Ernst Denert Software Engineering Award 2022

Eric Bodden, Michael Felderer, Wilhelm Hasselbring, Paula Herber, Heiko Koziolek, Carola Lilienthal, Florian Matthes, Lutz Prechelt, Bernhard Rumpe, and Ina Schaefer

E. Bodden
Secure Software Engineering, Heinz Nixdorf Institut der Universität Paderborn, Paderborn University and Fraunhofer IEM, Paderborn, Germany
e-mail: eric.bodden@uni-paderborn.de

M. Felderer
Institute for Software Technology, German Aerospace Center (DLR), University of Cologne, Cologne, Germany
e-mail: michael.felderer@uni-koeln.de

W. Hasselbring
Software Engineering, Christian-Albrechts-Universität Kiel, Kiel, Germany
e-mail: hasselbring@email.uni-kiel.de

P. Herber
Embedded Systems Group, University of Münster, Münster, Germany
e-mail: paula.herber@uni-muenster.de

H. Koziolek
ABB Corporate Research, Ladenburg, Germany
e-mail: heiko.koziolek@de.abb.com

C. Lilienthal
WPS - Workplace Solutions GmbH, Hamburg, Germany
e-mail: carola.lilienthal@wps.de

F. Matthes
Software Engineering of Business Information Systems, Department of Computer Science (CS), Technical University of Munich, Garching bei München, Germany
e-mail: matthes@in.tum.de

L. Prechelt
Institut für Informatik, Freie Universität Berlin, Berlin, Germany
e-mail: prechelt@inf.fu-berlin.de

B. Rumpe (✉)
Software Engineering, RWTH Aachen, Aachen, Germany
e-mail: rumpe@se-rwth.de

I. Schaefer
Testing, Validation and Analysis of Software-Intensive Systems (TVA), Institute for Information Security and Dependability (KASTEL), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
e-mail: ina.schaefer@kit.edu

E. Bodden et al. (eds.), *Ernst Denert Award for Software Engineering 2022*,
https://doi.org/10.1007/978-3-031-44412-8_1

Abstract The Ernst Denert Award is already existing since 1992, which does not only honor the award winners but also the software engineering field in total. Software engineering is a vivid and intensively extending field that regularly spawns new subfields such as *automotive software engineering*, *research software engineering*, or *quantum software engineering*, covering specific needs but also generalizing solutions, methods, and techniques when they become applicable. This is the introductory chapter of the book on the Ernst Denert Software Engineering Award 2022. It provides an overview of the five nominated PhD theses.

1 Introduction

Software-based products, apps, systems, or other services are influencing all areas of our daily life. They are the basis and central driver for digitization and all kinds of innovation. This makes software engineering a core discipline to drive technical and societal innovations in the age of digitization [4].

As of 2023, software engineering operates in many new or significantly changed application domains, such as the Internet of Things (IoT), smart manufacturing, autonomous systems, machine learning, artificial intelligence (AI), and even quantum computing. Surveys argue that more than 90% of research projects use software for gaining new insights, managing their results, understanding the research topic, controlling the physical gadgets, etc. Researchers of nearly all domains are significantly developing software within their research. Model-driven software and systems engineering approaches nowadays support handling the ever-growing complexity of modern systems. Sophisticated static analysis tools identify more and more faults in the code and can mitigate the rising cyber-security challenges by identifying security vulnerabilities early or monitoring the system during runtime for a safe, reliable, robust, and secure operation.

A rather strong recent trend, which affects software engineering practices, is the advent of generative AI, thanks to large language models (LLMs) based on the transformer architecture [10]. These models were popularized in recent months by publicly available, easy-to-use tools (e.g., GitHub CoPilot, ChatGPT, Bard). Such tools can generate source code based on natural language queries but can also interpret, fix, or document existing code. Trained with a vast data set including many popular libraries, such LLMs can potentially relieve software engineers from many accidental complexities and focus on the essential complexities of solving computing problems. Early experiments at Microsoft Research demonstrated a 55% developer productivity increase from using GitHub CoPilot for web programming, signifying promising potential for advancing software development practices [7].

