New Strategies for Automated Random Testing

Mian Asbat Ahmad

Department of Computer Science

The University of York

A thesis submitted for the degree of

Doctor of Philosophy

August 23, 2013

Abstract

This is where you write your abstract ...

Contents

List of Figures

List of Tables

.

Acknowledgements

Several people have contributed to the completion of my PhD dissertation. However, the most prominent personality deserving due recognition is my worthy supervisor, Dr. Manuel Oriol. Thank you Manuel for your endless help, valuable guidance, constant encouragement, precious advice, sincere and affectionate attitude.

I thank my assessor Prof. John Clark for his constructive feedback on my various reports and presentations. I am also thankful and highly indebted to Prof. Richard Paige for his generous help, cooperation and guidance during my research at the University of York.

Special thanks to my father Prof. Mushtaq A. Mian who provided a conducive environment, valuable guidance and crucial support at all levels of my educational career and my very beloved mother whose love, affection and prayers have been my most precious assets. Also I am thankful to my elder brothers Dr. Ashfaq, Dr. Aftab, Dr. Ishaq, Dr. Afaq and my sister Dr. Haleema who have been the source of inspiration for me to pursue higher studies. My immediate younger brother Dr. Ilyas and my younger sister Ayesha studying in the UK, deserve recognition for their help, well wishes and moral support. Last but not the least I am very thankful to my dear wife Dr. Munazza for her company, help and cooperation throughout my stay at York.

I was funded by Departmental Overseas Research Scholarship (DORS), a financial support awarded to overseas students on the basis of outstanding academic ability and research potential. I am truly grateful to the Department of Computer Science for financial support that allowed me to concentrate on my research.

I feel it a great honour to dedicate my PhD thesis to my beloved parents for their significant contribution in achieving the goal of academic excellence.

Chapter 1

Introduction

In this chapter we give a brief introduction and motivation for the research work presented in this thesis. After brief motivation, we commence by introducing the problems in random testing. We then describe the alternative approaches to overcome these problems, followed by our research goals and contributions. At the end of the chapter, we give the structure of the thesis.

1.1 Motivation

Software is everywhere. In our world today, software flies spacecraft, monitors power plants, manages stock exchange, assist surgeries, drives cars and design graphics. The margin for error in these mission-critical and safety-critical systems is so small that a minor fault can incur huge cost to the economy and miseries to the mankind [45]. Therefore, software development companies leave no stone unturned to ensure the reliability and accuracy of the software. This dissertation is a step further towards the reduction in overall cost of software testing by devising new improved and highly effective software testing techniques.

1.2 The Problems

In software testing, one is often confronted with the problem of selecting a test data set, from a large or often infinite domain, as exhaustive testing is not always applicable. Test data set is a subset of domain carefully selected to test the given software. Finding an adequate test data set is a crucial process in any testing technique as it aims to represent the whole domain and evaluate the given system under test (SUT) for structural or functional properties [57], [44]. Manual test data set generation is a time-consuming and laborious exercise [49]; therefore, automated

test data set generation is always preferred. Test data generators are classified in to Pathwise, Goal-Oriented, Intelligent and Random [89]. Random test data generation generates test data set randomly from the whole domain. Unlike other approaches Random approach is simple, widely applicable, easy to implement in an automatic testing tool, fastest in computation, no overhead in choosing inputs and free from bias [25].

Despite the benefits random testing offers, its simplistic and non-systematic nature expose it to high criticism [88]. Myers & Sandler [59] mentioned it as "Probably the poorest methodology of all is random-input testing...". Where this statement is based on intuition and lacks any experimental evidence, it motivated the interest of research community to evaluate and improve random testing. Adaptive random testing [13], Restricted Random Testing [10], Feedback directed Random Testing [75], Mirror Adaptive Random Testing [14] and Quasi Random Testing [18] are few of the enhanced random testing techniques aiming to increase its fault finding ability.

