Software Qual J (2017) 25:797–802
DOI 10.1007/s11219-017-9383-5

The central role of test automation in software quality assurance

Leonardo Mariani[1] · Dan Hao[2] ·
Rajesh Subramanyan[3] · Hong Zhu[4]

Published online: 10 July 2017

Welcome to this special section on the Automation of Software Test. It is inspired by the 2015 IEEE/ACM International Workshop on Automation of Software Test (AST 2015) held in Firenze, Italy, on 23 and 24 May 2015, in conjunction with the ICSE 2015 conference. The authors of selected papers presented at AST 2015 were invited to submit extended and revised versions of their corresponding workshop papers. At the same time, an open call-for-papers was announced publicly and distributed in the software engineering community. Eighteen papers were submitted. Each paper was reviewed by active researchers and experts in the related subject areas. The review process has followed the journal's review protocol and applied the journal's quality standards. Eight papers were finally accepted for inclusion.

AST 2015 was the 10th consecutive AST workshop at the ICSE conference. In the past few years, the workshops have attracted large numbers of high-quality papers and large numbers of delegates participating in the active exchanges of research results and practical experiences and lively charrette discussions on selected special themes of test automation. As a result, we have previously published six journal special issues/sections containing extended and revised

✉ Hong Zhu
hzhu@brookes.ac.uk

Leonardo Mariani
mariani@disco.unimib.it

Dan Hao
haodan@pku.edu.cn

Rajesh Subramanyan
rajesh.subramanyan@siemens.com

1 University of Milano Bicocca, Viale Sarca 336, 20149 Milan, Italy

2 Peking University, Room 1431, Science Building #1, Peking University, Beijing 100871, China

3 Siemens Healthcare Diagnostics, 1584 Enterprise Blvd, W, Sacramento, CA 95691, USA

4 Oxford Brookes University, Oxford OX33 1HX, UK

versions of selected papers in prestigious journals in the subject area of software engineering, which are the following:

(1) The Computer Journal (Vol.52, No.5, Aug. 2009)
(2) Information and Software Technology (Vol.51, No.11, Nov. 2009)
(3) IET Software (Vol. 5, No. 2, Apr. 2011)
(4) Software Quality Journal (Vol.19, No.4, Dec. 2011).
(5) Journal of Systems and Software (Vol. 86, No. 8, Aug. 2013)
(6) Software Quality Journal (Vol. 22, No. 2, June 2014)

This section is the 7th consecutive journal special issue/section. Over the past 10 years since the first AST workshop was held at ICSE 2006 in Shanghai, China, software test automation has developed significantly from merely a means of cost reduction and a tool for efficiency improvement to an essential ingredient of software quality assurance and now plays a central role in software development methodology. With the rapid growth of the acceptance of agile software development in industry, test automation has become a common practice in software development. The philosophical principle of “test driven” processes in agile methodologies and more generally the “shift-to-the-left” principle of DevOps have put test automation at the heart of software development processes. Many test automation techniques and tools have gone out of laboratories and entered the battlefield of software development. Test automation has become indispensable to modern software development. The practice of software test automation is also reflected in the research on the subject through the fast growth of the research community and the increasing scope of research topics.

The collection of papers included in this special section vividly demonstrates this trend. It covers both the application of test automation to complicated computer application domains and theoretical and empirical studies of test automation methods, techniques, and their foundations.

