Accepted manuscript (May 2015) – IEEE Software Magazine

Test Better by Exploring: Harnessing Human Skills and Knowledge

Juha Itkonen
Department of Computer Science, Aalto University
FI-00076 AALTO, FINLAND
juha.itkonen@aalto.fi
+358 505771688

Mika V. Mäntylä
Department of Information Processing Science, University of Oulu
FI-90014 OULU, FINLAND
mika.mantyla@oulu.fi

Department of Computer Science, Aalto University
FI-00076 AALTO, FINLAND

Casper Lassenius
Department of Computer Science, Aalto University
FI-00076 AALTO, FINLAND
casper.lassenius@aalto.fi

Abstract

End users continue to stumble upon software bugs, despite developers' efforts to build and test high-quality software. While traditional testing and quality assurance techniques are extremely valuable, we suggest that more focus should be given to the role of exploration in software testing. Exploration can bring direct utilization of knowledge and learning to the core of industrial software testing, helping to earlier reveal more relevant bugs. We describe the characteristics of exploration, the role of knowledge in software testing, and describe three levels of practices in exploratory testing. We propose that academics and practitioners focus their attention to exploiting the strengths of exploration in software testing and reporting existing practices and benefits form varying contexts, both from industry and academia.

Keywords

D.2.4.i Validation, D.2.5.k Testing strategies, D.2.5.l Test design, D.2.5.q Test management, D.2.19.c Methods for SQA and V&V, Exploratory testing

Introduction

Why do end users keep detecting bugs despite of the vendors' investments to testing and quality assurance? We suggest that more focus should be given to the role of exploration in software testing as a way of better addressing end users' needs. Traditionally, software testing emphasizes the need for systematic and documented approaches to detect bugs, using predefined test cases. Some models assume that when human testers are involved, their main task is to more or less mechanically execute the test cases and report any deviations from the expected results. However, testing professionals often see testing in a remarkably different light, describing it as an intellectually challenging, creative, and professionally demanding task that requires a wide variety of knowledge and skills [1].

In industrial practice, the contribution of human testers is a highly relevant part of software development, because most new bugs are found by humans testing the software, especially in the context of interactive systems. Human testers possess benefits over machines, including knowledge, creativity, intelligence, the ability to learn and adapt to new situations, and the ability to efficiently recognize problems.

In this article, we focus on the role of exploration as a facet of software testing. We discuss the benefits of exploration, the importance of the personal knowledge and skills, and the exploration practices used in industry. We encourage more research investments to these exploratory and human aspects in order to leverage what has been found working in practice.

Exploration as a Facet of Software Testing

Exploration occurs when the route to be traversed and the discoveries to be made are not known in advance. This is also the case in software testing, since the value and goal of testing is to discover new information about the (unknown) quality of the tested system.

Varying Degrees of Exploration

Software testing involves various degrees of exploring from a completely automated, confirmatory approach, to pure exploration, as illustrated in Figure 1. Between these two extremes, there are varying degrees of exploring associated with different types of testing. At one end of the spectrum, we have confirmatory testing. In this traditional testing paradigm, the aim is to design and document the test cases beforehand; then the tester's task is to follow these test cases accurately during test execution. The expected results are documented as predefined hypotheses of how the system shall work and the defect detection is based on checking against the documented results, see sidebar 1. In practice, even this confirmatory approach, if performed by humans, involves some exploration. However, the fundamental paradigm in confirmatory testing is to design and document the tests in a separate phase preceding the test execution.

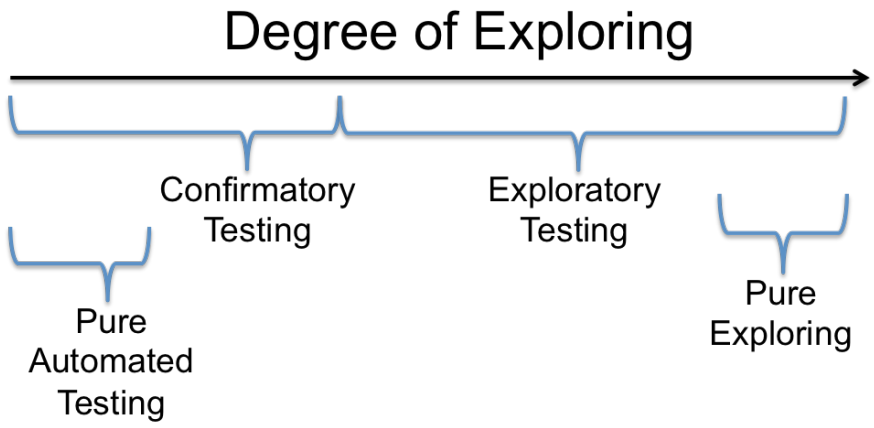

Fig. 1 The degree of exploring and approaches to testing

When the degree of exploration increases towards the exploratory paradigm, the tester is put in the center, rather than the test case documentation. Bach [2] has described this difference in the terms of testers' increasing freedom.

