Exploratory testing as a source of testing technical debt

SYED MUHAMMAD ALI SHAH, MARCO TORCHIANO, ANTONIO VETRO', MAURIZIO MORISIO

Automatics and Informatics Department (DAUIN)
Politecnico di Torino
Corso Duca degli Abruzzi, 24 10129
Torino, Italy
syed.shah@polito.it , marco.torchiano@polito.it, antonio.vetro@polito.it,
maurizio.morisio@polito.it

The cost of software testing is believed to range between 40 and 80% of the total cost of development, with safety critical systems closer to the high end [S1]. According to NIST [S2] inadequate software testing infrastructure is estimated to cost US society about 59.5 billion USD annually.

Among the several attempts to reduce such costs, automated software testing deserves a special mention [S3]. Automated software testing provides quick verification and reduces the testing execution effort but it requires a significant upfront investment to set up the infrastructure. The alternative, i.e. manual testing, is very labor intensive and requires human testers to execute the tests, e.g. test case based testing and exploratory testing. Despite the potentially higher costs, the latter approach is more common in industry compared to the former [S4].

The different types of manual testing approaches are typically selected based on the context; for instance structured and prescriptive techniques such as Test case based testing are adopted when a system undergoes the acceptance testing [S5].

In general, any technique typically leads to specific advantages, while it brings some drawbacks which often are not immediately apparent but tend to surface later. A testing approach introduces testing induced Technical Debt (TD) whose value is not known at the time of testing and appears in later phases of the life cycle.

We analyzed the software testing approach "*Exploratory Testing (ET)*" through a systematic review of literature to understand the consequences of ET as Technical debt. The evidence shows that ET is used as an alternative to any structured software testing approach to speed up the testing tasks and proved to be cost effective at the time of testing. Nevertheless ET also has many weaknesses that are not apparent at the time of testing but prompt up in later phases of system life cycle. Theses weaknesses incur increased rework and cost, and hence are considered to be the sources of TD. In addition we propose the possible solutions to embark upon these weaknesses that indeed help to reduce the testing technical debt of ET.

Technical Debt

The term Technical Debt refers to an increased cost of changing or maintaining a system in the future due to expedient shortcuts taken during development [1]. In other words, Technical Debt is a techno-financial instrument that makes quick development affordable at present time, at the cost of compound interests to be paid later [2].

Technical debt can be related to different activities: architecture, design, documentation, testing, and so on. We focus on Testing Technical Debt. We define one source of testing technical debt in the following way:

"Feature testing to attain one aspect while simultaneously ignoring -- either knowingly or not -- other aspects that may prompt up later with increased rework and cost"

There are different ways to deal with technical debt, Buschman [2] proposes three main categories of approaches:

- Pay the interests: we suffer the consequences of the debt and to cope with it we incur in repeated additional costs, e.g. in absence of automated tests we keep on carrying on expensive manual regression tests.
- Repay the debt: we get rid of the origin of the debt at the cost of significant extra rework effort, e.g. a structural limitation in the program is removed through an additional reengineering activity.
- Convert the debt: we replace the source of technical debt with another solution still implying some debt, though typically smaller, e.g. a flexible though hard-to-maintain run-time customization module is replaced with a rigid development-time one.

While the above definition may seem negative, contracting debt is not necessarily a bad thing. As in real-life situations a number of readers would not own a house without a mortgage, so most projects could not be successful.

Exploratory Testing

Exploratory testing (ET) is an approach that does not rely on the formal test case definition. In fact, the tester, instead of designing test cases, runs and evaluates the software behavior basing tests on his intuition and knowledge. The formal definition of ET was proposed by James Bach as [3]:

"Exploratory testing is simultaneous learning, test design, and test execution"

Given its widespread adoption in industrial practices to speed up verification activities of newly developed functionalities, several researchers have begun to focus their studies on ET. ET is considered as a cost effective practice due to the lack of test case documentation and planning. Moreover it also has good defect detection ability and is a very flexible process [4].

Practitioners tend to consider ET as a valid alternative to systematic testing approaches when not enough time is available [3]. In such context it is not possible to define plans and write test cases on the basis of requirement and design specification, and later abide. Then, adopting ET, testers utilize their own intuition and experience without the need of any documented guideline, testing only particular scenarios or functionalities. Such an unconstrained and creative approach makes ET attractive for testers. In addition, as a consequence ET provides rapid feedback, simultaneous learning and diversity in testing [3].