Random testing is also considered weak in providing high code coverage [63], [30]. For example, in random testing when the conditional statement "*if (x == 25) then ...* " is exposed to execution then there is only one chance, of the "*then...*" part of the statement, to be executed out of 2^{32}. If x is an integer variable of 32 bit value [40].

Random testing is no exception when it comes to the complexity of understanding and evaluating test results. Modern testing techniques simplifies results by truncating the lengthy log files and display only the fault revealing test cases in the form of unit tests. However efforts are required to show the test results in more compact and user-friendly way.

1.3 Research Goals

The overall goal of this thesis is to develop new techniques for automated testing based on random strategy that addresses the above-mentioned problems. Particularly,

1. We aim to develop an automated random testing technique that is able to generate more fault-revealing test data. To achieve this we exploit the presence of fault clusters found in the form of block and strip fault domains inside the input domain of a given SUT. Thus we are able to find equal number of faults in fewer numbers of test cases than other random strategies.

2. We aim to develop a novel framework for finding the faults, their domains and the presentation of obtained results on a graphical chart inside the specified lower and upper bound. It considers the correlations of the fault and fault domain. It also gives a simplified and user-friendly report to easily identify the faulty regions across the whole domain.

3. We aim to develop another automated testing technique which focuses on increase in code coverage and generation of more fault-revealing data. To achieve this we utilises Daikon— an automated invariant detector that reports likely program invariant. An invariant is a property that holds at certain point or points in a program. With these invariants in hand we can restrict the random strategy to generate values around these critical points. Thus we are able to increase the code coverage and quick identification of faults.

1.4 Contributions

To achieve the research goals described in Section 1.3, we make the following specific contributions:

1.4.1 Dirt Spot Sweeping Random Strategy

Random testing is a simple and effective technique to find failures in complex programs. However, its efficiency reduces when the failures lie in contiguous locations across the input domain. To overcome the deficiency, we developed a new automated technique: Dirt Spot Sweeping Random (DSSR) strategy. It is based on the assumption that unique failures reside in contiguous blocks and stripes. When a failure is identified, the DSSR strategy selects neighbouring values for the subsequent tests. Resultantly, selected values sweep around the failure leading to the discovery of new failures in the vicinity. To evaluate the effectiveness of DSSR strategy a total of 60 classes (35,785 lines of code), each class with 30 x $\mathbf{10^5}$ calls, were tested by Random (R), Random+ (R+) and DSSR strategies. T-Test analysis showed significantly better performance of DSSR compared to R strategy in 17 classes and R+ strategy in 9 classes. In the remaining classes all the three strategies performed equally well. Numerically, the DSSR strategy found 43 and 12 more unique failures than R and R+ strategies respectively. This study comprehends that DSSR strategy will have a profound positive impact on the failure-finding ability of R and R+ testing.

1.4.2 Automated Discovery of Failure Domain

There are several automated random strategies of software testing based on the presence of point, block and strip fault domains inside the whole input domain. As yet no particular, fully automated test strategy has been developed for the discovery and evaluation of the fault domains. We therefore have developed Automated Discovery of Failure Domain, a new random test strategy that finds the faults and the fault domains in a given system under test. It further provides visualisation of the identified pass and fail domain. In this paper we describe ADFD

strategy, its implementation in YETI and illustrate its working with the help of an example. We report on experiments in which we tested error seeded one and two-dimensional numerical programs. Our experimental results show that for each SUT, ADFD strategy successfully performs identification of faults, fault domains and their representation on graphical chart.

1.4.3 Invariant Guided Random+ Strategy

Acknowledgement of random testing being simple in implementation, quick in test case generation and free from any bias, motivated research community to do more for increase in performance, particularly, in code coverage and fault-finding ability. One such effort is Random+ — Ordinary random testing technique with addition of interesting values (border values) of high preference. We took a step further and developed Invariant Guided Random+ Strategy (IGRS). IGRS is an extended form of Random+ strategy guided by software invariants. Invariants from the given software under test are collected by Daikon— an automated invariant detector that reports likely invariant, prior to testing and added to the list of interesting values with high preference. The strategy generates more values around these critical program values. Experimental result shows that IGRS not only increase the code coverage but also find some subtle errors that pure Random and Random+ were either unable or may take a long time to find.

