Ernst Denert Software Engineering Award 2020

Michael Felderer, Wilhelm Hasselbring, Heiko Koziolek, Florian Matthes, Lutz Prechelt, Ralf Reussner, Bernhard Rumpe, and Ina Schaefer

Abstract This is the introductory chapter of the book on the Ernst Denert Software Engineering Award 2020. It provides an overview of the 11 nominated PhD theses, the work of the award winner, and the structure of the book.

1 Introduction

Software-based products, systems, or 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

M. Felderer (✉)
Department of Computer Science, University of Innsbruck, Innsbruck, Austria

W. Hasselbring
Department of Computer Science, Kiel University, Kiel, Germany

H. Koziolek
Corporate Research, ABB, Ladenburg, Germany

F. Matthes
Institute of Computer Science, Technical University Munich, Garching, Germany

L. Prechelt
Department of Mathematics and Computer Science, Freie Universität Berlin, Berlin, Germany

R. Reussner
Program Structures and Data Organization, Karlsruhe Institute of Technology, Karlsruhe, Germany

B. Rumpe
Software Engineering, RWTH Aachen University, Aachen, Germany

I. Schaefer
Software Engineering and Automotive Informatics, Technische Universität Braunschweig, Braunschweig, Germany

M. Felderer et al. (eds.), *Ernst Denert Award for Software Engineering 2020*,
https://doi.org/10.1007/978-3-030-83128-8_1

innovations in the age of digitization [4]. The IEEE Standard Glossary of Software Engineering Terminology [5] 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 lifecycle shall be supported, so that the gap between research and practice can be decreased.

Overall, 11 PhD theses that were defended between September 1, 2019, and October 31, 2020, were nominated and finally presented during the Software Engineering Conference SE 2021.

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].

This book includes a chapter on the jury decision process (chapter "Some Patterns of Convincing Software Engineering Research, or: How to Win the Ernst Denert Software Engineering Award 2020"), which might help future candidates to shape the presentation of their work for the Ernst Denert Software Engineering Award and beyond.

In this introductory chapter we give an overview of the nominated 11 PhD theses, present the work of the award winner, and outline the structure of the book.

2 Overview of the Nominated PhD Theses

As mentioned before, 11 PhD theses were nominated for the Ernst Denert Software Engineering Award 2020. They cover the broad range of software engineering research; show the wide spectrum and liveliness of the field with respect to different

application domains (covering information systems, embedded systems, and IoT systems in different application contexts) and research methods (covering formal methods, design science, as well as quantitative and qualitative empirical methods); and address software lifecycle activities (covering analysis, design, programming, testing, deployment, operation, and maintenance). In the following, we present a short overview of the nominee's PhD theses (alphabetically sorted by the nominee's names) by a short summary of the chapters on the respective theses contributed to this book.

The chapter by Jonathan Immanuel Brachthäuser entitled **"What You See Is What You Get: Practical Effect Handlers in Capability-Passing Style"** aims to bring effect handlers in programming languages closer to the software engineering practice by developing capability passing as an implementation technique for effect handlers. Capability passing provides the basis for the integration of effect handlers into mainstream object-oriented programming languages and thereby unlocks novel modularization strategies.

The chapter by Mojdeh Golagha entitled **"How to Effectively Reduce Failure Analysis Time?"** proposes methodologies and techniques supporting developers in knowing which data they need for further analysis, in being able to group failures based on their root causes, and in being able to find more information about the root causes of each failing group.

The chapter by Nikolay Harutyunyan entitled **"Open Source Software Governance: Distilling and Applying Industry Best Practices"** develops and evaluates a theory of industry best practices which captures how more than 20 experts from 15 companies worldwide govern their corporate use of open source software. Furthermore, the theory is operationalized via a handbook for open source governance that enables practitioners from various domains to apply the findings in their companies.

The chapter by Dominic Henze entitled **"Dynamically Scalable Fog Architectures"** provides a framework called xFog (Extension for Fog Computing) that models fog architectures based on set theory and graphs. It contains parts to establish the set-theoretical foundations to enable dynamic and scalable fog architectures to dynamically add new components or layers, and to provide a view concept which allows stakeholders to focus on different levels of abstraction.

The chapter by Anne Hess entitled **"Crossing Disciplinary Borders to Improve Requirements Communication"** investigates role-specific views in software requirements specifications. Based on a series of empirical studies that served as a baseline for a secondary analysis role-specific views are defined and evaluated. Moreover, a proof-of-concept implementation is realized that is capable of generating role-specific views.

The chapter by István Koren entitled **"DevOpsUse: A Community-Oriented Methodology for Societal Software Engineering"** introduces the DevOpsUse methodology, which extends DevOps by additionally fostering a stronger involvement of end user communities in software development by including them in the process of infrastructuring, that is, the appropriation of infrastructure during its usage. The developed DevOpsUse methodology and support tools are validated by the transitions between three generations of technologies: near-real-time peer-to-peer web architectures, edge computing, and IoT systems.

The chapter by Yannic Noller entitled **"Hybrid Differential Software Testing"** proposes the concept of Hybrid Differential Software Testing (HyDiff) as a hybrid analysis technique to generate difference-revealing inputs. HyDiff consists of two components that operate in a parallel setup, that is, a search-based technique that inexpensively generates inputs and a systematic exploration technique to also exercise deeper program behaviors. HyDiff is based on differential fuzzing directed by differential heuristics and differential dynamic symbolic execution that allows to incorporate concrete inputs in its analysis.