Exploratory Testing (ET) has been strongly advocated by the Context-Driven School of Testing [3]. ET is defined as "*a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project.*" [4] During the last decade, ET has been recognized as a serious approach, and has gained strong support within the practitioner community, e.g., in agile contexts.

The exploratory tester uses documentation and tools as much or as little as necessary. The tester is given a goal and the responsibility to carry out the testing, but no detailed instructions on how to accomplish the goal. ET can be free of planned structures or fairly structured—the tester might follow a list of features to be tested, limiting the exploration at the higher level—but ultimately, the testing of each feature is exploratory. Thus, ET can be structured and planned without restricting the tester's freedom to choose the best ways to test within the limits of the structure.

Table 1 compares ET with confirmatory testing. Confirmatory testing emphasizes mechanic, document-driven repeatability in the test execution. ET highlights knowledge, learning, and discovery of new information during software testing.

Table 1. Contrasting Exploratory Testing, Confirmatory Testing, and Automation

	Exploratory Testing	Confirmatory Testing	
	Performed by human testers		Automated
Testing philosophy	Testing is a knowledge intensive and creative activity requiring skills.	Testing is a mechanic and repetitive activity that can be described in explicit instructions.	Testing is automated and repeatable to provide fast feedback to development.
Test design	Test design and execution are parallel activities and proper test design requires exploratory learning of the tested system.	Test design and execution are separate and sequential activities.	Test design is challenging and expensive, but execution is fast and cheap.
Role of documentation	Testing requires knowledge and skills that are difficult to transfer through documentation.	Testing can be distributed to a large number of people with low knowledge and skill levels by relying on documented tests.	Certain types of tests can be scripted and effectively automated.
Knowledge needs	Software malfunctions in unpredictable ways and detecting these bugs requires knowledge of both the system and the application domain.	Software bugs can be predicted and expected outcomes documented for straightforward execution-time comparison.	Certain types of bugs can be effectively detected automatically.
Repeatability	Repeating the same tests over and over again does not reveal new bugs or provide new information of the quality and thus provides little added value.	Repeatability of tests is important. Exact test case descriptions reduce the individual variation in testing.	Repeatability and execution of tests in very short cycles is important to get fast feedback on regression during development.
Role of automation	Automation is one of the testers' tools, and should be used whenever reasonable to improve testing and free human resources for other types of testing activities.	Groups of human testers can be used for similar goals as automation.	Automation is the primary goal and testing should be automated as far as possible.

The goals of confirmatory human testing are similar to the automated software testing paradigm. This actually does not make much sense—confirmatory human testing is slow, laborious, and error-prone compared to automated tests. Furthermore, confirmatory testing lacks the benefits of exploration, since it strives for repeated checking of a predefined set of outcomes. This makes it a bad compromise between two clearly different approaches: exploratory vs. automated confirmatory, with their unique strengths. Instead of aiming at replacing human testers, automation is a way to remove repetitive and laborious checking tasks to free up tester's time for tasks that require more skill and knowledge.

Applying Knowledge when Exploring

Testers' personal experience and domain knowledge have been recognized as important aspects affecting the results of testing [5], [6]. Testers apply knowledge to different tasks and for different purposes, e.g., designing effective tests and recognizing bugs. It has been reported that application domain experts detect and report more relevant findings than non-experts [7].

A large part of the applied knowledge in software testing is tacit in nature, i.e. knowledge that cannot or has not been made explicit [8]. The difference in how knowledge is applied in the confirmatory and exploratory approaches is illustrated in Figure 2. Both approaches require knowledgeable people with sufficient testing skills. In both approaches the test design activity involves both tester's tacit knowledge and available documented, explicit, knowledge that the tester uses to design the tests. The test design activity is a highly exploratory and creative task and that should be recognized also in the confirmatory approach. E.g., exploring the actual system implementation can give much richer knowledge as a basis for test design than only studying the available specifications. In the exploratory approach, this design knowledge is applied directly by the tester, and the resulting findings fed directly back to the design and analysis process, without making it explicit by documentation or transferred to other persons. In the confirmatory approach, however, test design, execution and reporting are seen as separate phases. The person with the knowledge (test designer) transfers it in the form of explicit test cases to another person(s) for test execution, who in turn report their findings for other stakeholders. These multiple knowledge transfers, illustrated in Figure 2, are highly problematic and make the confirmatory process inefficient for humans to perform–the more complex the application domain and technical solution are, the more challenging these knowledge transfers become.