1.5 Structure of the Thesis

The rest of the thesis is organised as follows: In Chapter 2, we give a thorough review of the relevant literature. We commence by discussing a brief introduction of software testing and shed light on various techniques and types of software testing. Then, we extend our attention to automated random testing and the testing tools using random technique to test softwares. In Chapter 3, we present our first automated random strategy Dirt Spot Sweeping Random (DSSR) strategy based on sweeping faults from the clusters in the input domain. Chapter 4 describes our second automated random strategy which focus on dynamically finding the fault with their domains and its graphical representation. Chapter 5 presents the third strategy that focus on quick identification of faults and increase in coverage with the help of literals; Finally, in Chapter 7, we summarise the contributions of this thesis, discuss the weaknesses in the work, and suggest avenues for future work.

Appdx A

and here I put a bit of postamble ...

Appdx B

and here I put some more postamble ...

References

[1] W Richards Adrion, Martha A Branstad, and John C Cherniavsky. Validation, verification, and testing of computer software. *ACM Computing Surveys (CSUR)*, 14(2):159–192, 1982.

[2] H. Agrawal, J.R. Horgan, S. London, and W.E. Wong. Fault localization using execution slices and dataflow tests. In *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pages 143 –151, oct 1995.

[3] NY. American National Standards Institute. New York, Institute of Electrical, and Electronics Engineers. *Software Engineering Standards: ANSI/IEEE Std 729-1983, Glossary of Software Engineering Terminology ...* Inst. of Electrical and Electronics Engineers, 1984.

[4] Andrea Arcuri, Muhammad Zohaib Iqbal, and Lionel Briand. Random testing: Theoretical results and practical implications. *IEEE Transactions on Software Engineering*, 38:258–277, 2012.

[5] Luciano Baresi and Michal Young. Test oracles. *Techn. Report CISTR-01*, 2, 2001.

[6] Boris Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[7] Boris Beizer. *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., 1995.

[8] Chandrasekhar Boyapati, Sarfraz Khurshid, and Darko Marinov. Korat: automated testing based on java predicates. In *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis*, pages 123–133, New York, NY, USA, 2002. ACM.

[9] F.T. Chan, T.Y. Chen, I.K. Mak, and Y.T. Yu. Proportional sampling strategy: guidelines for software testing practitioners. *Information and Software Technology*, 38(12):775 – 782, 1996.

[10] Kwok Ping Chan, Tsong Yueh Chen, and Dave Towey. Restricted random testing. In *Proceedings of the 7th International Conference on Software Quality*, ECSQ '02, pages 321–330, London, UK, UK, 2002. Springer-Verlag. 2

[11] Kwok Ping Chan, Tsong Yueh Chen, and Dave Towey. Normalized restricted random testing. In *Reliable Software TechnologiesAda-Europe 2003*, pages 368–381. Springer, 2003.

[12] Juei Chang and Debra J. Richardson. Structural specification-based testing: automated support and experimental evaluation. *SIGSOFT Softw. Eng. Notes*, 24(6):285–302, 1999.

[13] T. Y. Chen. Adaptive random testing. *Eighth International Conference on Qualify Software*, 0:443, 2008. 2

[14] T. Y. Chen, F. C. Kuo, R. G. Merkel, and S. P. Ng. Mirror adaptive random testing. In *Proceedings of the Third International Conference on Quality Software*, QSIC '03, page 4, Washington, DC, USA, 2003. IEEE Computer Society. 2

[15] Tsong Yueh Chen, De Hao Huang, F-C Kuo, Robert G Merkel, and Johannes Mayer. Enhanced lattice-based adaptive random testing. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 422–429. ACM, 2009.