One of the grand challenges for adequately testing complex software is the oracle problem, which is to decide the pass or fail outcome of a test execution. It is still primarily an expensive and error-prone manual activity in practice. This special section contains two papers addressing this problem. In the paper entitled *Separating Passing and Failing Test Executions by Clustering Anomalies*, Rafig Almaghairbe and Marc Roper present an approach to automatically detect passing and failing executions using cluster-based anomaly detection on dynamic execution data based on, firstly, just a system’s input/output pairs, and secondly, amalgamations of input/output pairs and execution traces. The key hypothesis underlying the approach is that failures will group into small clusters whereas passing executions will group into larger ones. They report an evaluation of the validity of this hypothesis on three systems with a range of faults. In particular, they demonstrate that in many cases small clusters were composed of at least 60% failures and often more. Concentrating on the failures in these small clusters can substantially reduce the numbers of outputs that a tester would need to manually examine following a test run. Thus, the approach has the potential to improve the effectiveness and efficiency of the testing process. In the paper entitled *Application of Metamorphic Testing Monitored by Test Adequacy in a Monte Carlo Simulation Program*, Junhua Ding and Xin-Hua Hu study a different approach to the test oracle problem, the metamorphic testing (MT) method. MT alleviates the oracle problem by using one or multiple metamorphic relations (MRs) to automatically check test output correctness. An MR specifies the relationships among the system’s outputs on a set of related tests instead of the correctness of individual test outputs. MRs are more expressive and more flexible to use than other software specification formalisms, but it is difficult to develop good MRs and evaluate their adequacy. In this paper, Ding and Hu propose a framework for evaluating MRs by measuring code coverage and mutation score, and more importantly, using the evaluation results as feedback in an iterative process of developing MRs. They report a case study of the approach by testing real-world complex scientific computing software: a Monte Carlo modeling program that simulates photon propagations in turbid tissue phantoms for accurate and efficient generation of reflectance images from biological tissues. Their data show the effectiveness and applicability of the proposed technique.

Mutation analysis is one of the primary techniques for evaluating the quality of test cases based on the theory of fault-based testing. It is widely used in empirical studies of testing methods and techniques. A number of mutation testing tools are now available for various programming languages. In the paper entitled *Does Choice of Mutation Tool Matter?*, Rahul Gopinath, Ahmed Iftekhar, Amin Alipour, Carlos Jensen, and Alex Groce investigate mutation testing tools by evaluating the efficacy of mutants produced by different tools. They find that mutation tools rarely agree with each other on various measures of the efficacy of mutants that the tools generate. The measures include the difficulty of detection, strength of minimal sets, the diversity of mutants, and the information carried by the mutants. The disagreement between scores can be large, where the characteristics of the project form a significant factor of the variation. However, they also find that the mean difference between tools is very small indicating that no single tool consistently skews mutation scores as high or low for all projects. They suggest that experiments yielding small differences in mutation score, especially using a single tool or a small number of projects, may not be reliable. They point out that there is a need for standardization of mutation analysis tools.

Testability is one of the main factors that affect test effectiveness and test cost, but it has not been studied as intensively as test case generation and test adequacy. In this special section, we are pleased to be able to include two research papers on this topic. In the paper entitled *A Large Scale Study of Call Graph-based Impact Prediction using Mutation Testing*, Vincenzo Musco, Martin Monperrus, and Philippe Preux propose a method for evaluating impact propagation analysis techniques. Impact propagation analysis predicts how a change in a program will affect other elements in the program. It is often applied to measure testability and optimize testing effort. They applied the proposed technique to analyze impact prediction based on four types of call graphs in an empirical study. Their results show that the completeness of impact prediction increases as the graph sophistication increases. However, surprisingly, the most basic call graph gives the best trade-off between precision and recall for impact prediction. In the paper entitled *An Empirical Study on the Effects of Code Visibility on Program Testability*, Lei Ma, Cheng Zhang, Bing Yu, and Hiroyuki Sato report an empirical study on the relationship between code visibility and testability. They observed that for manual testing, code visibility does not necessarily affect test code coverage and fault detection rate. However, for automated testing using testing tools, low code visibility often leads to low code coverage and poor fault detection rate.

The existence of good models of the software under test is a precondition for model-based software testing, which has been an active research area since the inception of the model-driven software development methodology. In the paper entitled *Automated Refinement of Models for Model-Based Testing Using Exploratory Testing*, Ceren Sahin Gebizli and Hasan Sozer introduce an approach and a toolset called ARME for automatically refining system models based on recorded manual testing activities performed by test engineers. ARME compares the recorded execution traces of manual tests with respect to the possible execution paths in test models. Then, these models are automatically refined to incorporate any omitted system behavior and to update model parameters. The refined models are then used for generating more test cases. They report case studies on three industrial systems and demonstrate that, by using the refined model, some critical faults that are undetected by using the original model and missed during the manual exploratory testing activities can be detected.

