent inputs [Wa20]. All these techniques result in a significantly higher code coverage than state of the art.

Keywords: grammar; coverage; automated testing; input generation; knowledge-base; android

1 Achieving Grammar Coverage

Testing programs with randomly generated inputs is a cost-effective means to test programs for robustness. However, to reach deep layers of a program, the inputs must be syntactically valid. Using a grammar to specify the language of program inputs lends itself well to solving this problem: A grammar-based test generator uses such a grammar to expand its start symbol into further symbols repeatedly until only terminal symbols are left, constituting an input. When generating inputs, intuitively, a high variation in the inputs should lead to a high variation in program behavior.

We present a notion of grammar coverage called k-path coverage and an approach for quickly achieving it. A k-path consists of k consecutive symbols along a valid derivation sequence in a derivation tree or a grammar. For any given grammar the number of k-paths is finite and one can generate a set of inputs exhibiting all of them by greedily deriving towards yet unvisited k-paths while keeping track of any k-paths covered incidentally. Having fully derived a targeted k-path, we promptly close off the current tree as quickly as possible and start generating a new one for the next unvisited k-path.

[1] CISPA Helmholtz Center for Information Security, Saarbrücken nataniel.borges@cispa.de
[2] CISPA Helmholtz Center for Information Security, Saarbrücken nikolas.havrikov@cispa.de
[3] CISPA Helmholtz Center for Information Security, Saarbrücken zeller@cispa.de

doi:10.18420/SE2021_31

2 UI Element Interactions

In the context of testing a mobile app, automated test generators systematically identify and interact with its user interface elements. One key challenge hereby is to synthesize inputs that effectively and efficiently cover app behavior. This is usually approached by having a model mapping UI elements to actions they usually accept. Such a model can be mined statically from an app or dynamically from observing its executions. Both these approaches, however, are biased towards the distribution originally mined. They work well if the app under test is similar to those used to train the model, but fail if it is dissimilar.

We present a technique that automatically adapts the model to the app at hand by approaching test generation as an instance of the multi-armed bandit problem, where a finite set of resources (actions) has to be distributed among competing alternatives (UI elements) to increase its reward (test quality). We use reinforcement learning to address test generation from this perspective and to systematically and gradually adjust our test generation strategy towards the application under test.

3 Using Knowledge Bases

Staying in the context of mobile apps, many take complex data as input, such as travel booking, map locations, or online banking information. These inputs are, however, expensive to generate manually and challenging to synthesize automatically. Past research indicated that knowledge bases could be a reliable source of semantically coherent inputs.

We propose an approach for leveraging knowledge bases for mobile app test generation comprising four steps: Given a UI state, we start by identifying and matching descriptive labels with input fields according to a set of metrics based on the Gestalt principles. We then use natural language processing to extract a *concept* associated with each label. We use the extracted concepts, instead of the original labels, to query knowledge bases for input values. Finally, we fill all input elements with the queried values and randomly interact with the non-input elements.

Bibliography

[DBJZ19] Degott, Christian; Borges Jr., Nataniel Pereira; Zeller, Andreas: Learning User Interface Element Interactions. In: ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019). July 2019.

[HZ19] Havrikov, Nikolas; Zeller, Andreas: Systematically Covering Input Structure. In: IEEE/ACM International Conference on Automated Software Engineering (ASE 2019). November 2019.

[Wa20] Wanwarang, Tanapuch; Borges Jr., Nataniel Pereira; Bettscheider, Leon; Zeller, Andreas: Testing Apps With Real-World Inputs. In: 1st IEEE/ACM International Conference on Automation of Software Test (AST 2020). May 2020.

What Am I Testing and Where? Comparing Testing Procedures Based on Lightweight Requirements Annotations

Florian Pudlitz[1] Florian Brokhausen[2] Andreas Vogelsang[3]

Abstract:

The article was originally published in the international journal Empirical Software Engineering with the title "What am I testing and where? Comparing testing procedures based on lightweight requirements annotations" [PBV20].

Keywords: Annotation; Requirements modeling; Test management; Test evaluation; Simulation

1 Overview

Software complexity has increased dramatically in many areas in recent years, for example due to increasing automation or stronger interconnectivity between devices. This results in growing challenges in requirements and test management. Nowadays, system requirements are often written in natural language, which makes automated processing more difficult. The goal is to ensure that all requirements are checked despite the increasing complexity of the test cases. Previous test procedures use a transformation of requirements specification into test specification with consideration of the traceability of test results. However, these procedures reach their limits in complex test scenarios in different test levels, because the system runs through several situations automatically. For example, when testing driver assistance systems in real test drives or in traffic simulations with several hundred vehicles, new test approaches are required. In addition, it is not yet possible to make any statements about the similarity of test levels or test scenarios within a test level.

Our approach is based on a Multilevel Markup Language for annotating text passages in natural language requirements [PVB19]. The test engineer has the possibility to mark up text passages and observe them in test runs. After a test run, the log data is evaluated and the results can be displayed in relation to the annotations in the natural language requirements. Manual annotation can be partially automated by using machine learning algorithms [PBV19].

[1] Technische Universität Berlin, Fachgebiet Distributed and Operating Systems, Straße des 17. Juni 135, 10623 Berlin, Deutschland, florian.pudlitz@tu-berlin.de
[2] Technische Universität Berlin, Fachgebiet für Fluidsystemdynamik, Straße des 17. Juni 135, 10623 Berlin, Deutschland, florian.brokhausen@tu-berlin.de
[3] Universität zu Köln, Lehrstuhl für Software and Systems Engineering, Weyertal 121, 50931 Köln, Deutschland, vogelsang@cs.uni-koeln.de

doi:10.18420/SE2021_32

Our experimental evaluation, shown schematically in Figure 1, investigates four different evaluation foci based on annotations of the Multilevel Markup Language. First, it investigates large scale usage of the markup language, the annotation scalability. Second, we examine Test Case Allignment, which examines how well the test levels fit the requirements. Third, in Test Stage Compliance, we compare the test levels with respect to the annotations in the requirements. Fourthly, we compare the test scenarios that have been performed, the Test Stage Similarity.

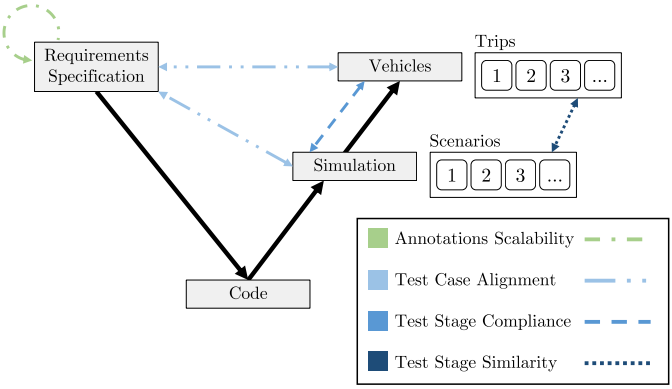


Fig. 1: Schematics of the experimental setup.

We investige two different test stages. On the one hand, we chose a traffic simulation with 1300 vehicles for all evaluations carried out, since simulations are becoming increasingly important, especially in the automotive sector. On the other hand, we used real driving data from 53 test drives. Investigations have shown that the use of field user data, especially in the automotive area, has been of little importance so far, but will be further expanded in the future [EBV20].

2 Results

Our experiment shows how different test levels are linked to natural language requirements. The test engineer receives not only concrete evaluations of the annotations but also holistic statements about test coverage of requirements. With regards to the performed simulation, 75 (25.5%) of the 294 requirements exclusively contain fulfilled annotations. The evaluation of the real trips shows that 80 (27.2%) of the considered requirements contain fulfilled annotations only. In contrast, 75 (25.5%) requirements are entirely unfulfilled within the simulation with regards to the inherent annotations and 42 (14.3%) with respect to real driving data.

Bibliography

[EBV20] Ebel, Patrick; Brokhausen, Florian; Vogelsang, Andreas: The Role and Potentials of Field User Interaction Data in the Automotive UX Development Lifecycle: An Industry Perspective. Association for Computing Machinery, New York, NY, USA, p. 141–150, 2020.

[PBV19] Pudlitz, Florian; Brokhausen, Florian; Vogelsang, Andreas: Extraction of System States from Natural Language Requirements. In: 27th IEEE International Requirements Engineering Conference (RE). 2019.

[PBV20] Pudlitz, Florian; Brokhausen, Florian; Vogelsang, Andreas: What Am I Testing and Where? Comparing Testing Procedures based on Lightweight Requirements Annotations. Empirical Software Engineering, 2020.

[PVB19] Pudlitz, Florian; Vogelsang, Andreas; Brokhausen, Florian: A Lightweight Multilevel Markup Language for Connecting Software Requirements and Simulations. In (Knauss, Eric; Goedicke, Michael, eds): Requirements Engineering: Foundation for Software Quality. Springer International Publishing, Cham, pp. 151–166, 2019.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 91

A Domain Analysis of Resource and Requirements Monitoring: Towards a Comprehensive Model of the Software Monitoring Domain

Rick Rabiser[1], Klaus Schmid[2], Holger Eichelberger[2], Michael Vierhauser[3], Paul Grünbacher[4]

Abstract: This is a summary of an article (with the same title) we published in the Information and Software Technology Journal in 2019 describing a domain model we developed to structure and systematize the field of software monitoring as well as a reference architecture to support developing software monitoring approaches.

Keywords: Software monitoring; Requirements monitoring; Resource monitoring; Domain model; Reference architecture

1 Summary

Complex software systems need to be monitored as their full behavior only emerges at run-time, e.g., when interacting with other systems or their environment. Diverse software monitoring approaches [Ra19, Ra17] have been developed in diverse communities for various kinds of systems and purposes. They observe and check properties or quality attributes of software systems during operation. For instance, requirements monitoring approaches check at run-time whether a software system adheres to its requirements, while resource or performance monitoring approaches collect information about the consumption of computing resources by the monitored system. Many venues publish research on software monitoring, often using diverse terminology, and focusing on different monitoring aspects and phases. We provide a domain model to structure and systematize the field of software monitoring, starting with requirements and resource monitoring. Based on earlier efforts, we systematically analyzed 47 existing requirements and resource monitoring approaches to iteratively refine the domain model and also to develop a reference architecture for software monitoring approaches. Our domain model covers the key elements of monitoring approaches and allows analyzing their commonalities and differences. Together with the reference architecture, our domain model supports the development of integrated monitoring solutions.

[1] LIT CPS, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, rick.rabiser@jku.at
[2] SSE, University of Hildesheim, Germany, schmid@sse.uni-hildesheim.de
[3] SE, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, michael.vierhauser@jku.at
[4] ISSE, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, paul.gruenbacher@jku.at

doi:10.18420/SE2021_33

Bibliography

[Ra17] Rabiser, Rick; Guinea, Sam; Vierhauser, Michael; Baresi, Luciano; Grünbacher, Paul: A Comparison Framework for Runtime Monitoring Approaches. Journal of Systems and Software, 125:309–321, 2017.

[Ra19] Rabiser, Rick; Schmid, Klaus; Eichelberger, Holger; Vierhauser, Michael; Guinea, Sam; Grünbacher, Paul: A Domain Analysis of Resource and Requirements Monitoring: Towards a Comprehensive Model of the Software Monitoring Domain. Information and Software Technology, 111:86–109, 2019.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 93

Tool Support for Correctness-by-Construction

Tobias Runge[1] Ina Schaefer[2] Loek Cleophas[3] Thomas Thüm[4] Derrick Kourie[5] Bruce W. Watson[6]

Abstract: This work was published at the International Conference on Fundamental Approaches to Software Engineering (FASE) 2019 [Ru19]. We tackled a fundamental problem of missing tool support of the correctness-by-construction (CbC) methodology that was proposed by Dijsktra and Hall and revised to a light-weight and more amenable version by Kourie and Watson. Correctness-by-construction (CbC) is an incremental approach to create programs using a set of small, easily applicable refinement rules that guarantee the correctness of the program with regard to its pre- and postcondition specifications. Our goal was to implement a functional and user-friendly IDE, so that developers will adapt to the CbC approach and benefit from its advantages (e.g., defects can be easily tracked through the refinement structure of the program). The tool has a hybrid textual and graphical IDE that programmers can use to refine a specification into a correct implementation. The textual editor fits programmers that want to stay in their familiar environment, while the graphical editor highlights the refinement structure of the program to give visual feedback if errors occur, using KeY as verification backend. The tool was evaluated regarding feasibility and effort to develop correct programs. Here, slight benefits in comparison to a standard verification approach were discovered.