One important aspect in exploration is the immediate feedback loop from the result of a previous test to the design of the next test. This makes it possible to guide the testing based on the discoveries made in interaction with the actual system, and gives the tester the opportunity for learning and discovery of new, previously unknown knowledge—more so than in the confirmatory following-the-script approach. This is a benefit highly emphasized by ET advocates [3], [9].

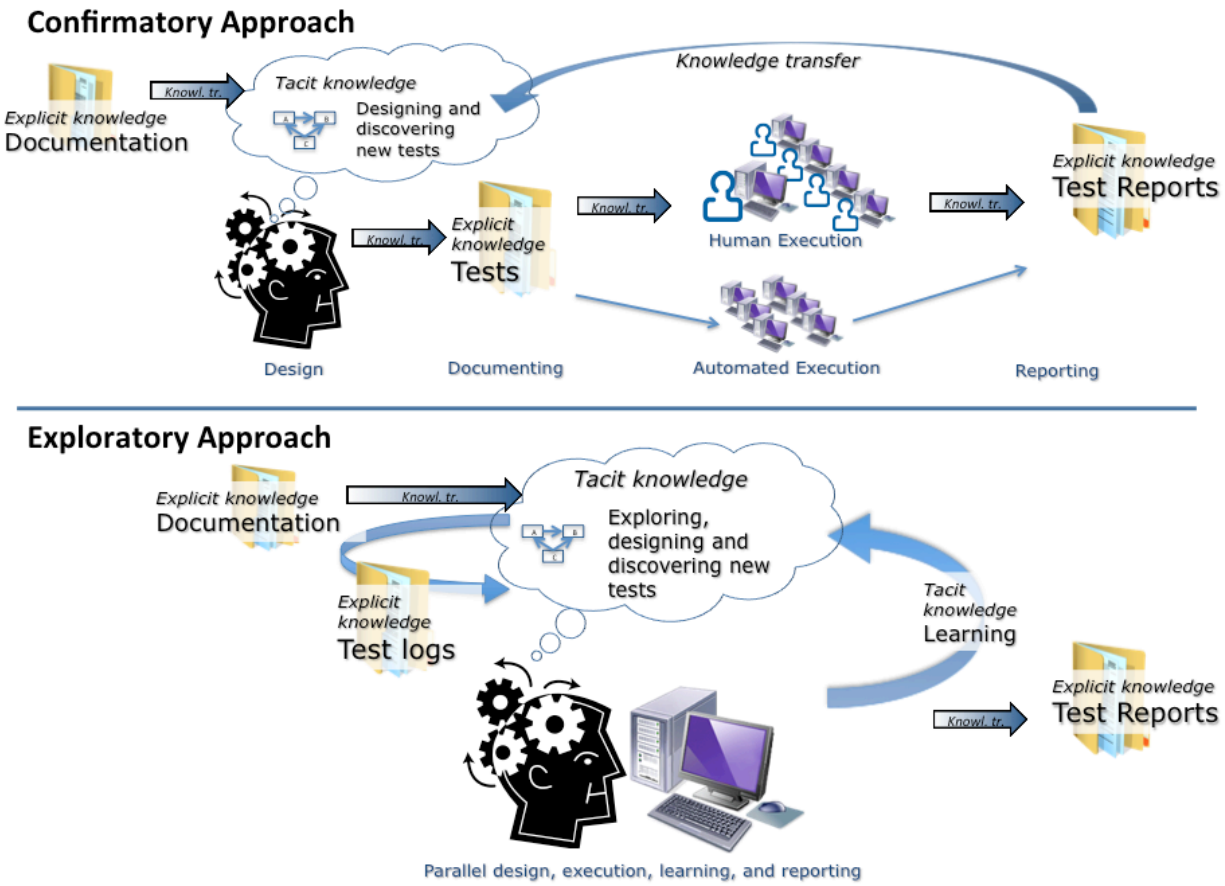

Fig. 2 The difference of confirmatory and exploratory approaches

The exploratory approach uses people with the required knowledge to do the actual testing, e.g., individuals with direct knowledge of the customer's business processes. This helps reveal problems that would be very obvious to end users, but alien to developers. It is far too common that the features of a software system have been thoroughly tested, yet nobody has tried to use the system for real. In addition, exploring is also a good way to gain that required knowledge. One could argue that it is difficult for people with the right knowledge to find the time to perform testing. However, it would be even more difficult to get them to document their tacit knowledge into test cases. Transferring tacit knowledge in explicit form can be extremely expensive or impossible depending on the type of the knowledge [8], see Sidebar 1.