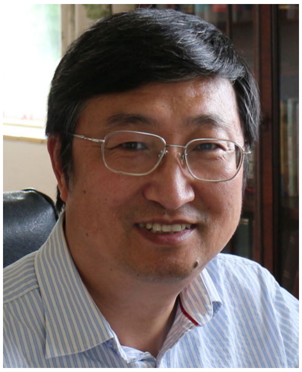
Test automation is now applied to more and more complicated software systems in practice. This special section contains two papers that address the complexity of software systems in two different application domains. In the paper entitled *APOGEN: Automatic Page Object Generator for Web Testing*, Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella are concerned with modern web-based applications, which are characterized by their rapid evolution and mostly developed following an agile methodology. This makes traditional automated test suites difficult to maintain. A solution to this problem is to adopt the Page Object pattern in test scripts. Page objects are facade classes abstracting the internals of web pages into high-level business functions that can be invoked by the test cases. They decouple test code from web page details and make web test cases more readable and maintainable. However, the manual development of such page objects requires substantial coding effort. Stocco et al. propose a novel approach for the automatic generation of page objects and implemented a tool called Apogen. Apogen automatically derives a testing model by reverse engineering the web application under test. It combines clustering and static analysis to identify meaningful page abstractions, which are then automatically transformed into Java page objects for the Selenium WebDriver. The paper also reports an evaluation on an open-source web application. Their data show that automatically generated page object methods cover most of the application functionalities, and result in readable and meaningful code. In the paper entitled *Automatic Generation of Test System Instances for Configurable Cyber-Physical Systems*, Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria, and Justyna Zander are concerned with another kind of highly complicated system, a Cyber-Physical System (CPS), which is a ubiquitous system that integrates digital technologies with physical processes. Such systems are becoming increasingly configurable in order to meet a wide range of user requirements. Testing a CPS in various configurations imposes a new challenge due to the large number and complexity of variants. Arrieta et al. propose a methodology supported by a tool called ASTERYSCO that automatically generates simulation-based system instances to test individual configurations of CPSs. The method and the tool are evaluated by a case study with a configurable unmanned aerial vehicle. It demonstrates that they can generate variants of the configurable system for testing more systematically and efficiently than a manual method.

The collection of papers in this journal special section clearly shows that software test automation binds very tightly to software development methodologies, especially agile methodologies. It plays a central role in such methodologies. It is no longer just focused on test generation, but also covers all phases of software development and all aspects of software quality assurance activities, including software requirements and specification (such as software modeling and model-driven testing, metamorphic testing, and formal specification-based testing), architectural design (such as the testing of microservice architecture), code design (such as for improving testability), etc. As one of the most active topics in test automation research, test generation has also widened its scope. It is not just the generation of test cases and test data, but also the generation of other artifacts of testing, such as system instances in configurable CPS applications. The subject area of software test automation is more active than ever before.

Finally, we, the guest editors of this journal special section, would like to express our appreciation of the authors' contributions to the AST workshop and thus special section. We are also grateful to the PC members of the AST workshop and reviewers of the journal papers for their excellent work in the review of the submissions. Their constructive and detailed comments on the papers are invaluable to the authors in their revision of the papers. Their recommendations are the basis of our decisions for the selection of the papers among the submissions. Without their hard work, this special section would not be possible. Thank you!

Leonardo Mariani received the PhD degree in computer science from the University of Milano Bicocca, in 2005. He is an associate professor with the University of Milano Bicocca. His research interests include software engineering, and in particular software testing, static and dynamic program analysis, automated debugging, and self-healing systems. He received the ERC Consolidator Grant in 2015 and he is currently active in several European and National projects. He is regularly involved in the organizing and program committees of major software engineering conferences.

Dan Hao received the PhD degree from Peking University in 2008. She is an associate professor of Peking University. Her research interests include software engineering, and in particular software testing, debugging, and maintenance. She received the national science fund for excellent young researchers in 2015 and won the national young ”yangzhe” program. She is regularly involved in the organizing and program committees of major software engineering conferences.



Hong Zhu received his BSc, MSc and PhD degrees in Computer Science from Nanjing University, China, in 1982, 1984 and 1987, respectively. He is a professor of computer science at Oxford Brookes University, UK, and chairs the Applied Formal Methods Research Group of the Department of Computing and Communication Technologies. His research interests are in the area of software development methodologies, including formal methods, agent-orientation, automated software development, foundation of software engineering, software design, modelling and testing methods, etc.