Keywords: correctness-by-construction; tool support; formal methods; verification

Overview

Correctness-by-Construction (CbC) [KW12] is a methodology to construct formally correct programs guided by a pre-/postcondition specification. Starting with an abstract program, formally verified refinement rules are applied to incrementally refine the program to a concrete implementation. In the literature [KW12, Wa16], CbC is described as having many benefits: The structured reasoning discipline that is enforced by the refinement rules reduces the appearance of defects. If defects do occur, they can be tracked through the refinement structure. Furthermore, the formal process increases trust in the program. To check these benefits, we implement tool support that enables CbC to be used by programmers. We want to compare CbC with the prevalent post-hoc verification approach, where program correctness is proven after construction.

[1] TU Braunschweig, Germany tobias.runge@tu-bs.de
[2] TU Braunschweig, Germany i.schaefer@tu-bs.de
[3] TU Eindhoven, The Netherlands and Stellenbosch University, South Africa l.g.w.a.cleophas@tue.nl
[4] Ulm University, Germany thomas.thuem@uni-ulm.de
[5] Stellenbosch University, South Africa derrick@fastar.org
[6] Stellenbosch University, South Africa bruce@fastar.org

doi:10.18420/SE2021_34

In this work, we present CorC, an IDE that supports the CbC approach with a hybrid textual and graphical user interface. The IDE support users to apply refinement rules to an abstract program until the program is fully refined. In each refinement step, the correct application is guaranteed by using the theorem prover KeY [Ah16]. Each proof obligation can be immediately discharged during program development. In the textual editor, programmers enrich source code with specification and directly see if a refinement is unprovable. The graphical editor is useful to get an overview of all refinement steps and track errors in the program. To not burden programmers, they can switch automatically between both views. As CbC is not tailored to a specific host language, we implemented CorC in such a way that the language can be exchanged: Any imperative programming language with a specification language and an automatic verification tool can be integrated.

In our evaluation, we compared CorC with standard post-hoc verification using KeY as prover in both cases. We investigated the hypothesis whether the verification of algorithms is faster with CorC than with post-hoc verification. Therefore, we implemented seven algorithms with CorC and as plain Java code with specification. In each case, we measured that the verification time was lower for CorC, indicating a reduced proof complexity. The result is even statistically significant which provides empirical evidence for our hypothesis.

In summary, we extended the KeY ecosystem with tool support for the correctness-by-construction methodology. With CorC, programmers can utilize CbC to construct correct programs and use the results to bootstrap post-hoc verification as an additional check if necessary. They can reduce the verification time, as demonstrated in our evaluation, and benefit from synergistic effects of both approaches.

Bibliography

[Ah16] Ahrendt, Wolfgang; Beckert, Bernhard; Bubel, Richard; Hähnle, Reiner; Schmitt, Peter H; Ulbrich, Mattias: Deductive Software Verification–The KeY Book: From Theory to Practice, volume 10001. Springer, 2016.

[KW12] Kourie, Derrick G; Watson, Bruce W: The Correctness-by-Construction Approach to Programming. Springer Science & Business Media, 2012.

[Ru19] Runge, Tobias; Schaefer, Ina; Cleophas, Loek; Thüm, Thomas; Kourie, Derrick; Watson, Bruce W.: Tool Support for Correctness-by-Construction. In: Fundamental Approaches to Software Engineering. volume 11424 of Lecture Notes in Computer Science. Springer, pp. 25–42, 2019.

[Wa16] Watson, Bruce W; Kourie, Derrick G; Schaefer, Ina; Cleophas, Loek: Correctness-by-Construction and Post-hoc Verification: A Marriage of Convenience? In: International Symposium on Leveraging Applications of Formal Methods. volume 9952 of Lecture Notes in Computer Science. Springer, pp. 730–748, 2016.

Trace Link Recovery Using Semantic Relation Graphs and Spreading Activation

Aaron Schlutter[1], Andreas Vogelsang[2]

Abstract: The paper was first published at the 28th IEEE International Requirements Engineering Conference in 2020. Trace Link Recovery tries to identify and link related existing requirements with each other to support further engineering tasks. Existing approaches are mainly based on algebraic Information Retrieval or machine-learning. Machine-learning approaches usually demand reasonably large and labeled datasets to train. Algebraic Information Retrieval approaches like distance between tf-idf scores also work on smaller datasets without training but are limited in providing explanations for trace links. In this work, we present a Trace Link Recovery approach that is based on an explicit representation of the content of requirements as a semantic relation graph and uses Spreading Activation to answer trace queries over this graph. Our approach is fully automated including an NLP pipeline to transform unrestricted natural language requirements into a graph. We evaluate our approach on five common datasets. Depending on the selected configuration, the predictive power strongly varies. With the best tested configuration, the approach achieves a mean average precision of 40% and a Lag of 50%. Even though the predictive power of our approach does not outperform state-of-the-art approaches, we think that an explicit knowledge representation is an interesting artifact to explore in Trace Link Recovery approaches to generate explanations and refine results.

Keywords: Traceability; Natural Language; Semantic Relation Graph; Spreading Activation

Trace Link Recovery (TLR) is a common problem in software engineering. While many tasks profit from links between related development artifacts, these are laborious to maintain manually and therefore rarely exist in projects. Automatic approaches aim for supporting engineers in finding related artifacts and creating trace links. Information Retrieval (IR) approaches build upon the assumption that if engineers refer to the same aspects of the system, similar language is used. Thus, tools suggest trace links based on Natural Language (NL) content.

State-of-the-art approaches use algebraic IR models (e.g., VSM, LSI), probabilistic models (e.g., LDA), or machine-learning approaches. These approaches rely on implicit models of key terms in documents (e.g., as points in a vector space or as probability distribution). Trace links are recovered based on similarity notions defined over these models. Therefore, it is hard to analyze and explain *why* specific trace links are identified in the model. Another drawback of machine-learning approaches is the need to train the models on reasonably large datasets. However, datasets usually consists of less than 500 artifacts (at least the ones used in scientific publications). [BRA14]

[1] Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin, Deutschland aaron.schlutter@tu-berlin.de
[2] Universität zu Köln, SSE, Weyertal 121, 50931 Köln, Deutschland vogelsang@cs.uni-koeln.de

doi:10.18420/SE2021_35

In our paper [SV20], we present a novel approach for TLR using semantic relations between parts of NL, stored in a semantic relation graph, and search trace links by Spreading Activation, a semantic search graph algorithm. While the approach is fully automated, it does not have any prerequisites with regard to the format or content of natural language and is scalable to various sizes of corpora. To improve the user confidence, we are able to generate an explanation between each query and target requirement by identifying and highlighting the contributing text passages.

The semantic relation graph is an explicit model of the knowledge represented in requirements. Our pipeline translates them automatically into vertices and edges that depict semantic parts of common NL (e.g., words and phrases within sentences, but also documents and corpora). The structure supports that single words have a greater distance (i.e., are less relevant) to a certain specification than phrases or whole statements. We use Spreading Activation [Ha19] to identify related requirements. The graph algorithm spreads activation in pulses over the vertices starting from a query vertex. Vertices with higher activation indicate higher relevance. Thus, we build a candidate list to sort all (reachable) targets based on their relations.

We applied the approach on 5 datasets [HHPL19] from different domains commonly used in TLR research, and evaluated in terms of *mean average precision* and *Lag* for answer sets of 5, 10, and 30 candidates. With the best tested configuration, our approach achieves an average precision around 40% and a Lag around 50%. While this performance does not outperform existing state-of-the-art approaches, the explicit representation of requirements content allows to "follow" a trace link through a chain of statements that may serve as an explanation why a trace link exists.

Bibliography

[BRA14] Borg, Markus; Runeson, Per; Ardö, Anders: Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering (EMSE), 19(6):1565–1616, 2014. https://doi.org/10.1007/s10664-013-9255-y.

[Ha19] Hartig, Kerstin: Entwicklung eines Information-Retrieval-Systems zur Unterstützung von Gefährdungs- und Risikoanalysen. PhD thesis, Technische Universität Berlin, 2019. https://doi.org/10.14279/depositonce-8408.

[HHPL19] Huffman Hayes, Jane; Payne, Jared; Leppelmeier, Mallory: Toward Improved Artificial Intelligence in Requirements Engineering: Metadata for Tracing Datasets. In: Artificial Intelligence for Requirements Engineering (AIRE). IEEE, pp. 256–262, 2019. https://doi.org/10.1109/REW.2019.00052.

[SV20] Schlutter, Aaron; Vogelsang, Andreas: Trace Link Recovery using Semantic Relation Graphs and Spreading Activation. In: Requirements Engineering. IEEE, pp. 20–31, 2020. https://doi.org/10.1109/RE48521.2020.00015.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 97

Probabilistic Grammar-based Test Generation

Ezekiel Soremekun[1,2] Esteban Pavese[3] Nikolas Havrikov[4] Lars Grunske[5] Andreas Zeller[6]

Abstract: Given a program that has been tested on some sample input(s), what does one test next? To further test the program, one needs to construct inputs that cover (new) input features, in a manner that is different from the initial samples. This talk presents an approach that learns from past test inputs to generate *new but different* inputs.

To achieve this, we present an approach called *inputs from hell* which employs *probabilistic context-free grammars* to learn the distribution of input elements from sample inputs. In this work, we employ probabilistic grammars as input parsers and producers. Applying probabilistic grammars as *input parsers*, we learn the statistical distribution of input features in sample inputs. As a *producer*, probabilistic grammars ensure that generated inputs are syntactically correct by construction, and it controls the distribution of input elements by assigning probabilities to individual production rules. Thus, we create inputs that are dissimilar to the sample by inverting learned probabilities.

In addition, we generate *failure-inducing inputs* by learning from inputs that caused failures in the past, this gives us inputs that share similar features and thus also have a high chance of triggering bugs. This approach is useful for bug reproduction and testing for patch completeness.

Keywords: Grammar; Test Case Generation; Probabilistic Grammars; Input Samples

1 Summary

This article is an abridged version of our paper titled "Inputs from Hell: Learning Input Distributions for Grammar-Based Test Generation" which is published in the proceedings of the IEEE Transactions on Software Engineering (TSE) [So20].

Grammar-based test generation techniques automatically produce thousands of valid inputs for software testing [HZ19]. However, it is also important to test programs on diverse inputs, in order to explore different features. In this work, we address the problem of generating *syntactically valid inputs that are (dis)similar to seen inputs.* Specifically, given a program that has been tested on some sample inputs, we ask the following: Which inputs should one test next? How can one generate inputs that are (dis)similar to the initial samples? To further

[1] SnT, University of Luxembourg, Luxembourg ezekiel.soremekun@uni.lu
[2] This work was done while working at CISPA Helmholtz Center for Information Security, Saarbrücken
[3] Department of Computer Science, Humboldt-Universität zu Berlin pavesees@informatik.hu-berlin.de
[4] CISPA Helmholtz Center for Information Security, Saarbrücken nikolas.havrikov@cispa.de
[5] Department of Computer Science, Humboldt-Universität zu Berlin grunske@informatik.hu-berlin.de
[6] CISPA Helmholtz Center for Information Security, Saarbrücken zeller@cispa.de

doi:10.18420/SE2021_36

test the program, one needs to construct inputs that cover new input features. Hence, the developer is tasked with the generation of *syntactically valid but different inputs*.

To tackle this challenge, we present a probabilistic grammar-based test generation approach called *inputs from hell*. The main idea of this technique is to apply *probabilistic context-free grammars* (PCFG) as input parsers to learn the distribution of input elements from sample inputs, then apply the learned grammar as a producer. Specifically, applying probabilistic grammars as parsers, we learn the frequency of occurrence of production rules in sample inputs. Then, we apply the learned PCFG as a producer to serve two major purposes, (1) it ensures that generated inputs are syntactically correct by construction, and (2) it controls the distribution of input elements by assigning probabilities to individual production rules.