Levels of Practices in Exploratory Testing

When applying ET, certain management practices are needed. We describe such practices, distilled from several industrial organizations and sources. During ET, testers apply test design strategies and techniques as they do in confirmatory testing. In addition, an exploratory tester

needs a method or strategy for guiding the testing on a higher level. We have identified practices on three levels: the organizing level, the session level, and the technique level.

Organizing Level Practices

The most commonly proposed solution to managing ET is Session-Based Exploratory Testing (SBET) [10]. SBET enables planning, tracking and reporting without sacrificing the flexibility benefits of ET, see Sidebar 2. At the heart of SBET are restricted, time-boxed testing sessions. Within the limits of these, typically roughly 2-hour, sessions the tester's activities are guided by a brief testing charter, including a mission statement and tested areas, but without further pre-design of testers actions. Next, we describe some practices for carrying out the actual testing work during the testing sessions.

Session Level Exploration Strategies

Testers benefit from an *exploration strategy* during a test session. These strategies are needed to guide the tester through a part of the tested software so that afterwards it is possible to describe what was covered and what is still missing. These exploration strategies are not strict paths for the tester; they provide a guiding principle, and the central characteristic of exploration is that the tester is encouraged to explore anything that seems interesting, suspicious, or otherwise valuable to the testing task at hand.

Examples of exploration strategies that we have observed in our studies include exploring weak areas or simulating a real usage scenario [11]. Exploration strategies can also be based on documentation such as user guides, specifications, or test data. For example, the functional specification or release notes can provide a structure for what to cover and how to proceed. The actual testing activity for each tested function can either be purely exploratory or follow selected testing techniques. Another proposed group of exploring strategies uses a tour metaphor, in which a tester explores a software system similarly to how a tourist explores a foreign city [1].

Testing Technique Level Practices

Detailed test design techniques are also applied in ET in order to design the actual tests that are executed. Some of the techniques are similar to the traditional test design techniques, such as boundary value analysis, and combination testing. At this level, ET allows the tester to apply the techniques and strategies best suitable for the specific testing task at hand, without the need for pre-specifying the steps [11].

The selection of the testing techniques is based on professional expertise. For example, for testing a given feature the tester might select a pair-wise combinatorial design for testing the interaction effects of a set of variables, based on the tester's knowledge of the interactions of those variables in the system.

Finally, oracle heuristics form a characteristic group of practices for ET that guide a tester to recognize certain types of problems in the system. Confirmatory testing relies on the existence and correctness of the documented expected outcome for each test. Documented outcomes aim at making the recognition of a failure a trivial checking activity, which it is not in practice—software systems fail in numerous, unanticipated ways and anticipating all these in explicit test case format is impossible. This is why testers need to rely on heuristics and knowledge-based oracles. The ET approach harnesses the human capabilities, such as tacit domain knowledge, for efficiently recognizing unanticipated problems in the tested system [5].

Conclusions

Exploration is an essential facet of software testing. It provides benefits for testing systems with rich user interaction, a complex application domain, or social contexts that require human expertise to understand. Despite the small amount of research on ET, see sidebar 2, there is support for the main benefits of the exploratory approach: the efficiency of defect detection, due to the reduced investment in test case pre-design; the high level of flexibility in testing activities, with fast testing feedback on new features and new risks; the efficient utilization of testers' knowledge directly in testing; and finally, the ability to reveal bugs and problems that algorithmic confirmatory testing cannot discover.

With the benefits, though, come certain challenges. The first challenge is the lack of methodology and tool support for planning and tracking ET. The second challenge is determining the level and type of documentation that would best support the exploratory approach, which is practically an unstudied area in research literature. And finally, the exploratory approach would also need more emphasis in engineering education and training.

Despite exploration being a widely applied practice in the industry, experienced testers build their own exploratory practices in many organizations, taking up a lot of time, trial, and error. Research can help knowledge sharing and accelerate this progress. Thus, we propose the recognition of the exploratory approach and investment to a research program to study testing from a behavioral and social sciences viewpoint. Such ideas are currently gaining attention in the software engineering community [12].

