Mälardalen University Licentiate Thesis
No. 78

On Evaluating Test Techniques in an Industrial Setting

Sigrid Eldh

2007

Department of Computer Science and Electronics

ISSN 1651-9256
ISBN 978-91-85485-68-0
Printed by Arkitektkopia, Västerås, Sweden
Distribution: Mälardalen University Press

"How to find a bug"

Artist: Johan Mauritzson

Utvärdering av testtekniker i industriell miljö

Teknologie Licentiatavhandling av Sigrid Eldh

Testning för att säkra programkvalitet är en viktig och kostsam aktivitet inom industrin. Antalet möjliga testfall i stora komplexa system är närmast oändligt, samtidigt som den tillgängliga tiden ofta är starkt begränsad. Det är därför viktigt att använda testmetoder som hittar många viktiga fel snabbt och effektivt. Tyvärr saknas riktlinjer för när det är lämpligt att använda olika testtekniker. Inte heller finns information om hur lätta teknikerna är att använda och automatisera.

Det huvudsakliga syftet med denna avhandling är att skapa ett ramverk där det är möjligt att utvärdera och jämföra testtekniker. Målet är att kunna utvärdera vilka testtekniker som fungerar bra i olika sammanhang. T.ex. för test på komponent-, integrations-, och systemnivå. En sådan utvärdering skulle ge värdefulla riktlinjer till industrin för vad som är snabba, effektiva och användbara tekniker i olika sammanhang. Nyckeln till framgång vid utvärdering av testtekniker är att kunna identifiera de olika typerna av fel som kan uppstå, och hur dessa fel resulterar i synliga symptom, dvs. felbeteenden. Denna avhandling belyser hur komplicerat det är att identifiera vilka fel som är representativa och som därför bör ligga till grund för jämförande utvärderingar. Avhandlingen presenterar ett antal frågeställningar som måste redas ut innan utvärderingar med generella och skalbara resultat kan genomföras.

Abstract

Testing is a costly and an important activity in the software industry today. The systems are becoming more complex and the amount of code is constantly increasing. The majority of systems need to rely on its testing to show that it works, is reliable, and performs according to user expectations and specifications.

Testing is performed in a multitude of ways, using different test approaches. How testing is conducted becomes essential, when time is limited, since exhaustive testing is not an option in large complex systems, Therefore, the design of the individual test case – and what part and aspect of the system it exercises, is the main focus of testing. Not only do we need to create, and execute test cases efficiently, but we also want them to expose important faults in the system. This main topic of testing has long been a focus of practitioners in industry, and there exists over 70 test techniques that aim to describe how to design a test case. Unfortunately, despite the industrial needs, research on test techniques are seldom performed in large complex systems.

The main purpose of this licentiate thesis is to create an environment and framework where it is possible to evaluate test techniques. Our overall goal is to investigate suitable test techniques for different levels, (e.g. component, integration and system level) and to provide guidelines to industry on what is effective, efficient and applicable to test, based on knowledge of failure-fault distribution in a particular domain. In this thesis, our research has been described through four papers that start from a broad overview of typical industrial systems and arrive at a specific focus on how to set up a controlled experiment in an industrial environment. Our initial paper has stated the status of testing in industry, and aided in identifying specific issues as well as underlined the need for further research. We then made experiments with component test improvements, by simple utilization of known approaches (e.g. static analysis, code reviews and statement coverage). This resulted in a substantial cost-reduction and increased quality, and provided us better understanding of the difficulties in deploying known test techniques in reality, which are described in our second paper. These works lead us to our third paper, which describes the framework and process for evaluating test techniques. The first sub-process in this framework deals with how to prepare the experiment with a known set of faults. We aimed to investigate fault classifications to get a useful set of faults of different types to inject. In addition, we investigated real faults reported in an industrial system, performed controlled experiments, and the results were published in our fourth paper.