This approach allows for three test generation strategies: 1) `Common inputs` – by generating inputs using the learned probabilistic grammar, we can create inputs that are similar to the sample; this is useful for regression testing. 2) `Uncommon inputs` – inverting the learned probabilities in the grammar yields inputs that are strongly dissimilar to the sample; this is useful for completing a test suite with (different) inputs that test uncommon features, yet are syntactically valid. 3) `Failure-inducing inputs` – learning from inputs that caused failures in the past gives us inputs that share similar features and thus also have a high chance of triggering bugs; this is useful for testing the completeness of fixes.

We examined the effectiveness of our approach using 20 subject programs and three input formats. Our experimental results show that "common inputs" reproduced 96% of the program features (i.e. methods) induced by the samples. In contrast, for almost all subjects (95%), the "uncommon inputs" covered significantly different methods from the samples. By learning from failure-inducing samples, our approach reproduced all failures triggered by the sample inputs and also reveals new failures.

We have presented a technique that applies PCFG to generate (dis)similar test inputs. This approach provides a general and cost-effective means to generate test cases that can then be targeted to the commonly used portions of the software, or to the rarely used features.

Bibliography

[HZ19] Havrikov, Nikolas; Zeller, Andreas: Systematically covering input structure. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 189–199, 2019.

[So20] Soremekun, Ezekiel; Pavese, Esteban; Havrikov, Nikolas; Grunske, Lars; Zeller, Andreas: Inputs from Hell: Learning Input Distributions for Grammar-Based Test Generation. In: IEEE Transactions on Software Engineering. 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 99

Learning to Generate Fault-revealing Test Cases in Metamorphic Testing

Helge Spieker[1], Arnaud Gotlieb[1]

Abstract: Metamorphic Testing is a software testing paradigm which aims at using necessary properties of a system under test, called metamorphic relations (MR), to either check its expected outputs, or to generate new test cases [Se16]. Metamorphic Testing has been successful to test programs for which a full oracle is unavailable or to test programs with uncertainties on expected outputs such as learning systems. In this paper, we formulate the effective selection of MRs as a reinforcement learning problem, based on *contextual bandits*. Our method *Adaptive Metamorphic Testing* sequentially selects a MR that is expected to provide the highest payoff, i.e., that is most likely to reveal faults. Which MRs are likely to reveal faults is learned from successive exploration trials. The bandit explores the available MRs and evaluates the fault landscape of the system under test, thereby providing valuable information to the tester. We present experimental results over two applications in machine learning, namely image classification and object detection, where Adaptive Metamorphic Testing efficiently identifies weaknesses of the tested systems. The original paper "Adaptive Metamorphic Testing with Contextual Bandits" first appeared in the Journal of Systems and Software (2020) [SG20].

Keywords: Software Testing; Metamorphic Testing; Machine Learning; Contextual Bandits

Metamorphic testing (MT) is a testing paradigm, in which a source test case is transformed into a new follow-up test case for which the exact expected outcome is unknown, but a relation between the source and follow-up test case is available [CCY98, Ch18]. MT aims at using necessary properties of a software under test to either check its expected outputs or to generate new test cases. By execution of the follow-up test case it can be confirmed whether the system-under-test behaves according to the so-called metamorphic relation. If the relation is violated, a failure in the system has been identified. Metamorphic testing thereby addresses the oracle problem in software testing, where it is impossible or difficult to know the exact system output for a test case.

Typical examples for successful applications include machine learning models used for classification tasks, for which only stochastic behaviors can be specified [Ba15]. Indeed, these models are often initially trained with existing datasets and then exploited to classify new data samples. However, the expected class of any new data sample is unknown and thus, these samples cannot be used for testing the trained models. Fortunately, transformations over the data samples which do not change their (unknown) class, are usually available. By applying these transformations, i.e. the metamorphic relations (MRs) in MT, it becomes possible to effectively test machine learning models and their training [Mu08, DHG17].

[1] Simula Research Laboratory, Norway {helge,arnaud}@simula.no

doi:10.18420/SE2021_37

This papers addresses the problem of MR selection, i.e. determining which from a set of known MRs are best suited to discover faults in the system under test. We formulate the effective selection of MRs as a reinforcement learning problem, based on *contextual bandits* [LZ07]. Our method *Adaptive Metamorphic Testing* (AMT) defines a *test transformation bandit* which sequentially selects a MR that is expected to provide highest payoff, i.e., is most likely to reveal faults. Which MRs are likely to reveal faults is learned from successive exploration trials. The bandit explores the different MRs and evaluates the fault landscape of the system under test, thereby providing valuable information to the tester.

Learning the selection of MRs can be useful when testing under resource-constraints, for example in cases where the system under test changes are frequently integrated and tested, but also for infrequent testing when the number of MRs is large or their checking is costly. We have applied our method to test deep learning systems for computer vision. Adaptive metamorphic testing showed to find the same failure rate and distribution than exhaustive testing while requiring less test executions, which can be costly or at least time-consuming. At the same time, it's error discovery is stronger than pure random testing, because it can adapt to the strengths of the available metamorphic relations for certain test cases. Our implementation and datasets are available at: `https://github.com/HelgeS/tetraband`

Bibliography

[Ba15] Barr, Earl T.; Harman, Mark; McMinn, Phil; Shahbaz, Muzammil; Yoo, Shin: The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering, 41(5):507–525, 2015.

[CCY98] Chen, T.Y.; Cheung, S.C.; Yiu, S.M.: Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[Ch18] Chen, Tsong Yueh; Kuo, Fei-Ching; Liu, Huai; Poon, Pak-Lok; Towey, Dave; Tse, T. H.; Zhou, Zhi Quan: Metamorphic Testing: A Review of Challenges and Opportunities. ACM Computing Surveys, 51(1), 2018.

[DHG17] Ding, Junhua; Hu, Xin-Hua; Gudivada, Venkat: A Machine Learning Based Framework for Verification and Validation of Massive Scale Image Data. IEEE Transactions on Big Data, 26(3):1–1, 2017.

[LZ07] Langford, John; Zhang, Tong: The Epoch-Greedy Algorithm for Multi-Armed Bandits with Side Information. In: Advances in Neural Information Processing Systems 20 (NIPS 2007). pp. 817–824, 2007.

[Mu08] Murphy, Christian; Kaiser, Gail; Hu, Lifeng; Wu, Leon: Properties of Machine Learning Applications for Use in Metamorphic Testing. Proc. of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 867–872, 2008.

[Se16] Segura, Sergio; Fraser, Gordon; Sanchez, Ana B.; Ruiz-Cortes, Antonio: A Survey on Metamorphic Testing. IEEE Transactions on Software Engineering, 42(9):805–824, 2016.

[SG20] Spieker, Helge; Gotlieb, Arnaud: Adaptive Metamorphic Testing with Contextual Bandits. Journal of Systems and Software, 165:110574, July 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 101

Automated Implementation of Windows-related Security-Configuration Guides

Patrick Stöckle[1] Bernd Grobauer[2] Alexander Pretschner[3]

Abstract: Dieser Vortrag wurde auf der 35. IEEE/ACM International Conference on Automated Software Engineering (ASE) präsentiert. Unsicher konfigurierte Geräte stellen ein großes Sicherheitsproblem dar. Eine Möglichkeit, dieses Problem zu lösen, sind öffentlich verfügbare und standartisierte Sicherheitskonfigurationsrichtlinien. Dieser Ansatz birgt jedoch die Schwierigkeit, dass Administratoren auf Basis der Anleitungen in diesen Richtlinien ihre Systeme manuell sichern müssen. Dieses manuelle Sichern ist teuer und fehleranfällig. In unserem Beitrag präsentieren wir einen Ansatz, mit dem wir Richtlinien für Windows-Systeme automatisiert anwenden können. Dafür wenden wir Techniken der Sprachverarbeitung an. Im ersten Teil unserer Evaluation können wir anhand einer öffentlichen Richtlinie für Windows 10 zeigen, dass unser Ansatz für 83% der Regeln keinerlei menschliche Interaktion benötigt. Im zweiten Teil zeigen wir anhand von 12 öffentlichen Richtlinien mit über 2000 Regeln, dass unser Ansatz die Regeln zu 97% korrekt anwendet. So wird die sichere Konfiguration von Windows-Systemen einfacher und wir hoffen, dass dies zukünftig zu weniger Sicherheitsvorfällen führen wird.

Keywords: Sicherheit; Konfigurationsmanagement; Computerlinguistik

Fehlkonfigurationen verringern die Sicherheit eines Systems, indem sie Schwachstellen einführen, die oft schwer aufzuspüren sind. Administratoren zufolge gibt es einen Hauptfaktor dafür: mangelndes Wissen. [Di18] Eine Möglichkeit, mit diesem Problem umzugehen, ist das Verwenden von bestehenden Sicherheitskonfigurationsrichtlinien. Diese bestehen aus Regeln für ein bestimmtes Softwaresystem wie Windows 10. Jede Regel erklärt, welche Einstellung auf welchen Wert gesetzt werden sollte, um das System sicherer zu machen, und warum wir sie anwenden sollte. Bekannte Herausgeber solcher Richtlinien sind das Center for Internet Security (CIS) oder die Defense Information Systems Agency (DISA).

Die Herausgeber veröffentlichen ihre Richtlinien in Formaten wie PDF und im *Extensible Configuration Checklist Description Format (XCCDF)*, das Teil des *Security Content Automation Protocol (SCAP)* ist. Obwohl XCCDF als maschinenlesbares Format konzipiert ist, sind die Anweisungen zur Implementierung der Sicherheitseinstellungen nur in menschenlesbarer Form enthalten. Die vorhandenen Richtlinien lösen zwar das Problem des mangelnden Wissens, bringen aber eine neue Herausforderung mit sich: Automatische Umsetzungen sind im SCAP-Standard nicht spezifiziert. Die Herausgeber umgehen diese Hürde manchmal, indem sie zusätzliche Artefakte wie Skripte oder Backup-Dateien zur

[1] Technische Universität München (TUM), Boltzmannstr. 3, 85748 Garching b. München patrick.stoeckle@tum.de
[2] Siemens AG bernd.grobaueratsiemens.com
[3] Technische Universität München (TUM), Boltzmannstr. 3, 85748 Garching b. München alexander.pretschner@tum.de

doi:10.18420/SE2021_38

Verfügung stellen. Dies ist auf drei Arten problematisch: Erstens gibt es solche Artefakte nicht für alle Richtlinien. Zweitens werden die Richtlinien häufig aktualisiert und die Artefakte müssen auf Herausgeber-Seite oft und manuell aktualisiert werden. Drittens wird bei eigenständigen Artefakten für die Implementierung das Anpassen (tailoring) von Richtlinien umständlich und fehleranfällig. Eine einfache und leichte Anpassung ist jedoch unerlässlich, da Richtlinien von CIS oder DISA nie ohne Anpassungen in der eigenen Organisation umgesetzt werden.

Unsere für Windows-Betriebssysteme und -Anwendungen realisierte Lösung für dieses Problem besteht aus drei Schritten. Zuerst verarbeiten wir die Dateien, die definieren, welche Einstellungen auf einem Windows-basierten System existieren, und speichern das Ergebnis, um während der Umsetzung darauf zugreifen zu können. Zweitens nutzen wir Techniken der Computerlinguistik, um die Einstellungen und die geforderten Werte aus den Richtlinien zu extrahieren. Wir verwenden die Informationen des ersten Schritts, um zu überprüfen, ob die extrahierte Einstellung existiert und ob der extrahierte Wert eine gültige Eingabe für diese Einstellung ist. So können wir das Risiko falsch extrahierter Werte auf ein Minimum reduzieren. Drittens übersetzen wir die Einstellungen und Werte unter Verwendung der Informationen aus dem ersten Schritt in ihre tatsächliche Umsetzung. Unsere Beiträge sind:

- eine Proof-of-Concept-Implementierung unseres Ansatzes.
- eine Schritt-für-Schritt-Dokumentation unseres Ansatzes unter Verwendung der DISA Windows Server 2016 Richtlinie[4] und eine aktualisierte Version für 2019[5]. Hierbei können wir zeigen, dass unser Ansatz 83% der Regeln ohne manuellen Aufwand umsetzen kann.
- eine Evaluation unseres Ansatzes unter Verwendung bestehender Richtlinien von DISA und CIS mit über 2000 Regeln[6]; unser Ansatz setzt die gegebenen Regeln zu 97% korrekt um.