While some authors already pro-claim “the end of programming” [9], the technology is still under development. LLMs sometimes find very helpful sentences and programs but sometimes only hallucinate. Generated source code thus may be partially semantically incorrect or doing something completely wrong. We will

have to evaluate the new technology carefully. It will affect software engineering research to utilize generative AI for the development of programs, models, and the understanding of requirements to the fullest. It may be that the new approaches will leverage methods from psychology, where intelligent interrogation allows to reveal how an AI really works.

We see a forthcoming challenging and very interesting future for software engineering research, not only for the application of AI models for software development but also for specific upcoming domains, such as research software engineering [5] or quantum computing [8].

It is important to recall that the *IEEE Standard Glossary of Software Engineering Terminology* [6] defines software engineering as follows:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
(2) The study of approaches as in (1).

It defines software engineering as an engineering discipline ("application of engineering to software") with its own methodology ("systematic, disciplined, quantifiable approach") applied to all phases of the software life cycle ("development, operation, and maintenance of software"). The two-part structure of the definition of software engineering also makes the tight integration of software engineering (1) and software engineering research (2) explicit.

Therefore, the Ernst Denert Software Engineering Award specifically rewards researchers who value the practical impact of their work and aim to improve current software engineering practices [3]. Creating tighter feedback loops between professional practitioners and academic researchers is essential to make research ideas ready for industry adoption. Researchers who demonstrate their proposed methods and tools on nontrivial systems under real-world conditions in various phases of the software life cycle shall be supported so that the gap between research and practice can be decreased.

Overall, five PhD theses that were defended between September 1, 2021, and October 31, 2022, were nominated and finally presented during the Software Engineering Conference SE 2023.

All submissions fulfill the ambitious selection criteria of the award defined in detail in the book for the Ernst Denert Software Engineering Award 2019 [2]. These criteria include, among others, practical applicability, usefulness via tools, theoretical or empirical insights, currentness, and contribution to the field. In a nutshell, "The best submissions are those that will be viewed as important steps forward even 15 years from now." [3].

In this introductory chapter, we give an overview of the nominated five PhD theses, present the work of the award winner, and outline the structure of the book.

2 Overview of the Nominated PhD Theses

As previously mentioned, the Ernst Denert Software Engineering Award 2022 committee identified five worthy nominations for PhD theses that were eligible to receive the Ernst Denert Award. These theses encompass a wide range of research in the field of software engineering, highlighting its diverse applications across various domains. They also demonstrate the vibrancy and diversity of the field through the utilization of different research methods, including formal methods, design science, and quantitative and qualitative empirical methods. Furthermore, these theses address various activities in the software life cycle, such as analysis, design, programming, testing, deployment, operation, and maintenance. This section provides a brief overview of the nominated PhD theses. They will be presented in alphabetical order based on the names of the respective nominees, accompanied by a concise summary of the chapters contributed by each thesis to this book.

The chapter of Jannik Fischbach and Andreas Vogelsang entitled "Conditional Statements in Requirements Artifacts: Logical Interpretation, Use Cases for Automated Software Engineering, and Fine-Grained Extraction" provides readers with an understanding of (1) the notion of conditionals in RE artifacts, (2) how to extract them in fine-grained form, and (3) the added value that the extraction of conditionals can provide to RE. Jannik Fischbach is the winner of the Ernst Denert Software Engineering Award 2022, and we present his work in more detail in the next section.

The chapter of Jörg Christian Kirchhof entitled "From Design to Reality: An Overview of the MontiThings Ecosystem for Model-Driven IoT Applications" proposes a model-driven process for rapid development of IoT applications. The chapter gives an overview of how to develop, deploy and analyze distributed IoT applications using MontiThings. MontiThings demonstrates the benefits of a model-driven development approach not only in the initial conceptualization of the application but also in later development phases (e.g., deployment), leading to an app store concept that separates hardware from software development.