The main contributions of this Licentiate thesis are the valuable insights in the context of evaluation of test techniques, specifically the problems of creating a useful experiment in an industrial setting, in addition to the survey of the state of practice of software testing in Industry. We want to better understand what needs to be done to create efficient evaluations of test techniques, and secondly what is the relation between faults/failures and test techniques. Though our experiments have not yet been able to create ‘the ultimate’ classification for such an aim, the results indicate the appropriateness of this approach. With these valuable insights, we believe that we will be able to direct our future research, to make better evaluations that have a larger potential to generalize and scale.

Breathing exercise from
"Peace is Every Step
The Path of Mindfulness in Everyday Life"
by Thich Nhat Hanh

Breathing in, I calm my body.
Breathing out, I smile.
Dwelling in the present moment,
I know this is a wonderful moment!

Till Per, min älskade

Preface

When I was five years old, I said I wanted to be a researcher. My love for horses moved me into science, and I ended up at Uppsala University for my Masters degree. I have really enjoyed my work career, most of them related to improvements in software quality or people. For more than 20 years, I have built my know-how in testing strategies and approaches, processes, methods with the basis of solid understanding of software and its problems. In 1994, I started at Ericsson and short thereafter finding my love Per and moving to and back from Australia was an adventure! Coming back home to Sweden, we both returned to Ericsson. The large, complex system and great mix of co-workers at Ericsson have challenged my skills to develop further. My work in all different positions, has led me back to where I started – to my own Mount Everest – to become a PhD. Without the support of my dear professors, mentors and supervisors Hans Hansson and Sasikumar Punnekkat, I would never have been able to get to base camp. Amazingly, the Mount Everest has shrunk to Kebenekajse, or maybe even Norra Stadsberget.

I also would like to acknowledge Ericsson, The Knowledge Foundation and Mälardalen University for all support in funding this research. Thanks to all of you people who have encouraged me along this path; friends, colleagues and family. Others who really deserve thanks for different moments in this climbing are: Joachim Wegener, Ivica Crnkovic, Henrik Thane, Lars-Olof Gustafsson, Anders Pettersson, Daniel Sundmark, Kenneth Malmqvist, Torbjörn Nyman, Ingvar Nordström, my dear mother Gudrun Eldh and my creative brother Johan Mauritzson, who also contributed with the drawing. Finally, I want to thank you, my dear husband and soul mate – Per Karlsson. I would not be where I am today without you. I'm looking forward to continue our great life journey together – Thank you for all the love.

Disclaimer: All claims in this thesis are solely my responsibility, and do not in any way represent Ericsson's, Mälardalen University's or the Knowledge Foundation's official view. The papers in this thesis have been altered in format but not in content from their original publishing.

Sigrid Eldh, Älvsjö, December 2007

List of Publications

Publications included in the thesis

Paper A: Eldh, S.: How to Save on Quality Assurance – Challenges in Software Testing, Jornadas sobre Testeo de Software, p 103--121, ITI, Universidad Politecnica de Valencia, Valencia, Editor(s): Tanja E. J. Vos (2006)

Paper B: Eldh, S., Punnekkat, S., Hansson, H.: Experiments with Component Test to Improve Software Quality**,** *International Symposium on Software Reliability Engineering (ISSRE)- Industrial Track*, IEEE, Trollhättan, Sweden (2007)

Paper C: Eldh, S., Hansson, H., Punnekkat, S., Pettersson, A., Sundmark, D.: A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques. *Proc. TAIC*, IEEE, London, UK (2006)

Paper D: Eldh, S., Punnekkat, S., Hansson, H., Jönsson., P.: Component Testing is Not Enough - A Study of Software Faults in Telecom Middleware, *Proc. 19th IFIP International Conference on Testing of Communicating Systems TESTCOM/FATES*, Springer LNCS-4581, Tallinn, Estonia (2007)