Durch unseren Ansatz wird die sichere Konfiguration von Windows-Systemen deutlich einfacher. Wir hoffen, dass in Zukunft mehr Administratoren ihre Systeme sicher konfigurieren werden und so das Risiko von Sicherheitsvorfällen sinkt.

Literatur

[Di18] Dietrich, C.; Krombholz, K.; Borgolte, K.; Fiebig, T.: Investigating System Operators' Perspective on Security Misconfigurations. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18, ACM, Toronto, Canada, S. 1272–1289, 2018, ISBN: 978-1-4503-5693-0, URL: http://doi.acm.org/10.1145/3243734.3243794.

[4] swh:1:dir:c3803619f51702199b19405547e2be2f2f55bdd2
[5] swh:1:dir:13ffd9d2566c64afdedd414336a95a35605392d7
[6] swh:1:dir:b5c15f48b2c288f58533c9354bea3703ffbbb0dd

Variability Representations in Class Models: An Empirical Assessment (Summary)

Daniel Strüber[1], Anthony Anjorin[2], Thorsten Berger[3]

Abstract: We present our paper originally published in the proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems 2020 (MODELS).
Owing to the ever-growing need for customization, software systems often exist in many different variants. To avoid the need to maintain many different copies of the same model, developers of modeling languages and tools have recently started to provide representations for such variant-rich systems, notably variability mechanisms that support the implementation of differences between model variants. Available mechanisms either follow the annotative or the compositional paradigm, each of them having unique benefits and drawbacks. Language and tool designers select the used variability mechanism often solely based on intuition. A better empirical understanding of the comprehension of variability mechanisms would help them in improving support for effective modeling. In this paper, we present an empirical assessment of annotative and compositional variability mechanisms for class models. We report and discuss findings from an experiment with 73 participants, in which we studied the impact of the chosen variability mechanisms during model comprehension tasks. We find that, compared to the baseline of listing all model variants separately, the annotative technique did not affect developer performance. Use of the compositional mechanism correlated with impaired performance. For a subset of our tasks the annotative mechanism is preferred to the compositional one and the baseline. We present actionable recommendations concerning support of flexible, tasks-specific solutions, and the transfer of best established best practices from the code domain to models.

Keywords: model-driven engineering; class models; variability; software product lines

1 Summary

Variant-rich systems can offer companies major strategic advantages, such as the ability to deliver tailor-made software products to their customers. Still, when developing a variant-rich system, severe challenges may arise during maintenance, evolution, and analysis, especially when variants are developed in the naive *clone-and-own* approach, that is, by copying and modifying them. As companies begin to streamline their development workflows for building variant-rich systems, they recognize a need for variability management in all key development artifacts, including models. The car industry is particularly outspoken on their need for model-level variability mechanisms.

Recognizing this need, researchers have started building variability mechanisms for models. Variability mechanisms are now available both for UML and DSMLs. Adoption in several

[1] Radboud University, Nijmegen, Netherlands d.strueber@cs.ru.nl
[2] IAV Automotive Engineering, Germany tony@anjorin.de
[3] Chalmers | University of Gothenburg, Sweden thorsten.berger@chalmers.se

doi:10.18420/SE2021_39

industrial DSMLs has demonstrated the general feasibility of model-level variability mechanisms in practice. Still, language and tool designers are offered little guidance on selecting the most effective variability mechanism for their purposes. In fact, there is a lack of evidence to support the preference of one mechanism over the other. Arguably, *comprehensibility* is a decisive factor for the efficiency of a variability mechanism—for any maintenance and evolution activity (e.g. bugfixing, feature implementations), the developers first need to understand the existing system. A better empirical understanding of the comprehension of variability mechanisms could support the development of more effective modeling languages and tools.

To this end, our paper [SAB20] presents an empirical study of variability mechanisms for class models, an ubiquitous modeling language. In a fully randomized experiment performed with 73 participants with relevant background, we studied how the choice of variability mechanism affects performance during model comprehension tasks. We consider two selected variability mechanisms that are representative for two main types: *Annotative* mechanisms maintain an integrated, annotated representation of all variants. They are conceptually simple, but can impair understandability since elements are cluttered with variability information. *Compositional* mechanisms allow to compose a set of smaller sub-models to form a larger model. They are appealing as they establish a clear separation of concerns, but involve a composition step which might be cognitively challenging. We aimed to shed light on the impact of these inherent trade-offs by using an annotative mechanism (model templates [CA05, St18]) and a compositional one (model refinement [An14]).

We present the following results: 1. Compared to working with an explicit enumeration of all variants, the annotative mechanism generally lead to a similar performance (completion times and correctness scores) and subjective difficulty ratings. 2. The compositional mechanism generally lead to worse performance and difficulty ratings. 3. The variability mechanism preferred by most participants depended on the considered task.

Bibliography

[An14] Anjorin, Anthony; Saller, Karsten; Lochau, Malte; Schürr, Andy: Modularizing triple graph grammars using rule refinement. In: FASE. Springer, pp. 340–354, 2014.

[CA05] Czarnecki, Krzysztof; Antkiewicz, Michał: Mapping features to models: A template approach based on superimposed variants. In: GPCE. Springer, pp. 422–437, 2005.

[SAB20] Strüber, Daniel; Anjorin, Anthony; Berger, Thorsten: Variability Representations in Class Models: An Empirical Assessment. In: MODELS. pp. 239–256, 2020.

[St18] Strüber, Daniel; Rubin, Julia; Arendt, Thorsten; Chechik, Marsha; Taentzer, Gabriele; Plöger, Jennifer: Variability-based model transformation: formal foundation and application. FAC, 30(1):133–162, 2018.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 105

Using Key Performance Indicators to Compare Software-Development Processes

Cem Sürücü[1,2] Bianying Song[1] Jacob Krüger[2] Gunter Saake[2] Thomas Leich[3]

Abstract: Extended abstract of our paper published at the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) 2020 [Sü20].

Keywords: Key performance indicators; Quality assurance; Monitoring; Experience report

Large software-developing organizations are often structured around various organizational units that cooperate in software-development processes. While the specialization of such units yields advantages, it can be challenging to keep an overview of complex development processes, to maintain transparency, comparability as well as traceability, and to steer business decisions—especially, when the units use varying metrics to measure and monitor a specific part of the whole development process. To tackle such challenges, key performance indicators are measured to compare an organization's performance with respect to specified objectives throughout whole development processes. Still, the structure of large organizations and the corresponding challenges also pose difficulties while introducing key performance indicators.

In our paper, we describe an experience report on establishing key performance indicators at Volkswagen Financial Services AG (VWFS), a large international organization in the finance sector with over 16 000 employees. We report how we introduced and use key performance indicators at VWFS to facilitate end-to-end analyses of software-development processes, pointing out their value, challenges we faced, and recommendations for other organizations. For this purpose, we present our light-weight, technology-independent methodology that allowed us to receive fast feedback from our stakeholders. While applying this methodology, we customized one existing, defined three new, and built on seven established key performance indicators to address the requirements of VWFS. To define the scope of our methodology and the key performance indicators we used, we closely collaborated with our stakeholders to define six criteria we aimed to improve: transparency, intelligibility, coverage, quantification, comparability, and communication. For each criterion, we report the impact we experienced from establishing the key performance indicators, and

[1] Volkswagen Financial Services AG
Cem.Sueruecue@vwfs.com
Bianying.Song@vwfs.com
[2] Otto-von-Guericke University Magdeburg
jkrueger@ovgu.de
saake@ovgu.de
[3] Harz University of Applied Sciences
tleich@hs-harz.de

doi:10.18420/SE2021_40

demonstrate five examples for concrete improvements we derived. Finally, we share six lessons learned to help other organizations and practitioners: (1) choose proper tooling and understand the limitations of using off-the-shelf solutions; (2) define and customize the key performance indicators they need to measure; (3) to get feedback from stakeholders, communicate benefits, and, thereby, facilitate acceptance; (4) understand why and how to incorporate stakeholders to create benefits for different target groups; (5) evaluate the value and usability of their key performance indicators; and (6) to be aware of, as well as control, costs and benefits of key performance indicators in a business case. By establishing our defined set of key performance indicators, we successfully improved with respect to the criteria defined. The success of these improvements has been underpinned by our ability to better compare the quality of software releases as well as affirmative feedback from users, managers, and other stakeholders.

Building on our experiences, we recently intensified our use of key performance indicators to compare different software-development processes, methods, and technologies during a shift to a more agile paradigm—aiming to measure and assess the benefits of this transformation. In this context, we have transformed the teamwork for developing special-business software from the predominant distributed, interacting organizational units to multidisciplinary, cross-functional agile teams. Up until now, we have measured and reported our defined set of key performance indicators across all relevant organizational units for more than one year, covering multiple software releases that provide a reliable data basis for comparing conventional waterfall and mixed-method development to the newly established agile paradigm. We focus particularly on comparing timely requirements, test progress, as well as software quality before (i.e., defect analysis of internal deliverables) and after (i.e., incident analysis and delivery speed of external deliverables on productive systems) releases. By investigating our data on the agile transformation, we conclude that we have to measure additional properties that help us capture agile-specific practices and values, such as multidisciplinary, cross-functional teamwork with fast feedback. For this purpose, we are currently analyzing what properties we need to measure, how to translate these into key performance indicators, and how to make them comparable between different software-development processes. Consequently, we argue that valuable future work is to study what key performance indicators we established are relevant for agile processes, how to adjust and introduce new key performance indicators for agile processes, how to ensure comparability, and what differences we experience between measuring as well as using different software-development processes, methods, and technologies.

Bibliography

[Sü20] Sürücü, Cem; Song, Bianying; Krüger, Jacob; Saake, Gunter; Leich, Thomas: Establishing Key Performance Indicators for Measuring Software-Development Processes at a Large Organization. In: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE. ACM, 2020.

A Longitudinal Study of Static Analysis Warning Evolution and the Effects of PMD on Software Quality in Apache Open Source Projects (Summary)

Alexander Trautsch[1] Steffen Herbold[2] Jens Grabowski[3]

Abstract: This article summarizes our work originally published in the journal Empirical Software Engineering [THG20].

Keywords: Static code analysis; Quality evolution; Software metrics; Software quality

Software engineering best practices have included the use of static analysis tools for years. These tools can help developers spot common coding mistakes and maintainability problems. Static analysis tools work by analyzing source code or byte code and perform pattern matching to find problematic lines of code. While they are seen by developers as quality improving there are also problems with false positives.

While some studies are investigating static analysis tools, none were focused on the evolution of warnings over the complete development history for a general purpose static analysis tool. In our study we use PMD as the static analysis tool. It contains a broad set of warnings, works directly on source code and has been under development for many years. Therefore, it is able to provide us with a comprehensive history of static analysis warnings in our study subjects.

We investigate 54 open source Java projects under the umbrella of the Apache Software Foundation. We collect up to 17 years of development history of our study subjects and plot the evolution and trends of static analysis warnings. Overall, we collect static analysis warnings and the number of logical lines of code for 112,266 commits of our study subjects. We also collect all reported bugs and complete build information including information from Maven Central for all study subjects. This data collection is facilitated by SmartSHARK [Tr17, Tr20] and a local HPC system.

As we do not want to rely on a heuristic to find removed warnings we include all warnings in every commit and plot the warning density, i.e., the sum of all warnings divided by the number of logical lines of code. To further restrict noise we only investigate production

[1] Georg-August-Universität Göttingen, Institut für Informatik, Goldschmidtstrasse 7, 37077 Göttingen, Deutschland alexander.trautsch@cs.uni-goettingen.de
[2] Karlsruher Institut für Technologie, AIFB, Kaiserstr. 89, 76133 Karlsruhe, Deutschland steffen.herbold@kit.edu
[3] Georg-August-Universität Göttingen, Institut für Informatik, Goldschmidtstrasse 7, 37077 Göttingen, Deutschland grabowski@cs.uni-goettingen.de

doi:10.18420/SE2021_41

code, excluding tests, documentation and example code. As we are interested in trends we restrict the commit graph of our study subjects to a single path to remove noise due to release branches.