The benefits listed above makes ET very attractive. The one example of ET adoption is the testing of whole web site testing carried out in just two days [5]. Such a practice makes very serious concerns about the overall effectiveness of the testing approach. This may probably introduce increased work, problems and cost in the future, when the complexity of the application will need to be managed in a more structured way also in testing.

On the other hand ET cannot be applied to some applications, e.g. safety critical systems, where the testing procedure must strictly adhere to well defined procedure to document the testing done, in order to conform to given requirements, e.g. safety integrity level.

Side box: Systematic Literature Review

In order to collect evidence about the effects of ET on technical debt we conducted a systematic literature review. The goal we set for the SLR was to assess the potential negative effects of applying the ET technique.

We followed a simple procedure: (i) first we searched in the most important publication databases --- IEEE explorer, ACM digital library, Springer link and Science Direct -- with selection limited to software/computer science studies. The search was done using the keyword "Exploratory Testing" and the option "search in all fields" was used. At this stage we obtained 95 articles. Then we screened out the irrelevant paper first looking at the title and then reading the abstracts; at this stage we had 32 articles left. Eventually we read through the remaining papers and decided whether to include them on the basis of the actual content; as a result we included 8 papers. The number of articles retained at each stage and grouped by source database is reported in Table 1. The selection criteria we used in the above filtering were: (i) the paper deals with Exploratory Testing, (ii) it contains empirical evidence base on a valid study, (iii) the evidence concerns potential drawbacks induced by ET.

Table 1 Distribution of selected papers.

Source	Found	Scanned	Included
IEEE explorer	7	7	5
ACM digital library	47	8	2
Springer Link	39	15	1
Sciencedirect	2	2	0

While examining the articles we had a twofold objective: first, identify which are the areas of potential negative impact of ET and, second, verify the existence of empirical evidence supporting such a link. Table 2 summarizes the practices impacted by ET and the relative supporting evidence as revealed in the surveyed articles.

Table 2 Supporting evidence Vs practices impacted by ET

Empirical Evidence	Weaknesses				
	Test Planning	Test case Definition	Test result Assessment	Human Dependence	Documentation
Controlled Experiments	E1	E1	E1		
		E3		E2	
		E5		E4	
Interviews	I1	I1		I1	I1
Case Study			CS1	CS1	CS1
Action Research	AR1				

Surveyed Publications

E1 J. Itkonen, M. V. Mantyla, and C. Lassenius, "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 61–70.

E2 L. Shoaib, A. Nadeem, and A. Akbar, "An empirical evaluation of the influence of human personality on exploratory software testing," in Multitopic Conference, 2009. INMIC 2009. IEEE 13th International, 2009, pp. 1–6.

E3 T. D. Hellmann and F. Maurer, "Rule-Based Exploratory Testing of Graphical User Interfaces," in AGILE Conference (AGILE), 2011, 2011, pp. 107–116.

E4 S. Al-Azzani and R. Bahsoon, "Using implied scenarios in security testing," in Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, Cape Town, South Africa, 2010, pp. 15–21.

E5 L. H. O. do Nascimento and P. D. L. Machado, "An experimental evaluation of approaches to feature testing in the mobile phone applications domain," in Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting, Dubrovnik, Croatia, 2007, pp. 27–33.

I1 J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in Empirical Software Engineering, 2005. 2005 International Symposium on, 2005, p. 10 pp.

CS1 J. Itkonen, M. V. Mantyla, and C. Lassenius, "How do testers do it? An exploratory study on manual testing practices," in Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on, 2009, pp. 494–497.
AR8 J. Tuomikoski and I. Tervonen, Absorbing Software Testing into the Scrum Method, 2009. Lecture Notes in Business Information Processing, 2009, Volume 32, Part 4, 199-215.

ET as a source of Testing Technical Debt

Does ET represent an archetypal example of technical debt inducing practice? Shall it be repaid later in the application life cycle?

To answer these questions we conducted a systematic literature review concerning ET (see side box) and collected evidence about its weaknesses. We examined eight articles reporting empirical evidence: five controlled experiments (sources: E1 to E5), one case study (source: CS1), one interview (source: I1), and one action research study (source: AR1). We were particularly interested in those aspects that in the short term represent a cost saving but might imply a debt whose accrued interests ought to be repaid later. In order to understand how these weaknesses may induce technical debt we need to:

- identify the different aspects of testing that are affected by ET in a potentially negative way,
- understand the effects of ET on those aspects, and
- evaluate them in terms of technical debt.