[16] Tsong Yueh Chen, Fei-Ching Kuo, and R. Merkel. On the statistical properties of the f-measure. In *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on*, pages 146 – 153, sept. 2004.

[17] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T. H. Tse. Adaptive random testing: The art of test case diversity. *J. Syst. Softw.*, 83:60–66, January 2010.

[18] Tsong Yueh Chen and Robert Merkel. Quasi-random testing. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 309–312, New York, NY, USA, 2005. ACM. 2

[19] T.Y. Chen, R. Merkel, P.K. Wong, and G. Eddy. Adaptive random testing through dynamic partitioning. In *Quality Software, 2004. QSIC 2004. Proceedings. Fourth International Conference on*, pages 79 – 86, sept. 2004.

[20] T.Y. Chen and Y.T. Yu. On the relationship between partition and random testing. *Software Engineering, IEEE Transactions on*, 20(12):977 –980, dec 1994.

[21] T.Y. Chen and Y.T. Yu. On the expected number of failures detected by subdomain testing and random testing. *Software Engineering, IEEE Transactions on*, 22(2):109 –119, feb 1996.

[22] John Joseph Chilenski and Steven P Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, 1994.

[23] I Ciupa, A Pretschner, M Oriol, A Leitner, and B Meyer. On the number and nature of faults found by random testing. *Software Testing Verification and Reliability*, 9999(9999):1–7, 2009.

[24] Ilinca Ciupa, Andreas Leitner, Manuel Oriol, and Bertrand Meyer. Object distance and its application to adaptive random testing of object-oriented programs. In *Proceedings of the 1st international workshop on Random testing*, RT '06, pages 55–63, New York, NY, USA, 2006. ACM.

[25] Ilinca Ciupa, Andreas Leitner, Manuel Oriol, and Bertrand Meyer. Experimental assessment of random testing for object-oriented software. In *Proceedings of the 2007 international symposium on Software testing and analysis*, ISSTA '07, pages 84–94, New York, NY, USA, 2007. ACM. 2

[26] Ilinca Ciupa, Andreas Leitner, Manuel Oriol, and Bertrand Meyer. Artoo: adaptive random testing for object-oriented software. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 71–80, New York, NY, USA, 2008. ACM.

[27] Ilinca Ciupa, Bertrand Meyer, Manuel Oriol, and Alexander Pretschner. Finding faults: Manual testing vs. random+ testing vs. user reports. In *Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*, pages 157–166, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Ilinca Ciupa, Alexander Pretschner, Andreas Leitner, Manuel Oriol, and Bertrand Meyer. On the predictability of random tests for object-oriented software. In *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*, pages 72–81, Washington, DC, USA, 2008. IEEE Computer Society.

[29] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, ICFP '00, pages 268–279, New York, NY, USA, 2000. ACM.

[30] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The aetg system: An approach to testing based on combinatorial design. *Software Engineering, IEEE Transactions on*, 23(7):437–444, 1997. 2

[31] Julie Cohen, Daniel Plakosh, and Kristi L Keeler. Robustness testing of software-intensive systems: Explanation and guide. 2005.

[32] Christoph Csallner and Yannis Smaragdakis. Jcrasher: An automatic robustness tester for Java. *Software—Practice & Experience*, 34(11):1025–1050, September 2004.

[33] Edsger W. Dijkstra. Structured programming. chapter Chapter I: Notes on structured programming, pages 1–82. Academic Press Ltd., London, UK, UK, 1972.

[34] Joe W. Duran and Simeon Ntafos. A report on random testing. In *Proceedings of the 5th international conference on Software engineering*, ICSE '81, pages 179–183, Piscataway, NJ, USA, 1981. IEEE Press.

[35] Joe W. Duran and Simeon C. Ntafos. An evaluation of random testing. *Software Engineering, IEEE Transactions on*, SE-10(4):438 –444, july 1984.

[36] Richard E Fairley. Tutorial: Static analysis and dynamic testing of computer software. *Computer*, 11(4):14–23, 1978.