Our study explores two main research questions. How are static analysis warnings evolving over time and what is the impact of using PMD. We want to know if "code gets better", i.e., are static analysis warnings removed over the observed development history. We find that while the sum of warnings usually increases the warning density decreases in most projects. The types of warnings that drive this positive trend are mostly related to coding best practices, e.g., naming, brace and design warnings. On average, every study subject removes 3.5 warnings per 1000 logical lines of code per year.

Using PMD as indicated in the build process of the study subjects has a positive impact on the number of warnings, however this is not the case in all of our study subjects. We find that the use of a specialized configuration of rules for PMD has a negligible impact on the removal of warnings. If we calculate trends of warning density we find that the instances where PMD was used are not statistically significantly better than those without PMD. However, if we use the sum of static analysis warnings there is a statistically significant, albeit small difference.

When we use defect density as a proxy metric for external software quality we find that years in which PMD is part of the build process perform slightly better. However, this part of the study is limited by the available data and will be investigated in more detail in a follow-up study.

Bibliography

[THG20] Trautsch, Alexander; Herbold, Steffen; Grabowski, Jens: A Longitudinal Study of Static Analysis Warning Evolution and the Effects of PMD on Software Quality in Apache Open Source Projects. Empirical Software Engineering, 2020.

[Tr17] Trautsch, Fabian; Herbold, Steffen; Makedonski, Philip; Grabowski, Jens: Addressing problems with replicability and validity of repository mining studies through a smart data platform. Empirical Software Engineering, August 2017.

[Tr20] Trautsch, Alexander; Trautsch, Fabian; Herbold, Steffen; Ledel, Benjamin; Grabowski, Jens: The SmartSHARK Ecosystem for Software Repository Mining. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. ICSE '20, Association for Computing Machinery, New York, NY, USA, p. 25–28, 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 109

Are Unit and Integration Test Definitions Still Valid for Modern Java Projects? An Empirical Study on Open-Source Projects

Fabian Trautsch[1] Steffen Herbold[2] Jens Grabowski[3]

Abstract: We summarize the article *Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects* [THG20], which was published in the Journal of Systems and Software in 2020.

Keywords: Software testing; Unit testing; Integration testing; Empirical software engineering

1 Overview

The article "Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects" published in the Journal of Systems and Software in 2020 presents the results of our investigations of the defect detection capability of unit and integration tests [THG20]. While the software development context evolved over time, the definitions of unit and integration tests remained unchanged. There is no empirical evidence, if these commonly used definitions still fit to modern software development. We evaluate if the existing standard definitions of unit and integration tests are still valid in modern software development context through the analysis of the defect types that are detected, because there should be differences according to the standard literature. We classify test cases according to the definition of the IEEE and use mutation testing to assess their defect detection capabilities.

2 Results

We analyzed 9356 unit tests and 29461 integration tests and could not find any evidence that one test type is more capable of detecting certain defect types than the other one. This implies that we need to reconsider the definitions of unit and integration tests and suggest that the current property-based definitions may be exchanged with usage-based definitions.

[1] Georg-August-Universität Göttingen, Institute für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland fabian.trautsch@cs.uni-goettingen.de
[2] Karlsruher Institut für Technologie, Institute AIFB, Kaiserstr. 89, 76133 Karlsruhe, Deutschland steffen.herbold@kit.edu
[3] Georg-August-Universität Göttingen, Institute für Informatik, Goldschmidtstr. 7, 37077 Göttingen, Deutschland jens.grabowski@cs.uni-goettingen.de

doi:10.18420/SE2021_42

Bibliography

[THG20] Trautsch, Fabian; Herbold, Steffen; Grabowski, Jens: Are unit and integration test definitions still valid for modern Java projects? An empirical study on open-source projects. Journal of Systems and Software, 159:110421, 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 111

Automated Large-scale Multi-language Dynamic Program Analysis in the Wild

Alex Villazón[1], Haiyang Sun[2], Andrea Rosà[3], Eduardo Rosales[4], Daniele Bonetta[5], Isabella Defilippis[6], Sergio Oporto[7], Walter Binder[8]

Abstract: Our paper published in the proceedings of the 33rd European Conference on Object-Oriented Programming (ECOOP 2019) [Vi19a] proposes NAB, a novel framework to execute custom dynamic analysis on open-source software hosted in public repositories. The publication is complemented by an accepted artifact [Vi19b]. Analyzing today's large code repositories has become an important research area for understanding and improving different aspects of modern software systems. Despite the presence of a large body of work on mining code repositories through static analysis, studies applying dynamic analysis to open-source projects are scarce and of limited scale. Nonetheless, being able to apply dynamic analysis to the projects hosted in public code repositories is fundamental for large-scale studies on the runtime behavior of applications, which can greatly benefit the programming-language and software-engineering communities. NAB is fully automatic, language-agnostic, and scalable. We describe NAB's key features and architecture. We also present three case studies on more than 56K Node.js, Java, and Scala projects, enabling us to 1) understand how developers use JavaScript Promises, 2) identify bad coding practices in JavaScript applications, and 3) locate task-parallel Java and Scala workloads suitable for inclusion in a domain-specific benchmark suite. A preliminary version of NAB is available at `http://dag.inf.usi.ch/software/nab/`

Keywords: Dynamic program analysis; code repositories; GitHub; Node.js; Java; Scala; promises; JIT-unfriendly code; task granularity

Bibliography

[Vi19a] Villazón, Alex; Sun, Haiyang; Rosà, Andrea; Rosales, Eduardo; Bonetta, Daniele; Defilippis, Isabella; Oporto, Sergio; Binder, Walter: Automated Large-scale Multi-language Dynamic Program Analysis in the Wild. In: Proceedings of the 33rd European Conference on Object-Oriented Programming (ECOOP). pp. 20:1–20:27, 2019.

[Vi19b] Villazón, Alex; Sun, Haiyang; Rosà, Andrea; Rosales, Eduardo; Bonetta, Daniele; Defilippis, Isabella; Oporto, Sergio; Binder, Walter: Automated Large-Scale Multi-Language Dynamic Program Analysis in the Wild (Artifact). Dagstuhl Artifacts Series, 5(2):11:1–11:3, 2019.

[1] Universidad Privada Boliviana, Bolivia avillazon@upb.edu
[2] Università della Svizzera italiana, Switzerland haiyang.sun@usi.ch
[3] Università della Svizzera italiana, Switzerland andrea.rosa@usi.ch
[4] Università della Svizzera italiana, Switzerland rosale@usi.ch
[5] Oracle Labs, United States daniele.bonetta@oracle.com
[6] Universidad Privada Boliviana, Bolivia isabelladefilippis@upb.edu
[7] Universidad Privada Boliviana, Bolivia sergiooporto@upb.edu
[8] Università della Svizzera italiana, Switzerland walter.binder@usi.ch

doi:10.18420/SE2021_43

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 113

Views on Quality Requirements in Academia and Practice: Commonalities, Differences, and Context-Dependent Grey Areas

Andreas Vogelsang [1], Jonas Eckhardt [2], Daniel Mendez [3], Moritz Berger [4]

Abstract: This article originally appeared in Information and Software Technology (IST) [Vo20]. **Context:** Quality requirements (QRs) are a topic of constant discussions both in industry and academia. While many academic endeavors contribute to the body of knowledge about QRs, practitioners may have different views. **Objective:** We report on a study to better understand the extent to which available research statements on QRs from academic publications, are reflected in the perception of practitioners. Our goal is to analyze differences, commonalities, and context-dependent grey areas in the views of academics and practitioners. **Method:** We conducted a survey with 109 practitioners to assess their agreement with the selected research statements about QRs. Based on a statistical model, we evaluate the impact of a set of context factors to the perception of research statements. **Results:** Our results show that a majority of the statements is well respected by practitioners; however, not all of them. When examining the different groups of respondents, we noticed deviations of perceptions that lead to new research questions. **Conclusions:** Our results help identifying context-dependent differences about how academics and practitioners view QRs and statements where further research is useful.

Keywords: quality requirements; non-functional requirements; context factors; requirements engineering; survey; empirical study

We want to better understand the extent to which available research statements on quality requirements [Gl07] are consistent with the perceptions held by practitioners. In particular, we aim at understanding the extent to which the views and perceptions held by practitioners are corroborated by those of academics. More precisely, we want to understand how well research statements frequently referred to in academic works are perceived by practitioners in their respective context. Questions we opt for answering are: (1) What is the agreement of practitioners with existing research statements about QRs? (2) Which context factors (e.g., industrial sector, company size, experience) influence the agreement of practitioners with research statements about QRs? (3) Can we assign a specific perception of QRs to stereotypical groups of practitioners?

Our hope is that an increased understanding of the practitioners' beliefs and views helps us identifying differences, commonalities, and context-dependent grey areas and pinpoint to existing (and regularly cited) statements where further context-dependent research would

[1] University of Cologne, Software & Systems Engineering, Cologne, Germany vogelsang@cs.uni-koeln.de
[2] Tableau Software, Munich, Germany, jonaseckhardt@googlemail.com
[3] Blekinge Institute of Technology and fortiss GmbH, Blekinge, Sweden and Munich, Germany, daniel.mendez@bth.se
[4] Universität Bonn, Bonn, Germany, moritz.berger@imbie.uni-bonn.de

doi:10.18420/SE2021_44

be useful. The paper makes the following contributions: (1) We define a set of 21 research statements about quality requirements from a total of 17 exemplary and commonly cited research papers from the RE research community. (2) We survey practitioners from several application domains and business contexts regarding their agreement with the previously identified statements about quality requirements. The survey results suggest that practitioners hold strong, and diverse opinions, and that some results inspire more passion and dissension than others. (3) We provide a statistical model that allows evaluating the impact of specific context factors on the perception of research statements. The results of the evaluation show that the perception of some research statements is homogeneous across different development contexts while the perception of others strongly depends on the context. (4) We provide a detailed discussion of the results and contrast them with the original studies from which the statements emerged.

Our intention is not to criticize selected academic manuscripts but to increase our understanding on (1) how much practitioners' views differ with respect to their daily working context, and (2) what we, the research community, can learn from it. Our vision is to contribute to reducing the gap between industrial practice and problems, and academic contributions and solution proposals.

In the past, we have conducted a number of studies in which we investigated the perception [EVM16b] and use [EVM16a, EMV15] of quality requirements by practitioners. The research questions, results, and the underlying data presented in this article are original in the sense that they have not been addressed in a previous analysis and consequently in a publication. The only commonality between the study at hand and one of our previous publications [EVM16b] is that the data underlying these studies have been collected using the same questionnaire (but different parts of it).

Bibliography

[EMV15] Eckhardt, Jonas; Mendéz Fernández, Daniel; Vogelsang, Andreas: How to specify Non-functional Requirements to support seamless modeling? A Study Design and Preliminary Results. In: 9th International Symposium on Empirical Software Engineering and Measurement (ESEM). 2015.

[EVM16a] Eckhardt, Jonas; Vogelsang, Andreas; Mendéz Fernández, Daniel: Are Non-functional Requirements Really Non-functional? An Investigation of Non-functional Requirements in Practice. In: 38th International Conference on Software Engineering (ICSE). 2016.

[EVM16b] Eckhardt, Jonas; Vogelsang, Andreas; Mendéz Fernández, Daniel: On the Distinction of Functional and Quality Requirements in Practice. In: 17th International Conference on Product-Focused Software Process Improvement (PROFES). 2016.

[Gl07] Glinz, Martin: On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference (RE). 2007.