Relevant Publications, not included in the thesis

Papers

- S. Eldh: *Licentiate Thesis Abstract: Evaluating Test Techniques using Fault and Failure Analysis*. 7th Conference on Software Engineering Research and Practices in Sweden, Gothenburg, Oct (2007)
- Sundmark, S. Petterson, A., Eldh, S., Ekman, M., Thane, H.:*Efficient System-Level Testing of Embedded Real-Time Software*, Work-in-Progress Session of the 17th Euromicro Conference on Real-Time Systems, p 53-56, Palma de Mallorca, Spain, Editor(s):Isabelle Puaut, July (2005)
- S. Eldh: *Making a Difference with Test Assessment*, Proceedings of the 10th Int. Conference, STARWest, Anaheim, CA, USA, (Nov 2002)
- S. Eldh: *Use and Misuse of Software Metrics*, 3rd Int. Conference ASMA, Melbourne, Australia, Nov (1996)
- S. Eldh: *The Challenge of Test Automation – a Process Change*, Proceedings of the 13th International Conference on Testing Computer Software, Washington DC. USA, June 13-16 (1996)
- S. Eldh: *Reaching Calm from Calamity*, Proceedings of the 11th Int. Conference on Testing Computer Software, Washington DC, June (1994)
- S. Eldh: *Automatiserad testing*, Proceedings of Software '94, Den Norske Dataforening, (1994)

Book

- S. Eldh: *Operativsystem & Datorsystem*, 526 pages, Studentlitteratur, ISBN 91-44-24131-3, (1987)

Conferences: Keynotes, Workshops and Tutorials

- S.Eldh: *Creating a Company Wide Measurement Program for Verification*, Seoul, International Software Testing Conference (ASTA), South Korea, October (2007)
- S.Eldh: *Achieving Quality – Efficient Verification of Telecom Systems*, ESQAT, Beijing, China, September (2007)
- S. Eldh: *Towards Testing Excellence***,** Proceedings of the I&V Conference, Ericsson, Linköping, September (2005)
- S. Eldh: *Testability – Knowing your tests***,** Proceedings of the 11th Int. Conference EuroStar, Köln, Germany, November (2004)
- S. Eldh: *Software Quality Rank- Improving Designers Test*, Tutorial of the 11th Int. Conference EuroStar, Köln, Germany, November (2004)
- S. Eldh: *Software Quality Rank – An Improvement in Component Test***,** Proc. of the International Conference ICSTest, Dusseldorf, Germany, April (2004)
- S. Eldh: *How to write an effective Test Plan according to IEEE Std 829***,** Proceedings of IIR Conference Testus, Helsinki, Finland, March (2003)
- S. Eldh: *To be a Leader in Testing***,** Proceedings of IIR Conference Testus, Helsinki, Finland, March (2003)
- S. Eldh: *Test Assessments based in TPI***,** Proceedings of the 10th Int. Conference EuroStar, Edinburgh, Scotland, November (2002)
- S. Eldh: *Software Testing Issues and Challenges from an Ericsson Perspective***,** Seminar, ARTES, Stockholm, Sweden, August (2002**)**
- S. Eldh: *A Successful TPI Assessment***,** Proceedings of the Advanced Testing Conference, Helsinki, Finland, March (2002)
- S. Eldh: *How to improve Dynamics in a Test Team***,** Proceedings of the Advanced Testing Conference, March 2002, Helsinki, Finland
- S. Eldh: *From Art of Software Testing to Leadership***,** Closing Keynote of the 9th Int. Conference EuroSTAR, Stockholm Sweden, (2001)
- S. Eldh: *Successful Test Strategies***,** Proceedings of the 9th Int. Conference EuroSTAR, Stockholm, Sweden (2001)
- S. Eldh: *Measurements in Testing***,** Full day Tutorial, 9th Int. Conference EuroSTAR, Stockholm, Sweden (2001)
- S. Eldh: *Testning förr och nu*, Proceedings of IIR Conference "Testning, Verifiering och Test Strategier", incl. Workshop on Test Strategies, Stockholm, Sweden, May and Sept (2001)
- S. Eldh: *Testmodeller*, Proceedings of IIR Conference "Testning, Verifiering och Test Strategies", incl. Workshop on Test Strategies, Stockholm, Sweden, May and Sept (2001)

