Autom Softw Eng (2018) 25:45–46
https://doi.org/10.1007/s10515-017-0222-5

Guest editorial: selected areas in automated software engineering

Marsha Chechik[1] · Paul Grünbacher[2]

Received: 24 July 2017 / Accepted: 4 August 2017 / Published online: 16 August 2017

It is a wonderful time to do research in automated software engineering! Everything is becoming programmable—phones, TVs, tablets, cars, and even watches and glasses. Software engineers are the ones who bring life to these programmable devices by writing systems and applications software. As software engineering researchers, we are the ones who are tasked with developing techniques and tools that will help software engineers in meeting the exploding demand in software production. We are facing an endless list of research problems that not only bring many challenges, but also bring many opportunities for contribution and impact.

In their paper "Self-Adaptive Concurrent Components", Österlund, Löwe look at the problem of selecting the optimum component implementation variant. Doing so is often difficult since it can depend on the component's usage context at runtime, including the concurrency level of the application using the component, call sequences, available hardware, etc. Optimal component implementation selection cannot be done statically, and a single optimal variant might not even exist since the usage contexts can change significantly over the runtime. The authors introduce self-adaptive concurrent components that automatically and dynamically change not only their internal representation and operation implementation variants but also their synchronization

✉ Paul Grünbacher
paul.gruenbacher@jku.at

Marsha Chechik
chechik@cs.toronto.edu

1 Department of Computer Science, University of Toronto, 10 King's College Road, Toronto, Ontario, Canada

2 Institute Software Systems Engineering, Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria

mechanism based on a possibly changing usage context. This approach allows selecting the most suitable variant at runtime rather than at compile time, revising the decision if the context changes. This allows programmers to defer implementation and optimization decisions about their programs to context-aware runtime optimizations. The authors demonstrate the effect on performance of their approach with self-adaptive concurrent queues, sets, and ordered sets. In all three cases, experimental evaluation shows close to optimal performance.

In their paper "Automatic Performance Prediction of Multithreaded Programs: a Simulation Approach", Tarvo and Reiss look at the problem of performance of multithreaded programs. It is often difficult to understand and predict since multiple threads engage in synchronization operations and use hardware simultaneously, resulting in a complex non-linear dependency between the configuration of a program (the number of threads, the hardware characteristics) and its performance. Performance prediction models, as their name suggests, allow to predict the impact of different program configurations on the performance; yet, building models of large applications manually is extremely time-consuming and error-prone. The paper presents an approach for the automatic construction of performance models of multi-threaded programs. The approach uses a combination of static and dynamic analyses of a single representative run of a system to collect information about the structure of the program, the semantics of interaction between the program's threads, and resource demands of individual program's components. The experiments demonstrate that the built models can accurately predict performance of various multi-threaded programs, including complex industrial applications.

In his paper "Seeking the User Interface", Reiss looks at the complex and often messy problem of interface design and coding. He proposes a system that uses code search to simplify and automated the exploration of such code, starting with a simple sketch of the desired interface along with a set of keywords describing the application context. This resulting solutions constitute only the user interface and yet can compile and run. The paper compares the generated interfaces with user sketches and reports on the experiment where programmers can interact with the generated interfaces and end up with the running code for the solutions they choose, validating the usefulness of the system for exploring alternative interfaces and for looking at graphical user interfaces in a code repository.

We are deeply grateful to the authors for their excellent submissions and for the reviewers of this special section for their time and feedback. We hope you will enjoy reading this special section.