[Vo20] Vogelsang, Andreas; Eckhardt, Jonas; Mendez, Daniel; Berger, Moritz: Views on quality requirements in academia and practice: commonalities, differences, and context-dependent grey areas. Information and Software Technology (IST), 121, 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 115

Status Quo in Requirements Engineering: A Theory and a Global Family of Surveys

Stefan Wagner[1], Daniel Méndez Fernández[2], Michael Felderer[3], Antonio Vetrò[4], Marcos Kalinowski[5], Roel Wieringa[6], Dietmar Pfahl[7], Tayana Conte[8], Marie-Therese Christiansson[9], Desmond Greer[10], Casper Lassenius[11], Tomi Männistö[12], Maleknaz Nayebi[13], Markku Oivo[14], Birgit Penzenstadler[15], Rafael Prikladnicki[16], Guenther Ruhe[17], André Schekelmann[18], Sagar Sen[19], Rodrigo Spínola[20], Ahmed Tuzcu[21], Jose Luis de la Vara[22], Dietmar Winkler[23]

Abstract: While researchers have been investigating the Requirements Engineering (RE) discipline with a plethora of empirical studies, attempts to systematically derive an empirical theory in context of the RE discipline have just recently been started. We aim at providing an empirical and externally valid foundation for a theory of RE practice, which helps software engineers establish effective and efficient RE processes in a problem-driven manner. We designed a survey instrument and an engineer-focused theory that has been conducted in 10 countries. We have a theory in the form of a set of propositions inferred from our experiences and available studies, as well as the results from our pilot study in Germany. We evaluate the propositions with bootstrapped confidence intervals and derive potential explanations for the propositions.

Keywords: Requirements Engineering; Survey; Theory

[1] University of Stuttgart, Stuttgart, Germany, stefan.wagner@iste.uni-stuttgart.de
[2] BTH, Karlskrona, Sweden, daniel.mendez@bth.se
[3] University of Innsbruck, Innsbruck, Austria, michael.felderer@uibk.ac.at
[4] Pollitecnico di Torino, Torino, Italy, antonio.vetro@polito.it
[5] Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil, kalinowski@inf.puc-rio.br
[6] University of Twente, Enschede, The Netherlands, r.j.wieringa@utwente.nl
[7] University of Tartu, Tartu, Estonia, dietmar.pfahl@ut.ee
[8] Universidade Federal do Amazonas, Manaus, Brazil, tayanaconte@gmail.com
[9] Karlstad University, Karlstad, Sweden, marie-therese.christiansson@kau.se
[10] Queen's University Belfast, Belfast, UK, des.greer@qub.ac.uk
[11] Aalto University, Espoo, Finland, casper.lassenius@aalto.fi
[12] University of Helsinki, Helsinki, Finland, tomi.mannisto@helsinki.fi
[13] York University, Toronto, Canada, mnayebi@yorku.ca
[14] University of Oulu, Oulu, Finland, markku.oivo@oulu.fi
[15] Chalmers University, Gothenburg, Sweden, birgitp@chalmers.se
[16] Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil, rafael.prikladnicki@gmail.com
[17] University of Calgary, Calgary, Canada, ruhe@ucalgary.ca
[18] Niederrhein University of Applied Sciences, Krefeld, Germany, andre.schekelmann@hs-niederrhein.de
[19] Simula, Fornebu, Norway, sagar@simula.no
[20] Salvador University - UNIFACS, Salvador, Brazil, rodrigoospinola@gmail.com
[21] zeb.rolfes.schierenbeck.associates GmbH, Munich, Germany, atuzcu@zeb.de
[22] Carlos III University of Madrid, Madrid, Spain, jvara@inf.uc3m.es
[23] Technische Universität Wien, Vienna, Austria, dietmar.winkler@tuwien.ac.at

doi:10.18420/SE2021_45

1 Summary

This talk reports on the second run of the *Naming the Pain in Requirements Engineering* (NaPiRE) initiative that has the goal to characterise requirements engineering practice and problems and was published in the *ACM Transactions on Software Engineering and Methodology* in 2019 [Wa19].

An empirical theory of requirements engineering (RE) is needed if we are to define and motivate guidance in performing high quality RE research and practice. We aim at providing an empirical and externally valid foundation for a theory of RE practice, which helps software engineers establish effective and efficient RE processes in a problem-driven manner. We designed a survey instrument and an engineer-focused theory that was first piloted in Germany and, after making substantial modifications, has been replicated in 10 countries. We have a theory in the form of a set of propositions inferred from our experiences and available studies, as well as the results from our pilot study in Germany. We evaluate the propositions with bootstrapped confidence intervals and derive potential explanations for the propositions.

In this article, we report on the design of the family of surveys, its underlying theory, and the full results obtained from the replication studies conducted in 10 countries with participants from 228 organisations. Our results represent a substantial step forward towards developing an empirical theory of RE practice. The results reveal, for example, that there are no strong differences between organisations in different countries and regions, that interviews, facilitated meetings and prototyping are the most used elicitation techniques, that requirements are often documented textually, that traces between requirements and code or design documents are common, that requirements specifications themselves are rarely changed and that requirements engineering (process) improvement endeavours are mostly internally driven. Our study establishes a theory that can be used as starting point for many further studies for more detailed investigations and complements the theory we established on problems, their causes and effects in requirements engineering practice [Mé17]. Practitioners can use the results as theory-supported guidance on selecting suitable RE methods and techniques.

Bibliography

[Mé17] Méndez Fernández, Daniel; Wagner, Stefan; Kalinowski, Marcos; Felderer, Michael; Mafra, Priscilla; Vetrò, Antonio; Conte, Tayana; Christiansson, Marie-Therese; Greer, Desmond; Lassenius, Casper et al.: Naming the pain in requirements engineering. Empirical Software Engineering, 22(5):2298–2338, 2017.

[Wa19] Wagner, Stefan; Méndez Fernández, Daniel; Felderer, Michael; Vetrò, Antonio; Kalinowski, Marcos; Wieringa, Roel; Pfahl, Dietmar; Conte, Tayana; Christiansson, Marie-Therese; Greer, Desmond et al.: Status quo in requirements engineering: A theory and a global family of surveys. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(2):1–48, 2019.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 117

Programming in Natural Language with *fu*SE: Synthesizing Methods from Spoken Utterances Using Deep Natural Language Understanding

Sebastian Weigelt[1] Vanessa Steurer[2] Tobias Hey[1] Walter F. Tichy[1]

Abstract: With *fu*SE laypeople can create simple programs: one can teach intelligent systems new functions using plain English. *fu*SE uses deep learning to synthesize source code: it creates method signatures (for newly learned functions) and generates API calls (to form the body). In an evaluation on an unseen dataset *fu*SE synthesized 84.6% of the signatures and 66.9% of the API calls correctly[3].

Keywords: Programming in Natural Language; End-User Programming; Deep Learning; AI; NLP

Introduction: Intelligent systems became rather smart lately. One easily arranges appointments by talking to a virtual assistant or controls a smart home through a conversational interface. For the time being, users can only access built-in functionality. However, they will soon expect to add new functionality themselves. For humans, the most natural way to communicate is by natural language. Thus, future intelligent systems must be programmable in everyday language. We propose to apply deep natural language understanding to the task of synthesizing methods from spoken utterances. *fu*SE combines deep learning techniques with information retrieval and knowledge-based methods to grasp the user's intent.

Approach: *fu*SE is a system for programming in (spoken) natural language: laypersons can create method definitions by using natural language only. To investigate how laypersons teach new functionality we ran a preliminary study in which subjects were supposed to teach new skills to a humanoid robot. The study consists of four scenarios in which a humanoid robot should be taught a new skill: greeting someone, preparing coffee, serving drinks, and setting a table for two. We used the online micro-tasking platform *Prolific*[4] and were able to gather 3168 descriptions from 870 participants. Based on the findings of the preliminary study we develop the following three-tiered approach (see Figure 1). First, *fu*SE classifies teaching efforts, i.e. it determines whether an utterance comprises an explicitly stated teaching intent or not. Second, it classifies the semantic structure, i.e. *fu*SE analyzes (and labels) the semantic parts of a teaching sequence. Teaching sequences are composed of a *declarative* and a *specifying* part as well as superfluous information. Third, *fu*SE synthesizes methods, i.e. it builds a model that represents the structure of methods from syntactic

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany, {weigelt|hey|tichy}@kit.edu
[2] inovex GmbH, Karlsruhe, Germany, vsteurer@inovex.de
[3] This contribution is a short version of the paper originally published in the proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (2020) [We20].
[4] Prolific: `https://www.prolific.co/`

doi:10.18420/SE2021_46

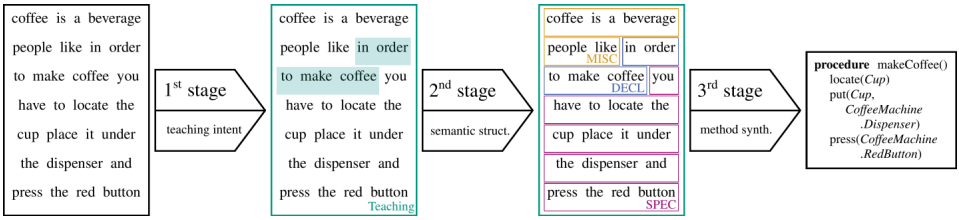


Fig. 1: Schematic overview of *fu*SE's three-tiered approach.

information and classification results. Then, it maps the actions of the specifying part to API calls, injects control structures to form the body and creates the method signature. The first two stages are classification problems. For the first we compared classical machine learning techniques, such as logistic regression and support vector machines, with neural network approaches including the pre-trained language model *BERT* [De19]. For the second task we narrow down to neural networks and *BERT*. However, for both tasks *BERT*-based models performed best: test set accuracy 97.7% (1st stage) resp. 97.3% (2nd stage). The third stage is a combination of syntactic analysis, knowledge-based techniques and information retrieval. We use semantic role labeling, coreference analysis, and a context model to build a semantic model. Afterwards, we synthesize method signatures heuristically and map instructions from the body to API calls using ontology search methods and datatype analysis. To cope with spontaneous (spoken) language, our approach relies on shallow NLP techniques only.

Evaluation: To measure the performance of *fu*SE on unseen data, we set up a case study. We created two new scenarios and used *Prolific* to collect 202 descriptions, of which we randomly drew 100; 78 of these comprise a teaching intent. In sum, the descriptions require the generation of 473 API calls. *fu*SE synthesized 73 method signatures; five were missed due to an incorrect first-stage classification. Out of 73 signatures we assessed only seven to be inappropriate. The generation of API calls (that form the method bodies) also performs well (F_1: 66.9%). These results are promising; however, we plan to improve *fu*SE with a dialog module to query the user in case of ambiguities.

Bibliography

[De19] Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186, June 2019.

[We20] Weigelt, Sebastian; Steurer, Vanessa; Hey, Tobias; Tichy, Walter F.: Programming in Natural Language with fuSE: Synthesizing Methods from Spoken Utterances Using Deep Natural Language Understanding. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, pp. 4280–4295, July 2020.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 119

Data-driven Risk Management for Requirements Engineering: An Automated Approach based on Bayesian Networks

Florian Wiesweg [1], Andreas Vogelsang [2], Daniel Mendez [3]

Abstract: This paper has been accepted at the 2020 IEEE Requirements Engineering Conference (RE) [WVM20]. RE is a means to reduce the risk of delivering a product that does not fulfill the stakeholders' needs. Therefore, a major challenge in RE is to decide how much RE is needed and what RE methods to apply. The quality of such decisions is strongly based on the RE expert's experience and expertise in carefully analyzing the context and current state of a project. Recent work, however, shows that lack of experience and qualification are common causes for problems in RE. We trained a series of Bayesian Networks on data from the NaPiRE survey to model relationships between RE problems, their causes, and effects in projects with different contextual characteristics. These models were used to conduct (1) a post-mortem (diagnostic) analysis, deriving probable causes of sub-optimal RE performance, and (2) to conduct a preventive analysis, predicting probable issues a young project might encounter. The method was subject to a rigorous cross-validation procedure for both use cases before assessing its applicability to real-world scenarios with a case study.

Keywords: Requirements Engineering; Data-Driven RE; Risk Management

The purpose of Requirements Engineering (RE) is to elicit, document, analyze, and manage requirements to minimize the risk of delivering a system that does not meet the stakeholders' desires and needs. Over the last 30 years, a number of methods, processes, tools, and best practices have been proposed to support this goal. However, there is no silver-bullet method or process that fits every project. In fact, a large part of the job of a requirements engineer in practice is to observe and analyze the context and current state of a project carefully and decide how much and what kind of RE is beneficial. As already addressed in the above-mentioned definition of RE, this decision is often a matter of controlling risks. Conducting RE tasks always comes with costs that ideally pay off in the sense that they lower a particular risk for a project [FV19]. Making such decisions demands social and technical skills but also a lot of experience. Recent studies have shown that lack of experience and lack of qualification of RE team members are the second and third most common causes for problems in RE (lack of time being the top cause) [MF17]. As a result, a number of projects fail either because of too little RE leading to stakeholder dissatisfaction or too much RE leading to high costs and developer frustration.