- S. Eldh: *Testdomäner*, Proceedings of IIR Conference "Testning, verifiering och Test Strategies", incl. Workshop on Test Strategies, Stockholm, Sweden, May and Sept (2001)
- S. Eldh: *Testning i framtiden*, Proceedings of IIR Conference "Testning och verifiering och Test Strategies", incl. Workshop on Test Strategies, Stockholm, Sweden, May and Sept (2001)
- S. Eldh: *Test Metrics - To measure Test*, SAST Conference, Stockholm, Sweden, Feb (2001)
- S. Eldh: *Why is automatic testing important today*: Proceedings of Sweden Engineering Industries, (Verkstadsindustrier), Conference on "Automatic testing with Daily Build", Stockholm, Sweden, Nov (2000)
- S. Eldh: *Performance Testing*, Proceedings of the 7' th Int. EuroStar, Barcelona, Spain Dec (1999)
- S. Eldh: *What is Next in testing*, Keynote of the 7th EuroSTAR, Barcelona, Spain (1999)
- S. Eldh: *Developers in testing* Keynote of the 7th EuroSTAR, Barcelona, Spain (1999)
- S. Eldh: *Advanced Testing in Telecommunications*, 5th Int. EuroSTAR, November, Edinburgh, Scotland (1997)
- S. Eldh: *Essentials in OO-testing*, Proceedings of the 4th Int. EuroSTAR, Amsterdam, Netherlands, Dec (1996)
- S. Eldh: *Helping Developers Do it better in an OO environment*, Proceedings of the 3rd Int. EuroSTAR, London, U.K, also Proceedings of the 5th Int. STAR, 2-5 May 1996, Orlando, Florida, USA, Dec (1995)
- S. Eldh: *Testing with Third Party Products*, Proceedings of the 3rd Int. EuroSTAR, London, U.K, Dec (1995)
- S. Eldh: *Hur görs en ändring i praktiken - de sex faserna*, Proceedings of "Systemförvaltning för ökad kvalitet", (NFI) Stockholm, Sweden, Mars (1993)

Other relevant work for this thesis

Contributing Master Theses supervised by S. Eldh, who created thesis work proposals, approach and context in addition to normal supervision.

- Larsson, M.: *Evaluation and overview of test techniques and their applicability in different real-time systems* IDE, MDH (2004)
- Stenmark, J., Bokvist, H.: *Analysis and evaluation of the method Software Quality Rank on component test* IDE, MDH (2004)
- Albertsson, D., Sandberg, P.: *An Evaluation Model for Selecting Test Tools*, IDE, MDH (2005)
- Jönsson, P.: *Analysis and Classification of Faults and Failures in a Large Complex Telecom Middleware System* IDE, MDH (2007)

Contents

Chapter 1 – Introduction

1.1 Factors that Impact Quality in Large Complex Systems

Software testing is without doubt a cost consuming standard practice in industry. The tolerance of faulty software differs for different industries and domains. We accept trains to be stopping, when there is a signaling fault in the system. On a telephone system, we rather allow some occasional loss of voice quality rather than loose the entire call. Loosing a call is not considered a disaster. If you are calling from a train and you are losing your mobile call, you just call again some seconds later. Yet, the impatience from a customer when loosing a call or sitting still and waiting for a signal are both qualities indirectly related to testing and cost. We can build a system to be fault tolerant and also "fail-safe" but it usually has its price.

Systems are now growing in size and are becoming more complex, with more features. We also use software to make sure that software systems are designed, developed and tested as efficiently as possible. Development of safety critical systems requires "flawless" software. Component based software is dependent on high quality components that work in most situations, hence are possible to reuse. Testing is often performed to achieve several goals, such as showing that major functionality works, minimizing impact in maintenance (costs), achieving measurable reliability, and contributing to improved software quality.

Regardless of the industry; electronics, embedded, automotive, process, telecommunication, banking, insurance or services, the development and maintenance of software has strong impact on the profit of these companies. Companies are currently faced with several issues related to software;

- Legacy code that is contributing to increase the size of software
- More frequent and parallel updates
- Dependency on third party software and hardware
- More concurrent software and hardware
- More fault-tolerant and fail-safe code
- Code containing security measures
- Self correcting code
- Additional operation and management features and functions built in e.g. to
    - measure performance
    - maintain and optimize the software
    - report about hardware status
    - handle all kinds of operation

This can be summarized by: "Software gets more and more complex". In addition to this, the people aspects have also grown more complex. Software companies use a larger diversity of people from all over the world in all aspects of development, testing, and maintenance. The retention of people, in addition to comprehensive skills and a broadened knowledge base will impact quality, when working with complex software.