We categorized the different type of debt in which testers might occur when practicing ET and, after we identified its general weaknesses, we mapped them to the technical debt instances. Hereinafter, we list the weaknesses of ET identified as a source of technical debt. We will discuss later on how to tackle the debt.

Test Planning

Typically test planning defines all those aspects related to testing strategy, resource utilization, responsibilities, risks, testing priorities, budget, and timeline [6]. Moreover planning is the prerequisite for monitoring and tracking the test activities, enabling awareness and visibility of the process.

ET effects: Different type of evidence is available in the literature (sources: E1, I1, AR8) on the lack of any planning of ET activities, given the vocation to personal freedom and creativity of this practice. ET lack of test planning results into having no control over the testing and increases the likelihood of double testing or overlooking important tests. Moreover the lack of plans makes the coverage of system functionalities unknown, therefore it is not possible to accurately estimate the overall quality of the system and make accurate plans for effort maintenance. In addition, due to the lack of planned responsibilities the tester's progress is difficult to track and responsibilities remain undefined. For example if the defects appear in a tested application during operations no one has the responsibility for that test.

TD implications: According to established approaches testing without a test plan cannot be managed in an effective way: for example, overruns of one or two hundred percent have been reported and test managers have difficulty understanding and monitoring testing as well [6]. Therefore, the higher cost of tests execution is the first type of debt introduced by ET. Yet another consequence of the lack of planning is the higher number of residual defects due to unmanaged functionality coverage and to the inappropriate handling of defects.

Test Cases Definition

A test case defines how the implementation of a given specification can be validated and typically includes preconditions, test steps, input data, and expected output. The documentation of test case provides a structure, guidance and traceability to the testing tasks [6].

ET effects: On ET, testing is based on simultaneous design and execution utilizing human intuitions; therefore it is performed without neither defining nor documenting the test cases. If we consider that we often need to re-execute the tests [S6] (e.g., to verify that a modification did not introduce new errors), the absence of documented test cases will make the re-execution quite hard or even impossible where tests were traced/recorded. Moreover, we found evidence that the lack of test cases implies that ET cannot assure 100% testing of all functionalities. There are evidences where functionalities remained without going under the course of testing. (sources: E1, I1).

TD implications: In ET, re-execution of tests (regression testing) is quite hard and expensive due to the absence of test cases. In addition to that, considering that some functionalities are not tested, residual defects could prompt in later phases like in production and maintenance, not only causing the malfunctioning of the system but also higher cost of defect removal.

Test result assessment

The outcome of a test is assessed in order to verify the correctness of functionality. A test oracle is required to determine whether the results of a program execution in a test are correct or not; an oracle can be just the expected result or a criterion to take a formal decision.

ET effects: In ET a formal oracle is absent and it is replaced by the tester's own judgment, therefore the evidence suggests that it is not possible to judge formally the correctness of an output (sources: E1, CS1). The main effect is the high probability of judging the incorrect behavior of an application as a correct one or vice versa. The evidence indicates [7] that these sorts of oracle mistakes are responsible for possible residual defects.

TD implications: The test result assessment based on the missing oracle in ET would result in additional rework due to more residual defects directly affecting the maintenance costs.

Human dependence

Testing as many software development activities is a human intensive activity. Naturally the outcome of any task depends on individual feature, e.g. skill, but too much variability may represent a problem. In particular we found evidence that ET is highly human dependent specifically on two factors.

A first factor is experience: different studies (sources: I1, CS1, E4) reported that ET is highly dependent on the experience of the tester. A second factor is personality extraversion: testers that possess an extrovert personality achieve the highest ET effectiveness (source: E2). Though, this is a trait that is not apparent and noticeable at the time of testing and selection of exploratory tester.

ET effects: Experience has positive effects on various parameters such as domain knowledge, speed of completing tasks, ability to identify meaningful patterns, superior recall [S7]. Further, experienced testers identify more potential categories of defects [S8]. This makes ET more a form of test for experienced staff. Therefore ET could be suboptimal if it is performed by some junior testers and possibly re-work or re-tests might be necessary.

TD implications: The major consequences of being ET highly human dependent might be the accumulation of residual defects due to a sub-optimal tester fitness and in general a non uniform test accuracy over the whole system. Thus the probability of rework to fix defects not found in testing is high and dependent on the difficulty of the fix task.

Documentation

The documentation covers all the required information related to system development; it includes typically preparation -- e.g. planning, requirements, and design -- and final documents -- logs and reports --. Not only insufficient but also unmaintained documentation may introduce weaknesses that are identified as a source of error [S9].

