International Journal on Software Tools for Technology Transfer (2018) 20:611–613
https://doi.org/10.1007/s10009-018-0504-7

INTRODUCTION

Software quality tools and techniques presented in FASE'17

Marieke Huisman[1] · **Julia Rubin**[2]

Published online: 3 September 2018

Abstract
Software quality assurance aims to ensure that the software product meets the quality standards expected by the customer. This special issue of Software Tools for Technology Transfer is concerned with the foundations on which software quality assurance is built. It introduces the papers that focus on this topic and that have been selected from the 20th International Conference on Fundamental Approaches to Software Engineering (FASE'17).

Keywords Software engineering · Software foundations · Software quality

1 Introduction

This special issue of the journal Software Tools for Technology Transfer (STTT) contains revised and extended versions of six papers selected out of 25 papers presented at the 20th International Conference on Fundamental Approaches to Software Engineering (FASE'17) [4].

FASE is concerned with the foundations on which software engineering is built. Papers published at FASE describe contributions that make software engineering a more mature and soundly based discipline and are supported by appropriate arguments and validation. Topics that are of interest to FASE are:

- Software engineering as an engineering discipline, including its interaction with and impact on society;
- Requirements engineering: capture, consistency, and change management of software requirements;
- Software architectures: description and analysis of the architecture of individual systems or classes of applications;
- Specification, design, and implementation of particular classes of systems: adaptive, collaborative, embedded, distributed, mobile, pervasive, or service-oriented applications;
- Software quality: validation and verification of software using theorem proving, model checking, testing, analysis, refinement methods, metrics or visualization techniques;
- Model-driven development and model transformation: meta-modeling, design and semantics of domain-specific languages, consistency and transformation of models, generative architectures;
- Software processes: support for iterative, agile, and open source development;
- Software evolution: refactoring, reverse and re-engineering, configuration management and architectural change, or aspect-orientation.

The peer-reviewed papers collected in this special issue have been invited by the guest editors among the top papers presented at FASE'17 based on their relevance to STTT. They all report on advances in tools and techniques for software quality assurance.

The selected papers cover two domains: tools for system developers and techniques for enhancing quality-assurance tools. In the first domain, there are three papers that aim at helping developers to improve the quality of their systems. They are discussed in Sect. 2. In the second domain, there are three papers that describe techniques aiming to build foundations for new quality assurance tools. They are discussed in Sect. 3.

2 Tools for system developers

Various techniques to help developers to improve the quality of their systems are discussed: reducing system complexity

✉ Marieke Huisman
Marieke.Huisman@ewi.utwente.nl

Julia Rubin
mjulia@ece.ubc.ca

[1] University of Twente, Enschede, The Netherlands

[2] University of British Columbia, Vancouver, Canada

by splitting large CPS models into smaller ones, verification of model transformations in model-driven engineering, and automatic generation of workarounds to bypass system failures.

Tactical contract composition for hybrid system component verification: Müller et al. [5]

This paper presents an approach for verifying component-based models for cyber-physical systems (CPSs) with discrete and continuous dynamics. Examples for systems in the first category are autopilots in airplanes and controllers in self-driving cars, whereas examples of systems in the second category are motion of airplanes or cars. Componentization—splitting large models of CPSs into multiple smaller pieces with local responsibilities—reduces system complexity but introduces challenges for verification. That is because both local component behavior, e.g., decisions and motion of a robot, and inherently global phenomena e.g., time, co-occur, as components can interact virtually, e.g., robots communicate, and physically, e.g., a robot manipulates an object.

In order to verify each component in isolation, this paper introduces component contracts—input assumptions and output guarantees for each component—which emphasize the externally observable nature of component behavior. That is, the contracts specify the magnitude of change between two observations, e.g., current speed is at most twice the previously observed speed, and also capture the rate of change e.g., current speed is at most previous speed increased by accelerating for some time. The contracts abstract the hybrid, continuous-time behavior of one component to discrete-time observations available to other components. Given the definition of contracts, the paper proves that the safety of compatible components implies safety of the composed system, facilitating compositional verification of CPSs.

Slicing ATL model transformations for scalable deductive verification and fault localization: Cheng and Tisi [2]

This paper introduces an approach for verifying the correctness of model transformations, which are used in Model-Driven Engineering (MDE) for manipulating, evolving, and translating models, up to code generation. Correctness requirements for a model transformation are encoded as transformation contracts. A model transformation that does not satisfy the contract would generate faulty models, whose effect could be unpredictably propagated into subsequent MDE steps and compromise the reliability of the whole software development process.

The paper specifically focuses on ATL models. The relational nature of ATL is exploited to deduce static trace information, i.e., mappings among types of generated elements and rules that potentially generate these types. The trace information is then used to decompose the model transformation contract, and, for each sub-contract, slice the transformation to the only rules that may be responsible for fulfilling that sub-contract. Slicing makes it possible to devise a highly scalable technique for verifying large model transformations against a large number of contracts. To demonstrate industry-readiness, the technique is implemented as an extension to VeriATL verification system.