The chapter by Dominic Steinhöfel entitled **"Ever Change a Running System: Structured Software Reengineering Using Automatically Proven-Correct Transformation Rules"** proposes the concept of structured software reengineering replacing the ad hoc forward engineering part of a reengineering process with the application of behavior-preserving, proven-correct transformations improving nonfunctional program properties. Furthermore, a specification and verification framework for statement-based program transformation rules on Java programs building on symbolic execution is presented. It is applied to code refactoring, cost analysis of program transformations, and transformations reshaping programs for the application of parallel design patterns. Finally, a workbench for modeling and proving statement-level Java transformation rules is provided.

The chapter by Peter Wägemann entitled **"Static Worst-Case Analyses and Their Validation Techniques for Safety-Critical Systems"** provides a novel program analysis approach for worst-case energy consumption bounds, which accounts for temporarily power-consuming devices, scheduling with fixed real-time priorities, synchronous task activations, and asynchronous interrupt service routines. Regarding the validation of worst-case tools, a technique for automatically generating benchmark programs is provided. The generator combines program patterns so that the worst-case resource consumption is available along with the generated benchmark. Knowledge about the actual worst-case resource demand then serves as the baseline for evaluating and validating program analysis tools. The benchmark generator helped to reveal previously undiscovered software bugs in a widespread worst-case execution time tool for safety-critical systems.

The chapter by Michael von Wenckstern entitled **"Improving the Model-Based Systems Engineering Process"** provides approaches and tools for supporting the automotive software engineer with automatic consistency checks of large component and connector models, automatic verification of these models against design decisions, tracing and navigating between design and implementation models, finding structural inconsistencies during model evolution, defining different extra-functional properties for component and connector models, and formalizing constraints on these models for extra-functional properties for automatic consistency checks.

Finally, the chapter by Franz Zieris entitled **"Understanding How Pair Programming Actually Works in Industry: Mechanisms, Patterns, and Dynamics"** is aimed at understanding how pair programming actually works by uncovering the underlying mechanisms and in order to ultimately formulate practical advice for developers by following a grounded theory methodology. Franz Zieris is the winner

of the Ernst Denert Software Engineering Award 2020, and we present his work in more detail in the next section.

3 The Work of the Award Winner

We congratulate *Franz Zieris*, his advisor *Lutz Prechelt*, and his Alma Mater Freie Universität Berlin for winning the Ernst Denert Software Engineering Award 2020 for the PhD thesis "Qualitative analysis of knowledge transfer in pair programming."

Dr. Franz Zieris used qualitative research methods to break down the actual work process involved in pair programming (PP). There has been a lot of research on PP in the last 20 years, but it provided remarkably little insight overall:

- In controlled experiments, PP is usually faster than solo programming, but sometimes not (and almost never twice as fast). The results are often better with respect to design quality or reliability than with solo programming, but sometimes they are not.
- The subject of the experiments was almost always toy programs. The results' generalizability to realistic code bases is completely unknown.
- Explanations for the considerable variations between experiments are almost nonexistent.

The work of Franz Zieris has now conceptualized what is going on in the pair programming process in an industrial context:

1. Zieris found a number of behavioral patterns and anti-patterns that can explain why the previous quantitative results have varied so much. For example, the much-investigated differences in programming experience play a much smaller role than whether the developers manage to build and maintain a common mental model of the software system.
2. These patterns provide professional pair programmers and developers who do not like pair programming so far with an opportunity to reflect on their behavior and avoid inefficient behaviors.
3. Zieris found an overall dynamic of PP sessions that makes it clear that the previous experiments completely lack the central element of professional PP sessions, namely the alignment and acquisition of system understanding; thus, the realism of previous experiments is very low.
4. From these findings it is derived which pair constellations are helpful which can help teams to use PP wisely.
5. Both advantages, that is, (2) and (4), were validated with professional developers.

The spectacular thing about Franz Zieris' work is the enormous generality of the improvements, because the results are applicable in any application domain and with any technology, and they are also timeless. The work of Franz Zieris is presented in more detail in chapter "Understanding How Pair Programming Actually Works in Industry: Mechanisms, Patterns, and Dynamics" of this book.

4 Structure of the Book

The remainder of the book is structured as follows. In the next chapter Lutz Prechelt describes the jury's decision process, which might help future candidates to shape the presentation of their work for the Ernst Denert Software Engineering Award and beyond.

This is followed by 11 chapters, one for the work of each nominee listed above. Each nominee presents in his or her 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 afterward, for example, 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 last year's award [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. Still, the respective chapters are written by the nominees only.

Thanks

We gratefully thank Professor Ernst Denert for all his help in the process and the *Gerlind & Ernst Denert-Stiftung* for the kind donation of the price and the overall support. We thank the organization team of the Software Engineering Conference SE 2021 that was virtually organized by the Technical University of Braunschweig to host the presentations of the nominees and the award ceremony. Finally, we thank the German, Austrian, and Swiss computer science societies, that is, the GI, the OCG, and the SI, respectively, for their support in making the Ernst Denert Software Engineering Award 2020 a success.

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 and software-engineering-Forschung im Zeitalter der Digitalisierung. Inform. Spektrum **44**(2), 82–94 (2021)
5. IEEE: IEEE standard glossary of software engineering terminology. IEEE Std 610.12-1990 pp. 1–84 (1990)