ET effects: In ET, there is typically not much available documentation. No test guidance is available and other testing artifacts are not produced; usually only failed tests are reported. ET limited documentation can provoke problems in the testing phase particularly because in following up a failure report is difficult to understand what has been tested and what has not (sources: I1, CS1) and in the maintenance phase too, where having no comprehensive documentation might slow down the activities.

Since ET test cases are born out of intuition and experience in tester's mind and there remain confined without any documented trace, other testers will not be able to reproduce them due to the lack of documentation and identifying the root cause of problems may result extremely difficult.

Such problems are even more pressing when a tester leaves the test team and new resources have to repeat tests or to perform regression testing: the lack of documentation makes knowledge transfer within the company very difficult or even impossible.

Moreover, insufficient or missing documentation might lead to difficulty in estimating the effort required in maintenance phase (or in allocating the proper time for testing) because no or inconsistent logs of past activities are available.

TD implications: The main consequence of lack of documentation in ET is increased and repeated costs in the maintenance phase. In addition, lack of documentation jeopardizes knowledge management in the company with additional costs for new testers. On top of those costs there might also be estimation errors in effort planning due to the lack of proper logs of the past testing activities.

Tackling the Exploratory testing debt

Figure 1 summarizes how exploratory testing influences the testing activities and what is the result in terms of technical debt on the basis of the SLR conducted.

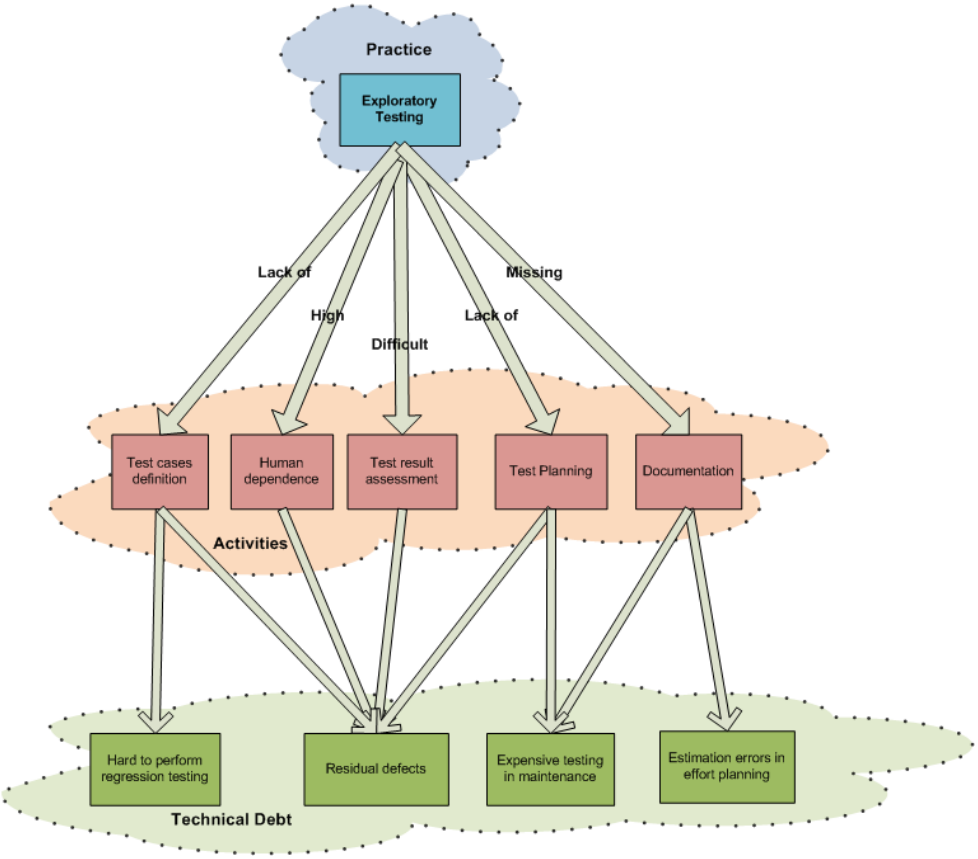

Figure 1 Induced TD by the implication of ET

In presence of technical debt, either induced by ET or other practices, the management ought to cope with it: that is paying the interests, retiring the debt, or converting it [2].

