10111 Abstracts Collection
Practical Software Testing : Tool Automation and Human Factors
— Dagstuhl Seminar —

Mark Harman[1], Henry Muccini[2], Wolfram Schulte[3] and Tao Xie[4]

[1] King's College - London, GB
Mark.Harman@kcl.ac.uk
[2] Univ. degli Studi di L'Aquila, IT
henry.muccini@di.univaq.it
[3] Microsoft Corp. - Redmond, US
schulte@microsoft.com
[4] North Carolina State University, US
xie@csc.ncsu.edu

Abstract. From March 14, 2010 to March 19, 2010, the Dagstuhl Seminar 10111 "Practical Software Testing : Tool Automation and Human Factors" was held in Schloss Dagstuhl – Leibniz Center for Informatics. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Software testing, Test generation, Test automation, Test oracles, Testing tools, Human-computer interaction, Code-based testing, Specification-based testing, Model-based testing

10111 Executive Summary – Practical Software Testing : Tool Automation and Human Factors

The main goal of the seminar "Practical Software Testing : Tool Automation and Human Factors" was to bring together academics working on algorithms, methods, and techniques for practical software testing, with practitioners, interested in developing more soundly-based and well-understood testing processes and practices. The seminar's purpose was to make researchers aware of industry's problems, and practitioners aware of research approaches. The seminar focused in particular on testing automation and human factors. In the week of March 14-19, 2010, 40 researchers from 11 countries (Canada, France, Germany, Italy, Luxembourg, the Netherlands, Sweden, Switzerland, South Africa, United Kingdom, United States) discussed their recent work, and recent and future trends

in software testing. The seminar consisted of five main types of presentations or activities: topic-oriented presentations, research-oriented presentations, short self-introduction presentations, tool demos, and working group meetings and presentations.

Keywords: Software testing, test generation, test automation, test oracles, testing tools, human-computer interaction, code-based testing, specification-based testing, model-based testing

Joint work of: Harman, Mark; Muccini, Henry; Schulte, Wolfram; Xie, Tao

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2623

Model-Based Testing for the Cloud

Robert Hierons (Brunel University, GB)

Software in the cloud is characterised by the need to be highly adaptive and continuously available. Incremental changes are applied to the deployed system and need to be tested in the field. Different configurations need to be tested. Higher quality standards regarding both functional and non-functional properties are put on those systems, as they often face large and diverse customer bases and/or are used as services from different peer service implementations. The properties of interest include interoperability, privacy, security, reliability, performance, resource use, timing constraints, service dependencies, availability, and so on. This paper discusses the state of the art in model-based testing of cloud systems. It focuses on two central aspects of the problem domain: (a) dealing with the adaptive and dynamic character of cloud software when tested with model-based testing, by developing new online and offline test strategies, and (b) dealing with the variety of modeling concerns for functional and non-functional properties, by devising a unified framework for them where this is possible. Having discussed the state of the art we identify challenges and future directions.

Keywords: Cloud computing, model-based testing, non-functional properties

Joint work of: Bertolino, Antonia; Grieskamp, Wolfgang; Hierons, Robert; Le Traon,Yves; Legeard, Bruno; Muccini, Henry; Paradkar, Amit; Rosenblum, David; Tretmans, Jan

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2625

Computing and Diagnosing Changes in Unit Test Energy Consumption

Andrew J. Ko (University of Washington, US)

Many developers have reason to be concerned with with power consumption.

For example, mobile app developers want to minimize how much power their applications draw, while still providing useful functionality. However, developers have few tools to get feedback about changes to their application's power consumption behavior as they implement an application and make changes to it over time. We present a tool that, using a team's existing test cases, performs repeated measurements of energy consumption based on instructions executed, objects generated, and blocking latency, generating a distribution of energy use estimates for each test run, recording these distributions in a time series of distributions over time. Then, when these distributions change substantially, we inform the developer of this change, and offer them diagnostic information about the elements of their code potentially responsible for the change and the inputs responsible. Through this information, we believe that developers will be better enabled to relate recent changes in their code to changes in energy consumption, enabling them to better incorporate changes in software energy consumption into their software evolution decisions.

Keywords: Energy, oracles

Joint work of: Ko, Andrew J.; Young, Michal; Andrews, Jamie; Robinson, Brian P.; Grechanik, Mark

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2624

Groundwork for the Development of Testing Plans for Concurrent Software

Eileen Kraemer (University of Georgia, US)

While multi-threading has become commonplace in many application domains (e.g., embedded systems, digital signal processing (DSP), networks, IP services, and graphics), multi-threaded code often requires complex co-ordination of threads. As a result, multi-threaded implementations are prone to subtle bugs that are difficult and time-consuming to locate. Moreover, current testing techniques that address multi-threading are generally costly while their effectiveness is unknown. The development of cost-effective testing plans requires an in-depth study of the nature, frequency, and cost of concurrency errors in the context of real-world applications. The full paper will lay the groundwork for such a study, with the purpose of informing the creation of a parametric cost model for testing multi-threaded software. The current version of the paper provides motivation for the study, an outline of the full paper, and a bibliography of related papers.