Automated workarounds from Java program specifications based on SAT solving: Uva et al. [7]

This paper presents an approach for automatically generating workarounds that bypass failures induced by bugs in Java programs. A workaround is an alternative routine that the system offers once the primary routine fails. Workarounds can be used in place of the failing routines, to circumvent failures.

Unlike existing approaches to workaround-based system recovery, which consider workarounds that are produced from user-provided abstract models or equivalent sequences of operations provided directly by the user, this paper computes workarounds from Java code equipped with formal specifications. Such approach makes it possible to consider the particular state where the failure occurred, leading to repairs that are more accurate and state-specific. The paper shows that the proposed technique is able to produce repairs in several cases where previous workaround-based approaches are inapplicable.

3 Techniques for enhancing quality assurance tools

Also various techniques are presented that extend and improve existing quality assurance tools, such as an improvement in model learning for probabilistic models, automated reasoning for graph-based modeling, and synthesis of program slicers from algebraic specifications.

Learning probabilistic models for model checking: an evolutionary approach and an empirical study—Wang et al. [8]

This paper focuses on the problem of obtaining a system model used for analysis by several automated techniques, such as model checking and model-based testing. System modeling is often done manually, which hinder the adoption of model-based system analysis and development techniques. To overcome this problem, existing approaches "learn" models based on sample system executions.

This paper is concerned with the question of how much one can generalize from the observed samples and how fast would learning converge. It proposes a new approach to control the degree of generalization and shows that the approach converges faster and learns models which are much smaller than those learned by existing approaches while providing better or similar analysis results. The paper also systematically analyzes existing learning approaches, showing that learning models for model checking might not be as effective as the estimation obtained by sampling many system executions within the same amount of time. Finally, the authors investigate how well current abstraction techniques for probabilistic model learning scale, showing that it can only work for a limited class of properties. The paper makes suggestions for future research on the topic.

Automated reasoning for attributed graph properties: Schneider et al. [6]

This paper focuses on attributed graphs: graphs that are equipped with attributes to express additional information about nodes and edges, such as names of entities or weights of relationships. Such graphs are often used in model-based engineering, in the formal analysis and verification of systems where the states are modeled as graphs, in the formal modeling and analysis of sets of semi-structured documents, especially if they are related by links, or of graph queries in the graph database domain.

The authors introduce a new logic for expressing properties on attributed graphs, which is expected to strengthening modeling capabilities and enable automated analysis. In this logic, graph and attribution parts are separated. The graph part is equivalent to first-order logic on graphs as introduced by Courcelle [3]. The attribution part is added to the graph part by reverting to the symbolic approach to graph attribution, where attributes are represented symbolically by variables whose possible values are specified by a set of constraints making use of algebraic specifications. The authors demonstrate the applicability of the proposed approach for graph database query validation.

Slicing from formal semantics: Asavoae et al. [1]

This paper describes a tool that synthesizes a program slicer from a given algebraic specification of a programming language operational semantics. The diversity of programming languages and the variety of definitions for program slices motivates the development of generic slicing techniques. The tool introduced in this work addresses this problem. More specifically, it takes as input a programming language semantics and synthesizes the necessary ingredients for program slicing, i.e., as data- and control-flow features of the languages. Then, it takes a program and, based on the flow features of the language, produces the program slice with respect to a slicing criterion. The authors show how the produced slicing tool can be used for validation of embedded system designs.

Acknowledgements We are grateful to all the authors for their contributions and to the program committee of FASE'17 for their help for selecting the conference program, including the papers for this issue. We are especially grateful to the FASE'17 program committee and external referees who reviewed the extended version of the papers that appear in this special issue.

References

1. Asavoae, I.M., Asavoae, M., Riesco, A.: Slicing from formal semantics: Chisel—a tool for generic program slicing. Int J Softw Tools Technol Transfer (2018). https://doi.org/10.1007/s10009-018-0500-y
2. Cheng, Z., Tisi, M.: Slicing ATL model transformations for scalable deductive verification and fault localization. Int J Softw Tools Technol Transfer (2018). https://doi.org/10.1007/s10009-018-0491-8
3. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Grzegorz, R. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, pp. 313–400. World Scientific Publishing Co., Inc, River Edge (1997)
4. Huisman, M., Rubin, J. (ed.): Fundamental Approaches to Software Engineering: 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, 22–29 April 2017, Proceedings, Volume 10202 of Lecture Notes in Computer Science. Springer, Berlin (2017)
5. Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: Tactical contract composition for hybrid system component verification. Int J Softw Tools Technol Transfer (2018). https://doi.org/10.1007/s10009-018-0502-9
6. Schneider, S., Lambers, L., Orejas, F.: Automated reasoning for attributed graph properties. Int J Softw Tools Technol Transfer (2018). https://doi.org/10.1007/s10009-018-0496-3
7. Uva, M., Ponzio, P., Regis, G., Aguirre, N., Frias, M.F.: Automated workarounds from Java program specifications based on SAT solving. Int J Softw Tools Technol Transfer (2018). https://doi.org/10.1007/s10009-018-0503-8
8. Wang, J., Sun, J., Yuan, Q., Pang., J.: Learning probabilistic models for model checking: an evolutionary approach and an empirical study. Int J Softw Tools Technol Transfer (2018). https://doi.org/10.1007/s10009-018-0492-7