The first possible choice is to simply accept the very nature of the technical debt: trade smaller cost at present for larger costs in the future; that means paying the debt and the interests. The installments may take different forms that, based on the evidence discussed above, appear to be mainly: a larger amount of rework needed for defect removal, the allocation of unanticipated and thus more expensive tasks, additional difficulties in performing regression testing. Another immediate consequence of ET is represented by a higher number of residual defects, which also causes bad advertisement with the customers; this may represent a significant cost.

The possible alternative is represented by debt conversion that is transforming some part of the approach that brings immediate advantages but induces debt into a different one that implies more immediate costs but less debt. To achieve debt conversion, a basic transformation consists in providing ET with a semi-formalized structure that overcomes or partially reduces the weaknesses. We can see alternative for structuring the ET:

> *"Use ET in conjunction with other testing approaches where ET is unified by other approaches in formal software testing process"*

We will explain the solutions with a scenario: unifying ET with Requirement Based Testing [8] as a solution.

Requirements Based Testing generates test cases from a set of requirements [8] that may be used to structure ET in such a way that it would be documented and executed to test the basic requirements first. Afterwards ET would be performed in the usual fashion utilizing the human intuition and giving freedom to the testers to design their test based on the experience and on the fly, to identify more focused defects. This combination might lower the risk of having residual defects on the most used part of the systems (i.e. the basic requirements) but at the same time leaves the testers free to explore unusual scenarios based on their creativity. Moreover, the presence of a plan to test the

basic requirements might drive down the debt associated to the lack of planning: it would help controlling the status of the testing process, the chance of double testing and overlooking of important test would be reduced and performing the regression testing should be much easier. It also helps to assess the test results based on requirements and the functionality coverage of testing will be computed as percentage of requirements tested.

Considerations

- The key issue is the understanding of exploratory testing as an approach for testing. We emphasized only on the free exploratory testing which is most widely used as industrial practice [4]. So we enforce that using free ET could lead to such implications of TD. However using other styles of ET or ET conducted through session based management system might not have such implications but subject to the further direction of research.
- We emphasize our findings based on the academic literature supported by empirical evidences published in electronic databases. There is also much literature available online in forms of test blogs, wiki's and consultant websites. We didn't consider such gray literatures which are not supported through empirical evidences.
- We emphasize that the outcome of this study is just the hypothetical understanding based on the evidences from the literature. However we state that these hypothesis are ought to be verified further with empirical evidences.

Summary

Exploratory testing is widely used in the industry; practitioners view ET as a cost effective substitute for their daily testing activities where they are not bound to follow the structured and systematic way of testing. However, empirical evidence reported in the literature provides a more comprehensive picture of ET that takes in consideration also the technical debt implication of this technique. While ET has some manifest and immediate positive aspects and benefits, evidence suggests that it also brings some deferred drawbacks that might consist in a technical debt burden.

Such a picture allows practitioners to take an informed decision about ET adoption: therefore when planning to adopt ET, in addition to the specific benefits, testers and managers should be aware of the future TD that are packaged with the benefits of ET. Since, most of us need a mortgage to own a house, so most projects could not be successful without any compromises i.e. contracting some technical debt.

Any solution is a point in a continuum, at one end of the spectrum there is pure Exploratory Testing, cheaper in the short term, with no upfront costs, but bearing a significant debt, while at the other end is a structured and possibly automated testing approach, more expensive, with important upfront costs, and with a limited debt. The difficult work of the project manager is to find the right compromise that best suits the context of the project.

References

[1] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, Waikiki, Honolulu, HI, USA, 2011, pp. 35–38.

[2] F. Buschmann, "To Pay or Not to Pay Technical Debt," *IEEE Softw.*, vol. 28, no. 6, pp. 29–31, 2011.

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

[4] J. Itkonen, M. V. Mantyla, and C. Lassenius, "How do testers do it? An exploratory study on manual testing practices," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, 2009, pp. 494–497.

[5] J. Vaga and S. Amland, "Managing high-speed web testing," in *Software quality and software testing in internet times*, Springer-Verlag New York, Inc., 2002, pp. 23–30.

[6] "IEEE Guide for Software Verification and Validation Plans." 1994.

[7] A. Memon, I. Banerjee, and A. Nagarajan, "What test oracle should I use for effective GUI testing?," in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, 2003, pp. 164– 173.

[8] C. Pecheur, F. Raimondi, and G. Brat, "A formal analysis of requirements-based testing," in *Proceedings the international symposium on Software testing and analysis*, Chicago, IL, USA, 2009, pp. 47–56.