In this context, the business aspects are also getting harsher: penalties are written into contracts, which must be paid if software systems are not delivered on time with the promised quality. The wish to deliver faster and faster, with sustained and

improved quality, is a time to market race, where market shares and showing know-how makes an impact on the companies' profits. All software producing companies have an underlying requirement to deliver quality products. Yet, what this actually means is difficult to define, but the consequence - if the product does not live up to the expected quality - is that the costumer will grow impatient quickly and leave for another supplier.

1.2 Testing to Improve Quality

Most software can be fixed, but the cost associated with maintenance and support is substantial. For instance, the cost of re-testing could easily be 90% of the actual maintenance cost. How to ensure quality software is an area that deserves more attention, and this is the context that motivates our work. The focus has long been on development of "fault free software" (e.g. through formal methods, new and novel architectures, development models and new "fault free" languages), but the reality is that people do make mistakes, regardless. In industries, the cost of software testing is usually between 40-80% [1] of the total cost of development, where safety critical systems are on the high end. According to NIST [1] failures in software is costing the US society about 80 BUSD annually. Depending on the nature of the product, software industries are paying the consequence with strategies to minimize maintenance. When you test or just execute the software, you will be able to show that the software works for the particular path with that particular input, in the particular context. In real-time systems, this does not necessarily mean that the particular tested path always works. Instead, you must be able to consider all aspects that can impact this result, such as resources, timing, scheduling and external interactions.

It is difficult to prove the absence of faults in software [2], and it is possible in only very small limited software – and yet you can never totally rely on the system, hardware and context in which the software executes – which can cause a "fault free" software to skip a line. The truth is that we are becoming good at building systems that have double, triple or multiple redundant hardware support. We often over-dimension the hardware, we are creating software to handle failures so we do not experience them as a user – and we create software to handle a multitude of problems.

The better we become at creating support for masking failures – the harder the testing of these systems become. We have not investigated enough in research and in industry how you can actually minimize the effort of testing; instead the trend seems to be that industry is taking more and more risks in some areas. The balance is delicate, and we risk exposing complicated systems to a multitude of vulnerabilities. We know that more tests are likely to find more faults, but we do not have endless amount of time to test commercial systems completely. This scenario is the setting of our research on testing that is suitable for industry. We need to focus on how to make software systems more testable, how to create efficient (automated) testing and how to aid testers to be more effective. Hierons [3] described in his keynote of the TTCN-3[1] conference 2007 that the most important task between industry and research is to

[1] TTCN-3 is a language and an ETSI standard, aimed to write test scripts for automatic execution on software

provide real guidelines for industry based on the research invented. This independent and recent comment justifies and confirms that our main goal has been accurate and timely as an important research area.

1.3 Faults and Failures, and their Relation to Testability

In telecommunication systems (and academic research), overcoming individual failures and minimizing the impact of faults have long been in focus, in order to provide system improvements. Overcoming faults has developed telecommunication systems to be distributed, fault-tolerant and self-organizing software systems, with many levels of management (architecture) to compensate for flaws. Other techniques used to achieve high service without failures are by mirroring or adding extra hardware. In addition to minimizing single-points of failures in the system, system architectural techniques such as fail-over are used. The analysis, development and complexity of these types of systems are overwhelming. Not to mention the problems of verifying and testing them, since the systems' goal is to eliminate the effects of faults by hiding and encapsulating them. This means in practical terms, that faults reside in the system, but seldom propagate to visible failures. One could argue, that faults that do no propagate do not cause any havoc, but that is not true in the long term. The more the software is utilized, re-used, and updated with new hardware, the more these faults get exposed, and the cost of maintenance will not diminish with new releases.