Sidebar 1: Background on Exploration and Knowledge

Software testing provides understanding about the quality of software. Similarly, science creates understanding, but in a larger context. In science, exploratory research aims at new scientific discoveries, whereas confirmatory research aims at confirming existing theories [13]. Exploration refers to activity of examining, analyzing, or investigating something. It can be characterized as a travel over or through a particular space for the purposes of discovery and adventure. Exploration

can be focused on innovation, exhaustive discovery, or it can be limited to searching systematically for something in particular [13]. Research on information searching also distinguishes between exploration and lookup; exploratory search is characterized with such activities as knowledge acquisition, interpretation, synthesis and discovery, whereas lookup searches incorporate fact retrieval, known item search and verification, see [14].

These views of exploration are applicable to software testing: part of testing is confirmatory in nature, i.e., checking that previous or existing tests still pass, often referred to as regression testing in literature. On the other hand, finding new tests and discovering unknown problems is an exploratory activity where the goal is to reveal new knowledge by proposing new hypotheses and testing those empirically. These contrasting viewpoints have been raised and discussed in the ET community under the topic of "testing versus checking" [15]. Thus, it is important to recognize the differences of the confirmatory and exploratory types of testing and consider when the approaches are applicable and best supporting each other.

Software quality is an elusive target that is difficult to define, and quality as such is a concept that different stakeholders can have varying interpretations. Ill-defined areas are good targets for exploratory methods in general [13]. If it would be possible to accurately define what one means with software quality and how to measure it, then confirmatory methods should be preferred. If, on the other hand, the quality definitions or requirements are not straightforward, and rely on tacit knowledge of number of stakeholders, then exploratory methods are preferred.

One could argue that ET methods are only needed to patch poorly done requirements engineering and software design. Yet, such argument fails to understand that making all the knowledge required to evaluate software quality explicit is highly expensive or impossible. Software systems are used in a social context. According to research, the knowledge which is collective and part of social relations is the most difficult type of knowledge to make explicit [8]. The implication is that ET is crucial for systems used in complicated social context, whereas confirmatory approach excels in verifying algorithmic correctness.

References

[1] J. A. Whittaker, *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Boston, MA, USA: Addison-Wesley Professional, 2009.

[2] J. Bach, "A Case Against Test Cases," *Quardev Blog*, 15-Oct-2007. [Online]. Available: http://www.quardev.com/blog/a_case_against_test_cases.

[3] J. Bach, "Exploratory Testing," in *The Testing Practitioner*, Second., E. van Veenendaal, Ed. Den Bosch: UTN Publishers, 2004, pp. 253–265.

[4] C. Kaner, “Defining Exploratory Testing,” *Kaner.com*, 14-Jul-2008. [Online]. Available: http://kaner.com/?p=46.

[5] J. Itkonen, M. V. Mäntylä, and C. Lassenius, “The Role of the Tester’s Knowledge in Exploratory Software Testing,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 707–724, 2013.

[6] A. Beer and R. Ramler, “The Role of Experience in Software Testing Practice,” in *Proceedings of Euromicro Conference on Software Engineering and Advanced Applications*, 2008, pp. 258–265.

[7] A. Følstad, “Work-Domain Experts as Evaluators: Usability Inspection of Domain-Specific Work-Support Systems,” *Int. J. Hum.-Comput. Interact.*, vol. 22, no. 3, p. 217, 2007.

[8] H. Collins, *Tacit and Explicit Knowledge*, Reprint edition. Chicago; London: University Of Chicago Press, 2010.

[9] C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*. New York: John Wiley & Sons, Inc., 2002.

[10] J. Bach, “Session-Based Test Management,” *Software Testing and Quality Engineering*, vol. 2, no. 6, 2000.

[11] J. Itkonen, M. V. Mäntylä, and C. Lassenius, “How do testers do it? An exploratory study on manual testing practices,” in *Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 494–497.

[12] P. Lenberg, R. Feldt, and L.-G. Wallgren, “Towards a Behavioral Software Engineering,” in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, New York, NY, USA, 2014, pp. 48–55.

[13] R. A. Stebbins, *Exploratory Research in the Social Sciences*. Thousand Oaks: SAGE Publications, 2001.

[14] G. Marchionini, “Exploratory search: from finding to understanding,” *Commun ACM*, vol. 49, no. 4, pp. 41–46, Apr. 2006.

