Advances in Unit Testing: Theory and Practice

Tao Xie
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
taoxie@illinois.edu

Nikolai Tillmann
Microsoft
Tools for Software Engineers
Redmond WA 98052, USA
nikolait@microsoft.com

Pratap Lakshman
Microsoft
Developer Division
Hyderabad, India
pratapl@microsoft.com

ABSTRACT

Parameterized unit testing, recent advances in unit testing, is a new methodology extending the previous industry practice based on traditional unit tests without parameters. A parameterized unit test (PUT) is simply a test method that takes parameters, calls the code under test, and states assertions. Parameterized unit testing allows the separation of two testing concerns or tasks: the specification of external, black-box behavior (i.e., assertions or specifications) by developers and the generation and selection of internal, white-box test inputs (i.e., high-code-covering test inputs) by tools. PUTs have been supported by various testing frameworks. Various open source and industrial testing tools also exist to generate test inputs for PUTs. This technical briefing presents latest research on principles and techniques, as well as practical considerations to apply parameterized unit testing on real-world programs, highlighting success stories, research and education achievements, and future research directions in developer testing.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Parameterized Unit Testing, Test Generation, Test Oracles

1. TECHNICAL BRIEFING'S TOPIC

Unit testing has been widely recognized as an important and valuable means of improving software reliability, as it exposes bugs early in the software development life cycle. However, manual unit testing is often tedious and insufficient. Testing tools can be used to enable economical use of resources by reducing manual effort. Parameterized unit testing [9, 4] is a new methodology extending the current industry practice based on closed, traditional unit tests (i.e., test methods without input parameters). A parameterized unit test (PUT) is simply a test method that takes parameters, calls the code under test, and states assertions. Test methods are generalized by allowing parameters. This generalization serves two main purposes. First, parameterized test methods are specifications of the behavior of the methods under test: they not only provide exemplary arguments to the methods under test, but ranges of such arguments. Second, parameterized unit tests describe a set of traditional unit tests that can be obtained by instantiating the parameterized test methods with given argument sets. Parameterized unit testing allows the separation of two testing concerns or tasks: the specification of external, black-box behavior (i.e., assertions or specifications) by developers and the generation and selection of internal, white-box test inputs (i.e., high-code-covering test inputs) by tools. PUTs have been supported by various testing frameworks. Various open source and industrial testing tools also exist to generate test inputs for PUTs.

This technical briefing presents latest research on principles and techniques, as well as practical considerations to apply parameterized unit testing on real-world programs, highlighting success stories, research and education achievements, and future research directions in developer testing. The technical briefing aims to help improve developer skills and knowledge for writing PUTs and give overview of tool automation in supporting PUTs. Attendees can acquire the skills and knowledge needed to perform research or conduct practice in the field of developer testing and to integrate developer testing techniques in their own research, practice, and education.

2. TECHNICAL BRIEFING'S INTEREST TO COMMUNITY

With recent advances in test generation research such as dynamic symbolic execution [3, 5], powerful test generation tools are now at the fingertips of developers in software industry. For example, Microsoft Research Pex [7, 8], a state-of-the-art tool based on dynamic symbolic execution, has been shipped as *IntelliTest* in Visual Studio 2015, benefiting numerous developers in software industry. Such test generation tools allow developers to automatically generate test inputs for the code under test, comprehensively covering various program behaviors to achieve high code coverage. These tools help alleviate the burden of extensive manual software testing, especially on test generation.

Although such tools provide powerful support for automatic test generation, by default only a predefined limited set of properties can be checked, serving as test oracles for

ICSE '16 May 14-22, 2016, Austin, TX, USA

ACM ISBN 978-1-4503-4205-6/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2889160.2891056

these automatically generated test inputs. Violating these predefined properties leads to various runtime failures, such as null dereferencing or division by zero. Despite being valuable, these predefined properties are *weak test oracles*, which do not aim for checking functional correctness but focus on robustness of the code under test. Parameterized unit tests serve as *strong test oracles* to complement these powerful test generation tools in research and practice.