A consequence of faults not propagating is that testing becomes a more difficult task. A contradictory goal to fault hiding is the aim to make faults more visible. We would like to make faults visible and to propagate into visible failures, since this would improve the ability to find them, and thereby improving testability as well. In short, we need the system to be easily *testable* to be effective. Testability is defined by W. Schütz [7] by the terms *observability* and *controllability*. A system/software is testable if it has high observability and high controllability. Observability is defined as "how well you actually can measure what you aim to measure, and get a result", since testing is a "measurement" of the system (or software) quality. Controllability is defined as how repeatable the result is (i.e. that you can do the measurement again and get the same result) and how possible it is to actually measure the specific aspect. This is of course very important, that a test executed will yield the same result (pass of fail) every time. The current complex telecom systems suffer partly from low testability, due to low observability and to some extent, being real-time systems, low controllability.

In this thesis, we are approaching the subject of testability from another viewpoint. Despite our research, we cannot help notice testing trends that are predominant in industries today. Simple test methods and approaches prevail, and these do not change fast. In many systems, emphasis is spent on agile, fast and lightweight processes, methods that aim in minimizing testing effort and eliminating any formal or structured approach, instead of using more rigorous testing (i.e. how to perform test correctly and efficiently). This phenomena is easy to explain, since there exists few proofs that structured techniques do outperform ad hoc techniques. Unfortunately, these test approaches are often dependent on an "expert" performing them to yield that fantastic result. The "expert" gladly claims how easy it is to do it his or her way, and is influencing testers not having domain knowledge, resulting in

less impressive results. That is why we need to spend time on providing better research on industrial systems, thus creating guidelines for professional testing, i.e. testing that could provide good results even when performed by testers lacking in-depth domain expertise. In addition, we need to know how to perform this evaluation to create results that can be generalized and are scaleable.

1.4 State of Practice of Testing in Industry

An interesting observation is that most software industries have a large number of specialized testers, excellent in understanding the system, its expected behavior, and the context where it is working. The necessity of testing is never or seldom questioned in a mature software development company, but how it is performed – and we might add - how effectively and efficiently it is performed - is not fully explored. Testing methods used in industry are by and large heuristic and based on ad hoc principles. Most companies base their test cases on requirements, and the test cases are in most cases functional. The requirements vary in quality, level, concepts, and efforts are made to make them clearer and more testable. The problem is that requirements can never capture all aspects of a complex system. In fact, much of the system descriptions can be found in implementation proposals, software design documentation, and interface descriptions or in the actual code. Many of the aspects defined in the documents, are never transformed into usable test cases – even if they easily could be. At least *reviewing* is a technique widely used – even if the efficiency and motivation of performing reviews vary over years. It is not the phenomena of doing the actual review (which is of course better than doing nothing) that will automatically yield a better quality. To our experience, effective fault finding depends greatly on *how* you perform reviews, your experience and skill in this technique.

Requirements are still not enough to base test cases on. Yet the majority of the industries only base their tests on the requirements, and this is clearly insufficient. We definitely believe that the testing should be performed at many levels if you aim to capture a majority of the faults and failures. By many levels we refer to the levels described in the V-model, see Figure 1. In our path to understand test techniques and their usage, we have explored the possibility to improve developers testing, at the component test level. We had the opportunity to introduce some added testing techniques, as well as give emphasis on some of the already known techniques and approaches. This attempt was a great success [A][B], which is encouraging. We have compared 23 design teams and their usage of our approach, called SQR (Software Quality Rank) in [B]. This approach substantially reduced the costs of testing, in this particular case, but also gave us valuable insights in the problems of deploying test techniques in reality. Regression tests represent the “motorway” through the software to capture the normal use by repeated execution. Yet, when systems are combined with other systems, new failures are exposed. This happens when new features are added or when new and faster hardware is added. In evolving systems, the users are constantly challenging the system outside those “motorway” paths. In fact, exposing systems on internet makes any fault a security threat – which poses new challenges to the software. This makes systems more vulnerable, and the need for more detailed and rigorous testing is increasing.