[15] J. Bach and M. Bolton, “Testing and Checking Refined,” *James Bach’s Blog*, 26-Mar-2013. [Online]. Available: http://www.satisfice.com/blog/archives/856.

Sidebar 2: Current Research on Exploratory Testing

A replicated experiment suggested that exploratory testing has same defect detection effectiveness, but superior efficiency compared to test-case-based testing [1]. An industrial case

study identified motivations for using exploratory testing [2]. First, ET was applied to tackle the combinatorial complexity by applying experience based test selection. Second, ET worked well with weak or frequently changing specifications and enabled quick feedback from testers to developers. Finally, ET facilitated learning about the new features of the product. Another industrial case study [3] highlighted the importance of ET in detecting non-functional aspects such as attractiveness, and usability. Researchers have proposed a hybrid approach to combine ET and scripted testing [4] and presented a team ET approach [5]. An industry survey found that ET is also used in critical domains to a high degree and raised the need for better tool support [6]. In addition, empirical research on the real-world practice of software testing [7], [8] sheds light on the social and exploratory aspects of testing.

The session based test management (SBTM) has been proposed to bring more accountability to ET [9]. According to industry reports, SBET enables measurement and control of the ET process and gives visibility of the work to the team and managers [10]. In medical domain, it was found that SBET diminished problems of highly compartmentalized testing, excess documentation, repetitive testing, and emphasized focus on requirements and code coverage instead of defect discovery [11].

[1] J. Itkonen and M. V. Mäntylä, “Are test cases needed? Replicated comparison between exploratory and test-case-based software testing,” *Empir. Softw. Eng.*, vol. 19, no. 2, pp. 303–342, Apr. 2014.

[2] J. Itkonen and K. Rautiainen, “Exploratory testing: a multiple case study,” in *Proceedings of International Symposium on Empirical Software Engineering*, 2005, pp. 84–93.

[3] J. Pichler and R. Ramler, “How to Test the Intangible Properties of Graphical User Interfaces?,” in *Proceedings of 1st International Conference on Software Testing, Verification, and Validation*, 2008, pp. 494–497.

[4] S. M. A. Shah, C. Gencel, U. S. Alvi, and K. Petersen, “Towards a hybrid testing process unifying exploratory testing and scripted testing,” *J. Softw. Evol. Process*, 2013.

[5] S. Saukkoriipi and I. Tervonen, “Team Exploratory Testing Sessions,” *ISRN Softw. Eng.*, vol. 2012, pp. 1–20, 2012.

[6] D. Pfahl, H. Yin, M. V. Mäntylä, and J. Münch, “How is Exploratory Testing Used? A State-of-the-Practice Survey,” in *Proceedings of the 8th International Symposium on Empirical Software Engineering and Measurement*, 2014, p. 10.

[7] D. Martin, J. Rooksby, M. Rouncefield, and I. Sommerville, “‘Good’ Organisational Reasons for ‘Bad’ Software Testing: An Ethnographic Study of Testing in a Small Software Company,” in *Proceedings of International Conference on Software Engineering*, 2007, pp. 602–611.

[8] J. Rooksby, M. Rouncefield, and I. Sommerville, "Testing in the Wild: The Social and Organisational Dimensions of Real World Practice," *Comput. Support. Coop. Work*, vol. 18, no. 5–6, pp. 559–580, 2009.

[9] J. Bach, "Session-Based Test Management," *Software Testing and Quality Engineering*, 2000. [Online]. Available: http://www.satisfice.com/articles/sbtm.pdf.

[10] J. Lyndsay and N. van Eeden, "Adventures in Session-Based Testing," 27-May-2003. [Online]. Available: http://www.workroom-productions.com/papers/AiSBTv1.2.pdf.

[11] B. Wood and D. James, "Applying Session-Based Testing to Medical Software," *Medical Device & Diagnostic Industry*, vol. 25, no. 5, p. 90, May-2003.

Author biographies

Juha Itkonen is a researcher at Aalto University. His research focuses on exploratory software testing and human issues in software engineering, including quality assurance in agile contexts. He has a D.Sc. degree from Aalto University. juha.itkonen@aalto.fi

Mika Mäntylä is a professor of software engineering at University of Oulu. His current research interests include software defects, behavioral software engineering, and software evolution. He has a D.Sc. degree from Aalto University. mika.mantyla@oulu.fi

Casper Lassenius is an associate professor at Aalto University. His current research interests include agile and lean software development, global software engineering, and software quality assurance. He has a D.Sc. degree from Aalto University. casper.lassenius@aalto.fi