The technical briefing is targeted at three groups of audience, which are broad in the software engineering community.

- **Software practitioners.** Practitioners can learn the state of the art in conducting developer testing. The technical briefing aims to demonstrate various practical applications of methodology, techniques, and tools for parameterized unit testing, so that practitioners could apply the learned skills and knowledge to conduct effective developer testing in their industrial setting environments, such as effectively cooperating with test-generation tools [11, 13]. Software developers and practitioners are encouraged to bring their own problems to the discussion after the technical briefing so that both researchers and software practitioners can share their experience, exchange ideas, and collaborate on industry relevant research.
- **Testing researchers.** Both software testing researchers in general and those working on specific subareas on developer testing can find the technical briefing informative and inspiring. After attending this technical briefing, the testing researchers can gain good understanding of research problems in developer-testing practice, research methodology in helping write PUTs [6, 2], and test generation techniques for PUTs [10]. Researchers can acquire the knowledge needed in order to apply advanced test generation techniques such as dynamic symbolic execution to assist developer testing.
- **Testing educators.** Software testing educators or software engineering educators in general can find the technical briefing helpful in improving their skills in teaching developer testing [12]. After attending this technical briefing, the educators can learn ways of teaching and training students or practitioners in writing PUTs. Educators can also learn the first-hand experience of teaching students in writing PUTs and learn how to leverage the teaching materials at the wiki site for teaching PUT/Pex [1].

Acknowledgment

This material is based upon work supported by the Maryland Procurement Office under Contract No. H98230-14-C-0141. Tao Xie's work is also supported in part by NSF grants CCF-1349666, CCF-1409423, CNS-1434582, CCF-1434596, and CNS-1513939, and a Google Faculty Research Award.

3. REFERENCES

[1] Wiki site for teaching/learning parameterized unit testing/Pex, 2016. https://sites.google.com/site/teachpex/.

[2] M. Boshernitsan, R. Doong, and A. Savoia. From Daikon to Agitator: Lessons and challenges in building a commercial tool for developer testing. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis (ISSTA 2006)*, pages 169–180, 2006.

[3] P. Godefroid, N. Klarlund, and K. Sen. DART: Directed automated random testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005)*, pages 213–223, 2005.

[4] D. Saff, M. Boshernitsan, and M. D. Ernst. Theories in practice: Easy-to-write specifications that catch bugs. Technical Report MIT-CSAIL-TR-2008-002, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, January 14, 2008.

[5] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*, pages 263–272, 2005.

[6] S. Thummalapenta, M. R. Marri, T. Xie, N. Tillmann, and J. de Halleux. Retrofitting unit tests for parameterized unit testing. In *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering (FASE 2011)*, pages 294–309, 2011.

[7] N. Tillmann and J. De Halleux. Pex: White box test generation for .NET. In *Proceedings of the 2nd International Conference on Tests and Proofs (TAP 2008)*, pages 134–153, 2008.

[8] N. Tillmann, J. de Halleux, and T. Xie. Transferring an automated test generation tool to practice: From Pex to Fakes and Code Digger. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE 2014)*, pages 385–396, 2014.

[9] N. Tillmann and W. Schulte. Parameterized unit tests. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*, pages 253–262, 2005.

[10] X. Xiao, S. Thummalapenta, and T. Xie. Advances on improving automation in developer testing. In *Advances in Computers*, volume 85, pages 165–212. 2012.

[11] X. Xiao, T. Xie, N. Tillmann, and J. de Halleux. Precise identification of problems for structural test generation. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*, pages 611–620, 2011.

[12] T. Xie, J. de Halleux, N. Tillmann, and W. Schulte. Teaching and training developer-testing techniques and tool support. In *Proceedings of the 25th Annual ACM Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH 2010), Educators' and Trainers' Symposium*, pages 175–182, 2010.

[13] T. Xie, L. Zhang, X. Xiao, Y. Xiong, and D. Hao. Cooperative software testing and analysis: Advances and challenges. *Journal of Computer Science and Technology*, 29(4):713–723, 2014.