Appendix

List of assisting references:

S1 *S. Eldh, H. Hansson, Sasikumar Punnekkat, A. Pettersson, and D. Sundmark, "A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques," in Testing: Academic and Industrial Conference - Practice And Research Techniques, 2006. TAIC PART 2006. Proceedings, 2006, pp. 159–170.*

S2 *NIST-Final Report, "The Economic Impacts of Inadequate Infrastructure for Software Testing," Table 8-1, National Institute of Standards and Technology, 2002.*

S3 *F. Ricca, M. Torchiano, M. D. Penta, M. Ceccato, and P. Tonella, "Using acceptance tests as a support for clarifying requirements: A series of experiments," Inf. Softw. Technol., vol. 51, no. 2, pp. 270–283, 2009.*

S4 *C. Andersson and P. Runeson, "Verification and validation in industry - a qualitative survey on the state of practice," in Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n, 2002, pp. 37– 47.*

S5 *J. Itkonen, "Do test cases really matter? an experiment comparing test case based and exploratory testing". Ph.D thesis, Helsinki University of Technology, Finland (2008).*

S6 *M. J. Arafeen and Hyunsook Do, "Adaptive Regression Testing Strategy: An Empirical Study," in Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on, 2011, pp. 130–139.*

S7 *E. S. Sim, S. Ratanotayanon, O. Aiyelokun, and E. Morris, "An Initial Study to Develop an Empirical Test for Software Engineering Expertise," Institute for Software Research, University of California, Irvine, CA, USA, UCI-ISR-06-6, 2006.*

S8 *P. L. Poon, T. H. Tse, S. F. Tang, and F. C. Kuo, "Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing," Software Quality Control, vol. 19, no. 1, pp. 141–163, 2011.*

S9 *B. Muranko and R. Drechsler, "Technical Documentation of Software and Hardware in Embedded Systems," in Very Large Scale Integration, 2006 IFIP International Conference on, 2006, pp. 261–266.*

S10 *E. Egorova, M. Torchiano, and M. Morisio, "Actual vs. perceived effect of software engineering practices in the Italian industry," J. Syst. Softw., vol. 83, no. 10, pp. 1907–1916, 2010.*

Authors Biography

Syed Muhammad Ali Shah is a PhD candidate at Politecnico di Torino in Italy. His research focuses on the analysis of the software testing techniques/approaches, analysis of defect data to calculate the reliability, characterizing the impact of software factors on quality. He received his M.Sc. Degree in Software Engineering from Blekinge Institute of Technology, Sweden. He has research interests in software testing, software reliability, software process improvement and empirical software engineering.

Marco Torchiano is an Associate Professor at Politecnico di Torino, Italy; he has been post-doctoral research fellow at Norwegian University of Science and Technology (NTNU), Norway. He received an MSc and a PhD in Computer Engineering from Politecnico di Torino, Italy. He is author or coauthor of more than 90 research papers published in international journals and conferences. He is the co-author of the book 'Software Development—Case studies in Java' from Addison-Wesley, and co-editor of the book 'Developing Services for the Wireless Internet' from Springer. His current research interests are: design notations, testing methodologies, OTS-based development and software engineering for mobile and wireless applications. The methodological approach he adopts is that of empirical software engineering.

Antonio Vetro' is a PhD candidate at Politecnico di Torino. The main goal of his doctoral activities is to evaluate how automatic static analysis can be used to improve specific attributes of software quality, related to standard quality models such as ISO/IEC 25010 and SQALE. His recent interests focus on maintainability and performance issues in the general area of technical debt identification. The methodology he uses is that one of the empirical software engineering, and he is specialized on statistical techniques to analyze process and product data. He received a B.Sc. in Management Engineering and a M.Sc. in Computer Engineering from Politecnico di Torino. He has been a PhD visitor at the Center for Experimental Software Engineering at Fraunhofer USA (College Park, MD).

Maurizio Morisio is Full Professor at Dept. of Control and Computer Engineering of the Politecnico di Torino where he leads the software engineering group. In 1998-2000 he spent two years at the University of Maryland at College Park, working with the Experimental Software Engineering Group led by Vic Basili. From September 1998 to June 2000 he was on the board of directors of the SEL (Software Engineering Laboratory), a consortium among NASA Goddard Space Flight Center, the University of Maryland and Computer Science Corporation, with the aim of improving software practices at NASA and CSC. He got a Ph.D. in Software Engineering and a M.Sc. in Electronic Engineering from Politecnico di Torino. He has published more than 70 papers in international journals and conferences and three books.