[37] Justin E. Forrester and Barton P. Miller. An empirical study of the robustness of windows nt applications using random testing. In *Proceedings of the 4th conference on USENIX Windows Systems Symposium - Volume 4*, WSS'00, pages 6–6, Berkeley, CA, USA, 2000. USENIX Association.

[38] Marie-Claude Gaudel. Software testing based on formal specification. In *Testing Techniques in Software Engineering*, pages 215–242. Springer, 2010.

[39] D. Gilbert. *The JFreeChart class library version 1.0.9: Developer's guide*. Refinery Limited, Hertfordshire, 2008.

[40] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *ACM Sigplan Notices*, volume 40, pages 213–223. ACM, 2005. 2

[41] W.J. Gutjahr. Partition testing vs. random testing: the influence of uncertainty. *Software Engineering, IEEE Transactions on*, 25(5):661 –674, sep/oct 1999.

[42] D. Hamlet and R. Taylor. Partition testing does not inspire confidence [program testing]. *Software Engineering, IEEE Transactions on*, 16(12):1402 –1411, dec 1990.

[43] Richard Hamlet. Random testing. *Encyclopedia of software Engineering*, 1994.

[44] William E Howden. A functional approach to program testing and analysis. *Software Engineering, IEEE Transactions on*, (10):997–1005, 1986. 1

[45] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, Der-Tsai Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th international conference on World Wide Web*, pages 40–52. ACM, 2004. 1

[46] Paul Hudak, John Hughes, Simon Peyton Jones, and Philip Wadler. A history of haskell: being lazy with class. In *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 12–1–12–55, New York, NY, USA, 2007. ACM.

[47] James A. Jones, Mary Jean Harrold, and John Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 467–477, New York, NY, USA, 2002. ACM.

[48] Sarfraz Khurshid and Darko Marinov. TestEra: Specification-Based testing of java programs using SAT. *Automated Software Engineering*, 11:403–434, 2004. 10.1023/B:AUSE.0000038938.10589.b9.

[49] Bogdan Korel. Automated software test data generation. *Software Engineering, IEEE Transactions on*, 16(8):870–879, 1990. 1

[50] Andreas Leitner, Ilinca Ciupa, Bertrand Meyer, and Mark Howard. Reconciling manual and automated testing: The autotest experience. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, HICSS '07, pages 261a–, Washington, DC, USA, 2007. IEEE Computer Society.

[51] Andreas Leitner, Manuel Oriol, Andreas Zeller, Ilinca Ciupa, and Bertrand Meyer. Efficient unit test case minimization. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 417–420. ACM, 2007.

[52] Andreas Leitner, Alexander Pretschner, Stefan Mori, Bertrand Meyer, and Manuel Oriol. On the effectiveness of test extraction without overhead. In *Proceedings of the 2009 International Conference on Software Testing Verification and Validation*, pages 416–425, Washington, DC, USA, 2009. IEEE Computer Society.

[53] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. 10(8):707–710, 1966.

[54] Richard C. Linger. Cleanroom software engineering for zero-defect software. In *Proceedings of the 15th international conference on Software Engineering*, ICSE '93, pages 2–13, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[55] Huai Liu, Fei-Ching Kuo, and Tsong Yueh Chen. Comparison of adaptive random testing and random testing under various testing and debugging scenarios. *Software: Practice and Experience*, 42(8):1055–1074, 2012.

[56] Johannes Mayer. Lattice-based adaptive random testing. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 333–336. ACM, 2005.

[57] Thomas J McCabe. *Structured testing*, volume 500. IEEE Computer Society Press, 1983. 1

[58] Glenford J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[59] Glenford J. Myers and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004. 2

[60] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. Wiley, 2011.

[61] Simeon Ntafos. On random and partition testing. In *ACM SIGSOFT Software Engineering Notes*, volume 23, pages 42–48. ACM, 1998.

[62] Simeon C. Ntafos. On comparisons of random, partition, and proportional partition testing. *IEEE Trans. Softw. Eng.*, 27:949–960, October 2001.

