Automated Software Engineering (2018) 25:743–744
https://doi.org/10.1007/s10515-018-0245-6

Guest editorial: advanced topics in automated software engineering

Lars Grunske[1] · **Mike Whalen**[2]

Received: 3 September 2018 / Accepted: 5 September 2018 / Published online: 21 September 2018

Welcome to the special issue of the Automated Software Engineering journal. Software now flourishes at many scales, from giant service-oriented cloud systems to tiny embedded IOT devices. Automation approaches for software design, construction, verification, and deployment are now regularly applied by practitioners to build scalable and robust software systems. As software engineering researchers, we must continue to improve the techniques and tools available to engineers so that this software, which has become central to many aspects of our societies, can be safely and cost-effectively fielded. The problems in constructing good software are substantial, as are the opportunities for improvement and impact for practitioners. This issue contains work automating aspects of design, synthesis, and verification drawn from highly rated papers from the ASE 2015 conference.

In their paper "Inferring Visual Contracts from Java Programs", Alshanqiti, Heckel, and Kehrer describe an approach to automatically construct visual contracts from Java programs. Such contracts are important for program understanding, testing, and analysis, but have in the past have required extensive manual effort to construct. Their approach infers accurate visual contracts based on examining program behavior dynamically over execution traces. The resulting specifications include object transformations, pre- and post-conditions in terms of object structures, parameter and attribute values, and generalised specification by universally quantified (multi) objects, patterns, and invariants. The paper focuses on construction techniques but also explores potential uses for many tasks in software engineering.

In "Synthesis of Probabilistic Models for Quality-of-Service Software Engineering", Gerasimou, Calinescu, and Tamburrelli examine a similar problem of architectural variants in terms of Quality of Service (QoS). They examine a problem

✉ Lars Grunske
grunske@informatik.hu-berlin.de

Mike Whalen
mwwhalen@umn.edu
http://www.cs.umn.edu/∼mwwhalen

1 Humboldt-Universität zu Berlin, Berlin, Germany

2 University of Minnesota, Minneapolis, MN, USA

space involving many alternative architectures and instantiations of system parameters that impact QoS which lead to more variants than can be reasonably analyzed. Their approach and tool, called EvoChecker, employs evolutionary algorithms to automate the model synthesis process to synthesise Pareto-optimal sets of probabilistic models associated with the QoS requirements of a system under design, and to support the selection of a suitable system architecture and configuration. EvoChecker can also be used for self-adaptation at runtime.

In "Static Window Transition Graphs for Android", Yang, Wu, Zhang, Wang, Swaminathan, Yan, and Rountev construct models of GUIs of Android applications. The approach constructs a graph (the window transition graph) that describes the set of windows that may be displayed and the transitions between them in terms of events and callbacks. Such graphs allow a concise description of the "flow" of applications and potentially unexpected transition sequences in the GUI. It is based on careful modeling of the active window stack and operations that modify this stack. They then describe how the WTG can be used for test-case generation and evaluate its performance and efficacy.

In "Developing and Evolving a DSL-Based Approach for Runtime Monitoring of Systems of Systems", Rabiser, Thanhofer-Pilisch, Vierhauser, Grünbacher, and Egyed examine an approach for monitoring heterogeneous large-scale complex software systems. Due to the heterogeneity and (often) different organizational boundaries in systems-of-systems (SoS) software, it is often not possible to determine system behavior at runtime. The authors present a rich domain specific language that can determine whether or not the SoS is meeting its requirements, and evaluate it on an industrial SoS from a collaborating organization. The evaluation demonstrates that while a DSL is a good solution, such languages must be extensible to evolve to support the needs of a diverse and evolving group of engineers.

Finally, in the paper "How Verified (or Tested) is My Code? Falsification-Driven Verification and Testing", Groce, Ahmed, Jensen, McKenney and Holmes propose a new falsification-driven methodology for formal verification and automated testing. The goal is to determine the adequacy of a particular verification effort in terms of mutation analysis. The methodology provides substantial insights into the interplay between the strength of the oracle (e.g., the property set) and the strength of the exploration (structural coverage of the test suite or depth of bounded verification). It also enables a uniform approach to verification using multiple techniques, where the verification activities are driven by the number of remaining mutants.

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.