[1] Technische Universität Berlin, Berlin, Germany, florian.wiesweg@alumni.tu-berlin.de
[2] University of Cologne, Software & Systems Engineering, Cologne, Germany vogelsang@cs.uni-koeln.de
[3] Blekinge Institute of Technology and fortiss GmbH, Blekinge, Sweden and Munich, Germany, daniel.mendez@bth.se

doi:10.18420/SE2021_47

In this paper, we propose a data-driven approach to risk management in RE. Our goal is to predict RE problems, their causes, and effects for a given project. Bayesian Networks can be used to characterize such dependencies quantitatively by conditional probabilities and update the probability of certain phenomena when other phenomena are observed. Therefore, we evaluated different versions of Bayesian Networks that model the relations between causes, problems, and effects in RE. We trained the models on data that was collected through two surveys with answers from 228 and 488 practitioners, respectively, about problems, causes, and effects encountered in real projects. These surveys also provide data on the context of the projects. We use the trained models for the following two use cases: (1) **Post-Mortem Analysis:** Given a set of problems and effects observed in a failing or failed project, the approach diagnoses the most likely causes leading to these issues. (2) **Preventive Analysis:** Given a set of causes and effects observed in a new or running project, the approach predicts the most likely problems to be faced.

We performed two types of evaluations for our approach. Firstly, we performed cross-validation to compare the predictive power of different models. We achieved the best results for both use cases with surprisingly simple models, which ignore the causal structure implied by the original survey but include a set of context factors. For varying probability thresholds $t \in \{0.3, 0.5, 0.7\}$, the best diagnostic model achieves recalls of 0.6, 0.48, 0.44 and precisions of 0.76, 0.92, 0.99, respectively. The best predictive model achieves recalls of 0.84, 0.69, 0.59 and precisions of 0.71, 0.89, 0.99. A ranking-based output of the top-5 predictions results in a recall of 0.81 and a precision of 0.38 for the best diagnostic model and a recall of 0.73 and a precision of 0.71 for the best predictive model. Secondly, we conducted a case study in industry to evaluate the external validity of the approach. We compared and discussed the predictions of the tool with the expectations of an RE expert for the diagnostic reasoning use case. Furthermore, we elicited feedback regarding the importance of recall vs. precision for the problem and how the tool should be tailored in detail to support practitioners best. In a nutshell, the case study showed that the method achieves good congruence between its predictions and the results expected by the expert, but requires additional tuning towards high precision. We conclude that such data-driven approaches are very likely to be practical and advantageous, but that the remaining potentials in the underlying data and the user interface should be realized first.

Bibliography

[FV19] Femmer, H.; Vogelsang, A.: Requirements Quality Is Quality in Use. IEEE Software, 36(3):83–91, 2019.

[MF17] Méndez Fernández, Daniel *et al.*: Naming the Pain in Requirements Engineering. Empirical Software Engineering, 22(5):2298–2338, 2017.

[WVM20] Wiesweg, F.; Vogelsang, A.; Mendez, D.: Data-driven Risk Management for Requirements Engineering: An Automated Approach based on Bayesian Networks. In: 28th IEEE International Requirements Engineering Conference (RE). 2020.

Explaining Pair Programming Session Dynamics from Knowledge Gaps

Franz Zieris[1], Lutz Prechelt[2]

This is an extended abstract of the paper with the same title [ZP20a] which was presented at the 42nd International Conference on Software Engineering (2020).

Keywords: pair programming; qualitative analysis; grounded theory methodology

1 Background, Data, and Research Method

Pair programming (PP) has many purported benefits, including higher code quality, faster progress, and knowledge transfer between developers. Despite a lot of research on the effectiveness of PP, the question when it is useful or less useful remains unsettled: A meta-analysis found mere tendencies and a lot of between-study variance [Ha09]; a large controlled experiment could not determine consistent moderating effects of task complexity and developer expertise [Ar07]. Even though the feasible experimental setups tend to be highly unrealistic, there have been only few qualitative studies which looked at the actual PP process in industrial contexts (e. g., [Pl15]).

We follow Straussian Grounded Theory Methodology [SC90] to understand how pair programmers actually transfer knowledge. We analyze 26 recordings of industrial PP sessions from 9 companies which we selected in the manner of *theoretical sampling* from the *PP-ind* session repository [ZP20b]. For *open coding*, we build on our own prior work [ZP14; ZP16] that identified various phenomena related to within-session knowledge build-up and transfer. We validate our findings with practitioners from four companies.

2 Results

We identify two different types of required knowledge and explain how different constellations of knowledge gaps in these two respects lead to different session dynamics:

- Industrial pairs mostly deal with gaps in project-specific *system understanding,* or S knowledge. They address any differences in their respective system understanding first before building up new system understanding together.

[1] Freie Universität Berlin, Institut für Informatik, Takustr. 9, 14195 Berlin, Deutschland zieris@inf.fu-berlin.de
[2] Freie Universität Berlin, Institut für Informatik, Takustr. 9, 14195 Berlin, Deutschland prechelt@inf.fu-berlin.de

doi:10.18420/SE2021_48

- Differences in *general software development knowledge*, or G knowledge, hardly hamper the PP process. Rather, such a difference is an opportunity to transfer knowledge—which pairs only do after they dealt with their S knowledge gaps.
- Building up lacking G knowledge together in a PP session appears to be difficult.
- Pair constellations with *complementary knowledge* allow both partners to contribute S knowledge and G knowledge, respectively, which makes pair programming a particularly effective practice.
- Software developers may use our findings when forming pairs (e. g., by choosing a partner and/or amending the goal of the session such that differences in their knowledge levels play out favorably) or as a means of reflecting after a session (e. g., whether the *right* knowledge gaps were addressed or which were newly identified).

References

[Ar07] Arisholm, E.; Gallis, H.; Dybå, T.; Sjøberg, D. I.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. IEEE Transactions on Software Engineering 33/2, pp. 65–86, 2007.

[Ha09] Hannay, J. E.; Dybå, T.; Arisholm, E.; Sjøberg, D. I.: The effectiveness of pair programming: A meta-analysis. Information and Software Technology 51/7, pp. 1110–1122, 2009.

[Pl15] Plonka, L.; Sharp, H.; van der Linden, J.; Dittrich, Y.: Knowledge transfer in pair programming: An in-depth analysis. International Journal of Human-Computer Studies 73/, pp. 66–78, 2015.

[SC90] Strauss, A.; Corbin, J.: Basics of Qualitative Research. Grounded Theory Procedure and Techniques. Sage Publications, 1990, ISBN: 978-0803932500.

[ZP14] Zieris, F.; Prechelt, L.: On Knowledge Transfer Skill in Pair Programming. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '14, ACM, 2014.

[ZP16] Zieris, F.; Prechelt, L.: Observations on Knowledge Transfer of Professional Software Developers During Pair Programming. In: Proceedings of the 38th International Conference on Software Engineering Companion. ICSE '16 (SEIP), ACM, pp. 242–250, 2016.

[ZP20a] Zieris, F.; Prechelt, L.: Explaining Pair Programming Session Dynamics from Knowledge Gaps. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. ICSE '20, ACM, Seoul, South Korea, pp. 421–432, 2020, ISBN: 9781450371216.

[ZP20b] Zieris, F.; Prechelt, L.: PP-ind: A Repository of Industrial Pair Programming Session Recordings, 2020, arXiv: 2002.03121v3 [cs.SE].

Workshops

2nd Workshop on Requirement Management in Enterprise Systems Projects (AESP'21)

Christoph Weiss[1] Johannes Keckeis[2]

Abstract: 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 ïncorrectëxpectations, divergents in definition and attitudes on requirements management between customers and suppliers. These challenges will be highlighted, talked over and discussed during this workshop.

Keywords: Enterprise systems; Enterprise resource planning; Requirements management

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. These challenges will be highlighted, talked over and discussed during this workshop.

2 Expected Results (Work Objectives) of the 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 management. Sketching of a new requirement management model in an Enterprise Systems project for the further development of the existing system landscape.

[1] AUB's German-language interdisciplinary Ph.D. programme (Economics), Budapest, Hungary christoph.weiss@andrassyuni.hu
[2] Department of Strategic Management, Marketing and Tourism, University of Innsbruck, Innsbruck, Austria johannes.keckeis@uibk.ac.at

doi:10.18420/SE2021_49

3 Workshop Agenda

- Introduction to the topic and keynote by the organizers of the workshop.
- Presentation of the submitted and accepted papers with subsequent discussion.
- Summary and resume by the organizers.

4 Program Committee

- Martin Adam, FH Kufstein Tirol
- Wolfgang Ahammer, VFI GmbH
- Christian Büll, FH Burgenland
- Fritz Fahringer, Standortagentur Tirol GmbH
- Gunther Glawar, EVVA Sicherheitstechnologie GmbH
- Felix Piazolo, Universität Innsbruck
- Kurt Promberger, Universität Innsbruck
- Ludwig Rupp, Rupp AG
- Anton Vorraber, KTM AG

5 Perspective

The results of the development of the new procedure model for requirement management in an Enterprise Systems project for the further development of the existing system landscape are processed in a paper and submitted to the next Software Engineering Conference 2022.

18th Workshop on Automotive Software Engineering (ASE'21)

Patrick Ebel[1], Steffen Helke[2], Ina Schaefer[3], Andreas Vogelsang[4]

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 18th Workshop on Automotive Software Engineering (ASE'21) 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.

[1] Universität zu Köln, ebel@cs.uni-koeln.de
[2] Fachhochschule Südwestfalen, helke.steffen@fh-swf.de
[3] Technische Universität Braunschweig, i.schaefer@tu-braunschweig.de
[4] Universität zu Köln, vogelsang@cs.uni-koeln.de

doi:10.18420/SE2021_50

Many thanks to all the reviewers who contributed with great commitment to the review process.

As in previous years, the workshop will be opened with a keynote speech. We would like to thank Prof. Dr.-Ing. Markus Maurer (Director of the Department of Vehicle Electronics, Technische Universität Braunschweig), who will a give a talk on 'The Inherent Risk of Autonomous Road Vehicles'.

Program Committee

Prof. Dr. Paula Herber	Universität Münster
Dr. Verena Klös	Technische Universität Berlin
Prof. Dr. Stefan Kugele	Technische Hochschule Ingolstadt
Dr. Thomas Noack	Datendeuter GmbH
Prof. Dr. Dirk Nowotka	Universität Kiel
Prof. Dr. Jörn Schneider	Hochschule Trier
Prof. Dr. Thomas Thüm	Universität Ulm
Dr. Rebekka Wohlrab	Carniege Mellon University

Organization

Patrick Ebel	Universität zu Köln
Prof. Dr. Andreas Vogelsang	Universität zu Köln
Prof. Dr. Ina Schaefer	Technische Universität Braunschweig
Prof. Dr. Steffen Helke	Fachhochschule Südwestfalen

For many years, this workshop has been organized by the GI interest group (Fachgruppe) on "Automotive Software Engineering"[5]. The steering committee was consequently involved in the organization of this workshop as well.

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

3rd Workshop on Avionics Systems and Software Engineering (AvioSE'21)

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

Abstract: Software development in the aerospace domain is driven by new application potentials, increasing complexity, rising certification effort, and increasing cost pressure. In particular, future applications such as e.g., autonomous air transport, aircrew workload reduction, commercial UAVs, and further enhancement of existing functionality add to the system complexity. At the same time, there are challenges in communication and navigation in airspace, certification for multi-core processors, artificial intelligence as well as security for software, hardware, and connectivity. New software development methodologies and techniques are required for dealing with these challenges.

Keywords: avionics; systems engineering; software engineering; formal method; model-based; requirement; qualification; certification; simulation; process; tool

1 Introduction

Considerable advances for aerospace applications are expected in the course of introduction of new technologies such as artificial intelligence (AI), multi-core processors, and new communication technologies. However, the requirements in the domain do not allow the application of these technologies straight away, but require for additional techniques and mechanisms in order to meet the high quality needed for product certification. Many of the existing techniques can be amended or extended towards fulfillment of these requirements. These challenges are to be addressed in addition to the progress to be made in development efficiency and quality assurance. The previous workshops AvioSE'19[4] and AvioSE'20[5] dealt with general challenges and development tools. They demonstrated that many participants were in favour of generic aspects, but also wanted a deeper dive in carefully selected areas with significant future potential.