Keywords: Concurrency, testing

Joint work of: Kraemer, Eileen; Dillon, Laura

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2621

Introducing Continuous Systematic Testing of Evolving Software

Per Runeson (Lund University, SE)

In today's evolutionary development of software, continuous testing is needed to ensure that the software is still functioning after changes. Test automation helps partly managing the large number of executions needed, but there is also a limit for how much automated tests may be executed. Then systematic approaches for test selection are needed also for automated tests. This manuscript defines this situation and outlines a general method and tool framework for its solution. Experiences from different companies are collected to illustrate how it may be set into practice.

Keywords: Regression testing, Continuous testing, Test selection

Joint work of: Harrold, Mary Jean; Marinov, Darko; Oney, Stephen; Pezze, Mauro; Porter, Adam; Penix, John; Runeson, Per; Yoo, Shin

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2622

AUTOMOCK: Automated Synthesis of a Mock Environment for Test Case Generation

Paolo Tonella (Fondazione Bruno Kessler - Trento, IT)

During testing, there are several reasons to exclude some of the components used by the unit under test, such as: (1) the component affects the state of the world in an irreversible way; (2) the component is not accessible for testing purposes (e.g., a web service); (3) the component introduces a major performance degradation to the testing phase (e.g., due to long computations); (4) it is hard (i.e., statistically unlikely) to obtain the output required by the test from the component. In such cases, we replace the component with a mock one. In this paper, we integrate the synthesis of mock components with the generation of test cases for the current testing goal (e.g., coverage). To avoid the generation of meaningless data, which may lead to assertion violation not related to bugs, we include a weak mock postcondition. We consider ways to automatically synthesize such postcondition. We empirically evaluate the quality of the mocks generated by our approach, as well as the benefits mocks introduce in terms of improved coverage and improved performance of the test case generator.

Keywords: Test case generation; code analysis; automated software testing.

Joint work of: Alshahwan, Nadia; Jia, Yue; Lakhotia, Kiran; Fraser, Gordon; Shuler, David; Tonella, Paolo

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2618

FITE - Future Integrated Testing Environment

Michael W. Whalen (University of Minnesota, US)

It is a well known fact that the later software errors are discovered during the development process, the more costly they are to repair. Recently, automatic tools based on static and dynamic analysis have become widely used in industry to detect errors, such as null pointer dereferences, array indexing errors, assertion violations, etc. However, these techniques are typically applied late in the development cycle, and thus, the errors detected by such approaches are expensive to repair. Additionally, these techniques can suffer from scalability and presentation issues due to the fact that they are applied late in the development cycle.

To address these issues we suggest that code should be continuously analyzed from an early stage of development, preferably as the code is written. This will allow developers to get instant feedback to repair errors as they are introduced, rather than later when it is more expensive. This analysis should also be incremental in nature to allow better scaling. Additionally, the presentation of errors in static and dynamic analysis tools can be improved due to the small increment of code being analyzed.

Keywords: Incremental analysis, incremental testing, human factors, static analysis, model checking

Joint work of: Godefroid, Patrice; Mariani, Leonardo; Polini, Andrea; Tillmann, Nikolai; Visser, Willem; Whalen, Michael W.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2619

Model-based Testing – Next Generation Functional Software Testing

Bruno Legeard (Smartesting - Besancon, FR)

The idea of model-based testing is to use an explicit abstract model of a SUT and its environment to automatically derive tests for the SUT: the behavior of the model of the SUT is interpreted as the intended behavior of the SUT. The technology of automated model-based test case generation has matured to the point where large-scale deployments of this technology are becoming commonplace. The prerequisites for success, such as qualification of the test team, integrated tool chain availability and methods, are now identified, and a wide range of commercial and open-source tools are available.

Although MBT will not solve all testing problems, it is an important and useful technique, which brings significant progress over the state of the practice for functional software testing effectiveness, and can increase productivity and improve functional coverage.

In this talk, we'll adress the current trend of deploying MBT in the industry, particularly in the TCoE - Test Center of Excellence - managed by the big System Integrators, as a vector for software testing "industrialization".