The chapter of Sven Peldszus entitled "Security Compliance in Model-Driven Development of Software Systems in Presence of Long-Term Evolution and Variants" provides an approach for tracing and verifying security requirements in the model-driven development of software-intensive systems. Early security considerations based on the principle of security by design are part of many modern development processes, but to ensure the security of the final product, which may even comprise an entire product line, it is essential to check each individual product for compliance with the planned security design. To this end, the thesis investigates the systematic traceability of security requirements throughout the software development life cycle and how this traceability can be used for automated security compliance checking. The individual solutions were validated against 18 objectives, and the overall approach was demonstrated on two open-source case studies.

The chapter of Florian Rademacher et al., entitled "Model-Driven Engineering of Microservice Architectures: The LEMMA Approach", investigates the application

of model-driven engineering (MDE) to the design, development, and operation of software systems that are based on microservice architecture (MSA). From a set of well-known challenges in MSA engineering as well as real-world microservice architectures and approaches to the modeling of service-oriented architectures, Rademacher et al. derive a set of integrated, stakeholder-oriented MSA modeling languages. Furthermore, they accompany these languages with a framework for the implementation of model processors that is oriented toward technology-savvy MSA stakeholders without an MDE background. Finally, Rademacher et al. present and discuss the application of their MSA modeling languages and framework for the (i) extensible generation of microservice code; (ii) microservice architecture reconstruction; (iii) quality assessment of microservices; (iv) microservice architecture defect resolution; and (v) establishment of a common architecture understanding among distributed MSA teams.

Finally, the chapter of Alexander Trautsch entitled "Usefulness of Automatic Static Analysis Tools: Evidence from Four Case Studies" presents results from multiple empirical studies in the context of software engineering research. The studies explore an automated static analysis tool and its impact on quality in a broad overview from multiple perspectives. The chapter contains studies that focus on the evolution of static analysis warnings, static analysis warnings in the context of software defects, as well as the context of developer intent.

3 The Work of the Award Winner

We congratulate *Jannik Fischbach*, his advisor *Andreas Vogelsang*, and his alma mater, Universität zu Köln, for winning the Ernst Denert Software Engineering Award 2022 for the PhD thesis "Why and How to Extract Conditional Statements From Natural Language Requirements." Dr. Jannik Fischbach focuses on conditionals (e.g., "If the system detects an error, an error message shall be shown") in requirements and highlights **why** and **how** *requirements engineering* can benefit from the automated extraction of conditionals. Specifically, he makes the following contributions:

1. He presents empirical results on the prevalence and logical interpretation of conditionals in RE artifacts. Jannik Fischbach found that conditionals in requirements mainly occur in explicit, marked form and may include up to three *antecedents* and two *consequents*. Hence, the extraction approach must understand conjunctions, disjunctions, and negations to fully capture the relation between *antecedents* and *consequents*. He also found that conditionals are a source of ambiguity, and there is not just one way to interpret them formally. This affects any automated analysis that builds upon formalized requirements (e.g., inconsistency checking) and may also influence guidelines for writing requirements.

2. Jannik Fischbach presents his tool-supported approach CiRA capable of detecting conditionals in NL requirements and extracting them in fine-grained form. For the detection, CiRA uses syntactically enriched BERT embeddings combined with a softmax classifier and outperforms existing methods. His experiments show that a sigmoid classifier built on RoBERTa embeddings is best suited to extract conditionals in fine-grained form. CiRA is available at http://www.cira.bth.se/demo/.
3. He highlights how extracting conditionals from requirements can help create acceptance tests automatically. Specifically, Jannik Fischbach shows how extracted conditionals can be mapped to a *Cause-Effect-Graph* from which test cases can be derived automatically. He demonstrates the feasibility of his approach in a case study with three industry partners. In his study, out of 578 manually created test cases, 71.8% can be generated automatically. Furthermore, his approach discovered 80 relevant test cases missed in manual test case design.