[63] A. Jefferson Offutt and J. Huffman Hayes. A semantic model of program faults. *SIGSOFT Softw. Eng. Notes*, 21(3):195–200, May 1996. 2

[64] Catherine Oriat. Jartege: a tool for random generation of unit tests for java classes. *CoRR*, abs/cs/0412012, 2004.

[65] M. Oriol. The york extensible testing infrastructure (yeti). 2010.

[66] M. Oriol. York extensible testing infrastructure, 2011.

[67] M. Oriol. Random testing: Evaluation of a law describing the number of faults found. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 201–210, 2012.

[68] M. Oriol. Random testing: Evaluation of a law describing the number of faults found. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 201 –210, april 2012.

[69] Manuel Oriol and Sotirios Tassis. Testing .net code with yeti. In *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, ICECCS '10, pages 264–265, Washington, DC, USA, 2010. IEEE Computer Society.

[70] Manuel Oriol and Faheem Ullah. Yeti on the cloud. *Software Testing Verification and Validation Workshop, IEEE International Conference on*, 0:434–437, 2010.

[71] Thomas Ostrand. White-box testing. *Encyclopedia of Software Engineering*, 2002.

[72] Carlos Pacheco. *Directed random testing*. PhD thesis, Massachusetts Institute of Technology, 2009.

[73] Carlos Pacheco and Michael D. Ernst. Eclat: Automatic generation and classification of test inputs. In *In 19th European Conference Object-Oriented Programming*, pages 504–527, 2005.

[74] Carlos Pacheco and Michael D. Ernst. Randoop: feedback-directed random testing for Java. In *OOPSLA 2007 Companion, Montreal, Canada*. ACM, October 2007.

[75] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-directed random test generation. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 75–84, Washington, DC, USA, 2007. IEEE Computer Society. 2

[76] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. Feedback-directed random test generation. In *Proceedings of the 29th international conference*

on Software Engineering, ICSE '07, pages 75–84, Washington, DC, USA, 2007. IEEE Computer Society.

[77] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, Jiayang Sun, and Bin Wang. Automated support for classifying software failure reports. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 465 – 475, may 2003.

[78] CV Ramamoorthy and Sill-bun F Ho. Testing large software with automated software evaluation systems. In *ACM SIGPLAN Notices*, volume 10, pages 382–394. ACM, 1975.

[79] Debra J Richardson, Stephanie Leif Aha, and T Owen O'malley. Specification-based test oracles for reactive systems. In *Proceedings of the 14th international conference on Software engineering*, pages 105–118. ACM, 1992.

[80] Koushik Sen. Effective random testing of concurrent programs. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 323–332. ACM, 2007.

[81] E. Tempero. An empirical study of unused design decisions in open source java software. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 33 –40, dec. 2008.

[82] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, December 2010.

[83] Ewan Tempero, Steve Counsell, and James Noble. An empirical study of overriding in open source java. In *Proceedings of the Thirty-Third Australasian Conferenc on Computer Science - Volume 102*, ACSC '10, pages 3–12, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[84] Nigel Tracey, John Clark, Keith Mander, and John McDermid. An automated framework for structural test-data generation. In *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*, pages 285–288. IEEE, 1998.

[85] Jan Tretmans and Axel Belinfante. Automatic testing with formal methods. 1999.

[86] Willem Visser, Corina S P?s?reanu, and Sarfraz Khurshid. Test input generation with java pathfinder. *ACM SIGSOFT Software Engineering Notes*, 29(4):97–107, 2004.

[87] Elaine J Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[88] Lee J. White. Software testing and verification. *Advances in Computers*, 26(1):335–390, 1987. 2

[89] Wikipedia. Plagiarism — Wikipedia, the free encyclopedia, 20013. [Online; accessed 23-Mar-2013]. 2

[90] S. Yoo and M. Harman. Test data regeneration: generating new test data from existing test data. *Softw. Test. Verif. Reliab.*, 22(3):171–201, May 2012.