To this end, **future capabilities driven by AI** is selected for AvioSE'21 as the main topic. Starting with the collection of capabilities that can be enabled or improved with AI, it shall be figured out afterwards which areas of development require attention. For the virtual

[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)

doi:10.18420/SE2021_51

discussion it is envisaged to cover fields such as the certification of AI, need and possibilities of Design Assurance Level segregation or performance assurance.

2 Workshop Objectives

The main objective of the workshop is to accelerate the 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'21 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. Those contributions are presented in a scientific format, but the small character of the workshop allows detailed discussion. (2) **Future capabilities driven by AI** are selected to be the main topic of AvioSE'21. This is addressed interactively by inviting all participants to discuss aspects of AI. This covers connecting academics and professionals with invited experts. The panel discussion with invited experts from academia, industry, and authorities supports the identification of most important aspects in this area and propose ways how to address them. The proposals are at the same time challenged and/or amended by workshop participants. The results are collected on virtual desktops and are available to all participants. (3) The AvioSE'21 also allows for a wild card topic that might show up during the workshop.

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 and especially our keynote speakers, Andreas Bierig from DLR Braunschweig and Rolf Büse from Diehl Aerospace. Second, we want to express our gratitude to the SE 2021 organizers for supporting and hosting our workshop. Additionally, we are glad that these people (listed in alphabetic order) served as members in the program committee, soliciting papers, and writing peer reviews: Jun.-Prof. Björn Annighöfer (University of Stuttgart), Prof. Dr.-Ing. Steffen Becker (University of Stuttgart), Umut Durak (DLR Braunschweig), Prof. Dr. Ralf God (Hamburg University of Technology), Dr. Christian Heinzemann (Robert Bosch GmbH), Prof. Dr. Eric Knauss (University of Gothenburg), Dr. Winfried Lohmiller (Airbus Defence and Space GmbH), Dr. Christian Meißner (Volkswagen AG), Prof. Dr. Alexander Pretschner (Munich University of Technology), Dr. Stephan Rudolph (Northrop Grumman LITEF GmbH), Prof. Dr. Bernhard Rumpe (RWTH Aachen University), Dr. Andreas Schweiger (Airbus Defence and Space GmbH), Katja Stecklina (Philotech Systementwicklung und Software GmbH), Prof. Dr. Sebastian Voss (Aachen University of Applied Sciences). Finally, sincere thanks are given to the organization committee members' management for welcoming and supporting the workshop.

Anne Koziolek, Ina Schaefer, Christoph Seidl (Hrsg.): Software Engineering 2021,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 131

8th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS'21)

Robert Heinrich[1], Reiner Jung[2], Marco Konersmann[3], Eric Schmieders[4]

Abstract: Dieser Beitrag gibt eine Einführung in die EMLS-Workshopreihe. Die EMLS-Workshopreihe bietet ein Forum zur Diskussion und zum Wissensaustausch rund um die Evolution und Wartung langlebiger Software-intensiver Systeme.

Keywords: Software-intensive Systeme; Langlebige Systeme; Evolution; Wartung

Die Digitalisierung stellt neue Herausforderungen an die Entwicklung und den Betrieb von Software. Die Unterstützung oder sogar erst die Ermöglichung sozialer, politischer, wissenschaftlicher und ökonomischer Prozesse durch digitale Lösungen führt zu gesellschaftlichen Transformationen und verändert die Umgebung, die Nutzung und die Entwicklung von Softwaresystemen. Systeme müssen den wandelnden Bedürfnissen folgen aber dennoch den Qualitätsansprüchen der Nutzer genügen. Dies hat weitreichende Auswirkungen auf die Entwicklung der Systeme, insbesondere bei datenintensiven Systemen, dem Architektur-Management von betrieblicher und administrativer Software und der Entwicklung der Werkzeuge für die Entwicklung der Systeme. Konkrete Herausforderungen umfassen daher unter anderem die Verzahnung der Entwicklungsschritte und der verschiedenen Ebenen der Entwicklung (Anwendungsebene, Werkzeugebene), die Erklärbarkeit von Software und den zugrundeliegenden Entscheidungen, neue Analyseansätze und -methoden für ein besseres Systemverständnis, Konsistenz der Artefakte, sowie die Evolution von Plattformen und Frameworks. Diese sind zentrale Herausforderungen für langlebige softwareintensive Systeme.

Ziel der EMLS-Workshopreihe ist es, diese Herausforderungen gemeinsam aus Sicht der Wissenschaft und der Industrie zu beleuchten und unterschiedliche Sichtweisen zur Evolution und Wartung langlebiger Systeme zusammenzubringen.

Die EMLS-Workshopreihe bietet dazu ein Forum zur Diskussion von Problemstellungen, Lösungsansätzen und Evaluationsstrategien. Beiträge werden in Kleingruppen intensiv besprochen. So soll ein Austausch an Wissen unterstützt und eine Grundlage für Kooperationen geschaffen werden, welche die Bildung von zukünftigen gemeinschaftlichen Vorhaben sowohl zwischen Forschung und Industrie, als auch zwischen Forschenden fördert.

[1] Karlsruher Institut für Technologie (KIT) robert.heinrich@kit.edu
[2] Christian-Albrechts-Universität zu Kiel reiner.jung@email.uni-kiel.de
[3] Universität Koblenz Landau konersmann@uni-koblenz.de
[4] IT.NRW eric.schmieders@it.nrw.de

doi:10.18420/SE2021_52

Workshop on Software Engineering in Cyber-Physical Production Systems (SECPPS'21)

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

Abstract: This workshop focuses on Software Engineering in Cyber-Physical Production Systems. It is an interactive workshop opened by keynotes and statements by participants, followed by extensive discussions in break-out groups. The output of the workshop is a research roadmap as well as concrete networking activities to further establish a community in this interdisciplinary field.

Keywords: Software engineering; cyber-physical production systems; workshop

1 Motivation

Software is playing an increasingly important role in assuring effective and efficient operation of industrial automation engineering systems. However, software engineering methods applied in this field lag behind the conventional software engineering methods, where tremendous progress has been made in the last years.

Particularly, we are currently facing a dramatically increasing complexity in the development and operation of systems with the emergence of Cyber-Physical Production Systems (CPPS). This demands for more comprehensive and systematic views on all aspects of systems (e.g., mechanics, electronics, software, and network) not only in the engineering process, but in the operation process as well. Moreover, flexible approaches are needed to adapt the systems' behavior to ever-changing requirements and tasks, unexpected conditions, as well as structural transformations [Mo14].

The aim of this workshop is to discuss new approaches and methods for the design of software for use in the production systems domain, which follows the latest trends from the software engineering domain. Additionally, the workshop addresses the challenges (see [Vo15] as well as `http://www.dfg-spp1593.de/` and `https://www.sfb.tum.de/768/`) in adopting state-of-the-art software engineering tools and techniques to the automation domain and discusses various approaches to tackle the issues.

[1] Christian Doppler Lab VaSiCS, LIT CPS Lab, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, rick.rabiser@jku.at
[2] AIS, TU Munich, Boltzmannstr. 15, 85748 Garching bei München, Germany vogel-heuser@tum.de
[3] Christian Doppler Lab MINT, SE, Johannes Kepler University Linz, Altenberger Str. 69, 4040 Linz, Austria, manuel.wimmer@jku.at

doi:10.18420/SE2021_53

2 Program, Format, and Topics

The workshop has an interactive format to stimulate group discussions which potentially lead to further activities such as joint publications, projects, and networks.

To reach this goal, two keynote speakers, one from the SE community as well as one from the CPPS community, enlighten the audience with concrete experiences and thoughts from both fields in order to set the stage. These keynotes are followed by statements by invited participants. From each statement, questions, challenges, and provocative statements are collected to be potentially discussed in break-out groups.

The collected statements from the morning sessions are clustered and the participants can vote for topics to be discussed in break-out groups. The results of the break-out groups are presented (by the break-out group leaders) and discussed in the large audience to come up with a collaborative research roadmap and networks to identify potential collaborations.

For discussions we foresee the following initial list of topics:

- Engineering Process (Requirements, Design, Implementation, Testing, ...)
- Operation and Evolution (Data-driven, Continuous Integration, DevOps, 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, ...)
- Usability and SE Tools (Adoption, User Interactions, ...)
- Emerging technologies (Cloud, AI, IoT, ...)
- Intelligent organization (Multi-Agent Systems, Flexible Architectures, ..)
- Interdisciplinary collaboration (Interfaces, Conflict Management, Optimization, ...)

The workshop is concluded with setting up a collaborative space to document the results and help with future activities of the workshop.

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.

Workshop on Software Engineering for E-Learning Systems (SEELS'21)

Sven Strickroth[1], Michael Striewe[2]

Abstract: The workshop "Software Engineering for E-Learning Systems" (SEELS) is interested in the software engineering question related to the design of e-learning systems, the realization of networked e-learning landscapes at schools and universities, and the operation and maintenance of such systems. The goal is to identify and discuss current research questions in that area. This may include topics such as technical interfaces of e-learning systems, security issues in heterogeneous e-learning landscapes, or management of domain-specific requirements in universal e-learning systems.

Keywords: E-Learning; Software Engineering; System Design; Distributed E-Learning Systems

1 Background and Goals

The development of e-learning systems and the composition of networked e-learning landscapes with a large number of heterogeneous systems must take into account a wide variety of different requirements. In parts, e-learning systems resemble classical information systems, including the associated questions of scalability, expandability, maintainability and secure communication between the distributed components. In the course of ubiquitous learning, however, they also require extensive expertise in mobile software engineering, place high demands on data protection due to the processing of personal data, even during the basic design of the systems, and in innovative scenarios they borrow from games engineering and the design of virtual reality. Even during the development of individual systems, these conditions require a careful approach, while the trend towards creating networked e-learning landscapes within an educational institution or even across several institutions further increases the complexity of development and operation. Recent experiences with online teaching on a large scale have also revealed further weaknesses where issues in the software-technical design of e-learning systems limit their effective and efficient use.

While national and international e-learning conferences focus on (media-)didactical, subject-specific and socio-technical questions (including usability) and therefore pursue the question,

[1] Ludwig-Maximilians-Universität München, Oettingenstraße 67, D-80538 München, Germany, sven.strickroth@ifi.lmu.de
[2] Universität Duisburg-Essen, paluno - The Ruhr Institute for Software Technology, Gerlingstraße 16, 45127 Essen, Germany, michael.striewe@paluno.uni-due.de

doi:10.18420/SE2021_54

how to build good e-learning systems, this workshop will focus explicitly on software-technical questions and therefore pursue the question, *how to build e-learning systems well*.

The goal of the workshop is to make the special features of the domain visible and at the same time to sharpen the view for software-technical questions within the community as well as to offer a forum for relevant discussions. Concrete working topics can be, for example, the question of universal interfaces for establishing e-learning landscapes, (data) security in heterogeneous e-learning landscapes (including BYOD scenarios), the management of subject-specific requirements in university-wide e-learning landscapes, or the use of microservice architectures. Current research questions in these or similar areas are to be identified in the workshop in order to provide the community with links to general software engineering topics.

2 Working Mode

The workshop is conducted as a half-day workshop that offers space for research contributions as well as for reports from the successful practical development of e-learning systems. In addition, the workshop calls for position papers, which can be presented at the workshop in short impulse talks and should lead to the identification of research questions beyond the current state-of-the-art.

3 Committee

Organizers

- Sven Strickroth (Ludwig-Maximilians-Universität München)
- Michael Striewe (Universität Duisburg-Essen)

Program Committee

- Matthias Ehlenz (RWTH Aachen)
- Jörg Haake (FernUniversität Hagen)
- Johan Jeuring (Utrecht University)
- Stephan Krusche (Technische Universität München)
- Herbert Kuchen (Universität Münster)
- René Röpke (RWTH Aachen)
- Steffen Zschaler (King's College London)