His findings prove that automated conditional extraction can contribute to implementing automatic acceptance test creation. However, he does not achieve full automation of acceptance test generation mainly due to (1) incomplete requirements and (2) errors of his approach in interpreting conditionals that contain three or more *consequents*. Hence, Jannik Fischbach suggests using CiRA to supplement the existing manual creation process to make test designers aware of all test cases that should be tested from a combinatorial point of view. He hypothesizes that this will help reduce the risk of missed negative test cases significantly. The work of Jannik Fischbach is presented in more detail in Chapter "Conditional Statements in Requirements Artifacts: Logical Interpretation, Use Cases for Automated Software Engineering, and Fine-Grained Extraction" of this book.

4 Structure of the Book

The remainder of the book is structured into five chapters, one for the work of each nominee listed above. Each nominee presents in his chapter

- an overview and the key findings of the work,
- its relevance and applicability to practice and industrial software engineering projects,
- additional information and findings that have only been discovered afterwards, e.g., when applying the results in industry or when continuing research.

The chapters of the nominees are based on their PhD theses and arranged in alphabetic order.

As already highlighted in the introductory book chapter of the Ernst Denert Software Engineering Award 2019 [3] and by Prof. Denert's reflection on the field [1], software engineering is teamwork. Outstanding research with high impact

is also always teamwork, which somewhat conflicts with the requirement that a doctoral thesis must be the work of a single author.

4.1 Thanks

We again thank Professor Ernst Denert for all his help in making this award a success and the *Gerlind & Ernst Denert-Stiftung* for the kind donation of the first price and the overall support. We thank the team of the Software Engineering Conference SE 2023, which was organized by Gregor Engels, Stefan Sauer, Regina Hebig and Matthias Tichy at Paderborn University, to host the presentations of the nominees and the award ceremony. We also thank the German, Austrian, and Swiss computer science societies, i.e., the GI, the OCG, and the SI, respectively, for their support in making the Ernst Denert Software Engineering Award 2022 a success. Finally, we thank all the people that helped in its organization, including Christian Kirchhof and Florian Rademacher (both RWTH Aachen University), who supported in the organization of this book.

References

1. Denert, E.: Software engineering. In: Ernst Denert Award for Software Engineering 2019, pp. 11–17. Springer, Berlin (2020)
2. Felderer, M., Hasselbring, W., Koziolek, H., Matthes, F., Prechelt, L., Reussner, R., Rumpe, B., Schaefer, I.: Ernst Denert Award for Software Engineering 2019: Practice Meets Foundations (2020)
3. Felderer, M., Hasselbring, W., Koziolek, H., Matthes, F., Prechelt, L., Reussner, R., Rumpe, B., Schaefer, I.: Ernst denert software engineering awards 2019. In: Ernst Denert Award for Software Engineering 2019, pp. 1–10. Springer, Berlin (2020)
4. Felderer, M., Reussner, R., Rumpe, B.: Software Engineering und Software-Engineering-Forschung im Zeitalter der Digitalisierung. Informatik Spektrum **44**(2), 82–94 (2021)
5. Felderer, M., Goedicke, M., Grunske, L., Hasselbring, W., Lamprecht, A.L., Rumpe, B.: Toward Research Software Engineering Research (2023). https://doi.org/10.5281/zenodo.8020525
6. IEEE: IEEE standard glossary of software engineering terminology. IEEE Std 610.12-1990 pp. 1–84 (1990)
7. Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M.: The impact of AI on developer productivity: Evidence from github copilot (2023). Preprint arXiv:2302.06590
8. Schaefer, I.: Quantum software engineering - quo vadis? In: Engels, G., Hebig, R., Tichy, M. (eds.) Software Engineering 2023, Fachtagung des GI-Fachbereichs Softwaretechnik, 20–24. Februar 2023, Paderborn, LNI, vol. P-332, pp. 19–20. Gesellschaft für Informatik e.V., Luxembourg (2023). https://dl.gi.de/20.500.12116/40069
9. Welsh, M.: The end of programming. Commun. ACM **66**(1), 34–35 (2022)
10. Zhou, C., Li, Q., Li, C., Yu, J., Liu, Y., Wang, G., Zhang, K., Ji, C., Yan, Q., He, L., et al.: A comprehensive survey on pretrained foundation models: A history from bert to chatgpt (2023). Preprint arXiv:2302.09419