Keywords: Model-based testing, functional testing, test automation, process industrialization

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2620

Preemption Sealing for Efficient Concurrency Testing

Thomas Ball (Microsoft Research - Redmond, US)

The choice of where a thread scheduling algorithm preempts one thread in order to execute another is essential to reveal concurrency errors such as atomicity violations, livelocks, and deadlocks. We present a scheduling strategy called *preemption sealing* that controls where and when a scheduler is *disabled* from preempting threads during program execution. We demonstrate that this strategy is effective in addressing two key problems in testing industrial-scale concurrent programs: (1) tolerating existing errors in order to find more errors, and (2) compositional testing of layered, concurrent systems. We evaluate the effectiveness of preemption sealing, implemented in the CHESS tool, for these two scenarios on newly released concurrency libraries for Microsoft's .NET framework.

Keywords: Concurrency, testing, correctness

Research on Design for Verification of Multi-threaded Software

Laura Dillon (Michigan State University, US)

This extended abstract describes recent and on-going research in design for verification (D4V) of multi-threaded software, which is being conducted by the Concurrent Software Design Group at Michigan State University. Specifically, we outline work that leverages synchronization contracts to organize and connect design artifacts with code in a way that facilitates long-term maintenance, localizes conformance obligations, and supports code generation and compositional correctness proofs. Additionally, we highlight results of empirical assessment of several notations and methods used in development and maintenance of concurrent software. More information can be found at `http://www.cse.msu.edu/sens/szumo`, `http://www.cse.msu.edu/sens/copse`.

Keywords: Design for verification, synchronization contracts, empirical assessment of development tools

Preserving Test Coverage While Achieving Data Anonymity For Database-Centric Applications

Mark Grechanik (Accenture Labs - Chicago, US)

Database-centric applications (DCAs) are common in enterprise computing, and they use nontrivial databases. Testing of DCAs is increasingly outsourced to test centers in order to achieve lower cost and higher quality. When releasing proprietary DCAs, its databases should also be made available to test engineers, so that they can test using real data. Testing with real data is important, since fake data lacks many of the intricate semantic connections among the original data elements. However, different data privacy laws prevent organizations from sharing these data with test centers because databases contain sensitive information. Currently, testing is performed with fake data that often leads to worse code coverage and fewer uncovered bugs, thereby reducing the quality of DCAs and obliterating benefits of test outsourcing.

We offer a novel approach, Testing Applications with Data Anonymization (TaDa). With TaDa, DCAs can be released to external testing organizations without disclosing sensitive information while retaining testing efficacy. We built a tool and applied it to example Java DCAs. Our results show that for the same level of anonymity TaDa enables higher test coverage than the current state-of-the-art anonymization algorithm, Datafly.

Keywords: K-anonymity, test coverage, quasi-identifiers, database-centric application

Distributed testing

Robert Hierons (Brunel University, GB)

This talk gives a brief introduction to my interest in distributed testing. Here, we have a system under test (SUT) that has physically distributed interfaces (ports). We place a tester at each port and a tester at port p sees only the events (inputs and outputs) at p. Thus, testing is black-box. The distributed nature of testing can lead to races (controllability problems) and also weaker implementation/conformance relations.

Keywords: Distributed testing, black-box testing, controllability, observability

Software Engineering of Concurrent Systems

Eileen Kraemer (University of Georgia, US)

These few slides provide a brief overview of my past and current work, beginning with a focus on the visualization of concurrent software, then on to the interactive steering of concurrent systems, and more recently focusing on diagrams, notations and tools that support software engineering tasks for concurrent software.

Keywords: Software engineering, concurrent software

Brief Overview of Darko Marinov's Group

Darko Marinov (University of Illinois - Urbana, US)

Darko Marinov is an assistant professor at the University of Illinois at Urbana-Champaign, where he is currently (co)advising five PhD students who work on the following topics. Steven Lauterburg works on testing message-passing code (actors). Brett Daniel works on test repair and test quality/augmentation. Vilas Jagannath works on testing shared-memory code and mutation testing. Milos Gligoric works on test generation and regression testing. Adrian Nistor works on hardware support for testing parallel code. More information can be found at http://mir.cs.illinois.edu/~marinov

Keywords: Test generation, test repair, regression testing, testing parallel code

Model-Based Testing

Jan Tretmans (Embedded Systems Institute - Eindhoven, NL)

Model-based testing is one of the promising technologies to meet the challenges imposed on software testing. In model-based testing, test cases are generated from a model that describes the required behaviour of the implementation under test. Model-based testing is more than just the generation of some test cases from a model. A well-defined and sound theory of model-based testing is feasible and necessary, and it brings many benefits, also practical ones. As an example we consider the *ioco*-testing theory, where models are expressed as labelled transition systems and correctness is defined with an implementation relation called *ioco*.

Keywords: Model-based testing

Full Paper:
http://dx.doi.org/10.1007/978-3-540-78917-8_1

See also: Tretmans, J., *Model Based Testing with Labelled Transition Systems*, in: R. Hierons, J. Bowen and M. Harman, editors, *Formal Methods and Testing*, Lecture Notes in Computer Science **4949** (2008), pp. 1–38.

Eco Testing for Components

Jan Tretmans (Embedded Systems Institute - Eindhoven, NL)

In component-based development, the correctness of a system depends on the correctness of the individual components and on their interactions. Model-based testing is a way of checking the correctness of a component by means of executing test cases that are systematically generated from a model of the component. This model should include the behaviour of how the component can be invoked, as well as how the component itself invokes other components. In many situations, however, only a model that specifies how others can use the component, is available. In this presentation we present an approach for model-based testing of components where only these available models are used. Test cases for testing whether a component correctly reacts to invocations are generated from this model, whereas the test cases for testing whether a component correctly invokes other components, are generated from the models of these other components. A formal elaboration is given in the realm of labelled transition systems. This includes an implementation relation, called *eco*, which formally defines when a component is correct with respect to the components it uses, and a sound and exhaustive test generation algorithm for *eco*.

Keywords: Model-based testing

Full Paper:
http://dx.doi.org/10.1007/978-3-540-74792-5_1

See also: Frantzen, L. and J. Tretmans, *Model-Based Testing of Environmental Conformance of Components*, in: F. de Boer, M. Bosangue, S. Graf and W.-P. de Roever, editors, *Formal Methods for Components and Objects*, Lecture Notes in Computer Science **4709** (2007), pp. 1–25.

Testing in Academia and Industry

Willem Visser (Stellenbosch University - Matieland, ZA)

This talk starts with a short self-introduction by means of tracing through my career from research at NASA, to SEVEN Networks and back to academia at Stellenbosch University. It highlights how testing at NASA was all about the once in a lifetime bugs, whereas at SEVEN bugs were common and the important ones were the ones that cost the company money.

Keywords: Testing, model checking, symbolic execution

Integration of Testing and Analysis

Michael W. Whalen (University of Minnesota, US)

It is a well known fact that the later software errors are discovered during the development process, the more costly they are to repair. Recently, automatic tools based on static and dynamic analysis have become widely used in industry to detect errors, such as null pointer dereferences, array indexing errors, assertion violations, etc. However, these techniques are typically applied late in the development cycle, and thus, the errors detected by such approaches are expensive to repair. Additionally, these techniques can suffer from scalability and presentation issues due to the fact that they are applied late in the development cycle.

To address these issues we suggest that code should be continuously analyzed from an early stage of development, preferably as the code is written. This will allow developers to get instant feedback to repair errors as they are introduced, rather than later when it is more expensive. This analysis should also be incremental in nature to allow better scaling. Additionally, the presentation of errors in static and dynamic analysis tools can be improved due to the small increment of code being analyzed.

Keywords: Incremental analysis, incremental testing, human factors, static analysis

Joint work of: Godefroid, Patrice; Mariani, Leonardo; Polini, Andrea; Tillmann, Nikolai; Visser, Willem; Whalen, Michael W.

Requirements and Coverage Metrics

Michael W. Whalen (University of Minnesota, US)

When creating test cases for software, a common approach is to create tests that exercise requirements. Determining the adequacy of test cases, however, is generally done through inspection or indirectly by measuring structural coverage of an executable artifact (such as source code or a software model). We present ReqsCov, a tool to directly measure requirements coverage provided by test cases. ReqsCov allows users to measure Linear Temporal Logic requirements coverage using three increasingly rigorous requirements coverage metrics: naïve coverage, antecedent coverage, and Unique First Cause coverage. By measuring requirements coverage, users are given insight into the quality of test suites beyond what is available when solely using structural coverage metrics over an implementation.

Keywords: Requirements, testing, test automation, structural coverage metrics,

Self Introduction: Improving Software Reliability and Productivity via Testing and Mining

Tao Xie (North Carolina State University, US)

We have worked on two research areas for improving software reliability and productivity: automated software testing and mining software engineering data. In the area of mining software engineering, we have developed approaches for mining source code, textual data, and program executions. In the area of automated software testing, we have developed approaches for test generation, regression testing, test oracles, mutation testing, and security policy testing. Along the tool automation and human factors in practical software testing, we have developed automated techniques to boost the upper limit of tool automation and proposed the concept of "cooperative testing" between developers/testers and tools. More information can be found at https://sites.google.com/site/asergrp/.

Keywords: Test automation, test generation, cooperative testing

Self Introduction: Michal Young

Michal Young (University of Oregon, US)

Michal Young works on a variety of things, but the theme running through all of them is a concern with how to exploit modular structure to make reasoning easier. Naturally, then, he thinks exploiting modularity is the key to human factors in software testing.

Keywords: Self introduction