1.5 Test Design Techniques

Unfortunately, there is very little evidence that shows that systematically applying a test design technique is in any way more efficient to find faults than currently used techniques in the industry. Yet, the industrial approaches to testing are not clearly investigated either. For every technique that is new in some aspect in the approach applied, you are likely to find a new range of existing failures. Do you find more failures than you would have, if you ONLY tested according to a particular technique? Probably not, but there are few new commercial systems that rigorously approach testing only by using one or a few techniques, and these are usually not available for comparative studies. This is why these guidelines and their relation to the fault and failure is such an interesting focus area. Another factor influencing this discussion is what particular technique we have in mind. There are many test techniques to choose from at any given time. Which of them are best and in which context? This is exactly our research questions and motivation. We suspect that the test oracle, the human, selects much better test strategies than a purely random approach, i.e., better in the sense that the failures found are more important for the product. Yet it would be great to have better knowledge on what faults a certain testing technique can expose, and when in the development process to apply which technique. There exists at least about 29 main test techniques groups and as many as 70 or more test techniques and the number could easily be increased depending on how we define a unique test technique [13]. We have briefly looked at the test design techniques used in industry (by studying test cases), and only to a small degree were test techniques systematically applied. We also found that by expanding existing test cases by aiming to apply one technique to unstructured test cases, improved the ability to find faults [4]. In conclusion, our experience is that testers are often not trained in testing, but on system behavior. Even if people are formally trained in test techniques, they easily fall back to approaching testing from a system usage viewpoint rather than applying a test technique, since requirements on testers are seldom assessed as long as they find some failures.

A common view is that if you apply a new test technique then you will find new failures. DeMillo [8] claims that any new technique or test approach will have peak failure finding abilities for a while. This motivates teaching of several techniques and invention of several techniques. James Whittaker [5] claims that even if you do not apply any technique at all, you will sooner or later stumble across a software failure by mere execution of the system. The ideal solution would be to start applying one test design technique on industrial software with the result that some failures would be found. If the software had many faults, any additional technique would find additional failures. Would this prove that one technique is effective and efficient if it finds more faults “faster” than another one? Whether this would make any two test techniques comparable is the main question that we aim to answer in our PhD research. The techniques have to be compared for efficiency and also regarding how important failures they find. We would also like to tell which of the testing techniques should be preferred, when faced with limited time and resources. That a testing technique should find failures more efficiently than ad hoc testing is only supported by common sense, but to our surprise, there is very little scientific evidence that supports different testing techniques and their efficiency. So what

makes a system tested: 600 test cases or 60000 test cases? There is no right answer, since it is not reasonable to measure stopping criteria for large complex systems, except with very coarse and crude methods. Testing for failures to find the same fault injected in two different systems is not necessarily the same. Therefore, to better understand how to select the best test approach, and how to apply it, at what level and to what characteristics of the system, is one of our main goals. The problem is which one of all techniques or approaches are to be executed first, and in what order to use other techniques, to assure the best testing possible in the limited time available for industry.

Testing techniques find failures, or provide evidence of their absence, but only for that selected path executed in the same context. It should typically not be possible to cause a system going into a failure state by providing a faulty input. Surprisingly, this is not the case in reality and many testing techniques rely on malicious inputs to cause failures. Fault handling code is often less tested than the normal case or main paths.

1.6 Test Process - Test Levels and Test Phases

We use a simplified Test process model (V-model), which of course could be used iteratively given different scopes at given times. The phases of the process model always exist in the software life cycle, regardless of how fast, or how many iterations you perform within a project. Often names are adapted depending on size and emphasis. The phases are:

- Requirement & analysis phase, also including test requirements and test analysis
- Design phase which can be divided into high-level and low-level design, and also test design
- Implementation phase (that sometimes include component test), including both creation of code, documentation, and test code for automation (or test procedures for manual testing)
- Component test phase
- Integration phase, which can be divided into high-level and low-level integration including integration tests
- System test, where tests requirements often are separated in internal phases called function test phases (functional test) and a characteristics test phase (non-functional tests)
- Acceptance test (for maintenance and/or customer release)

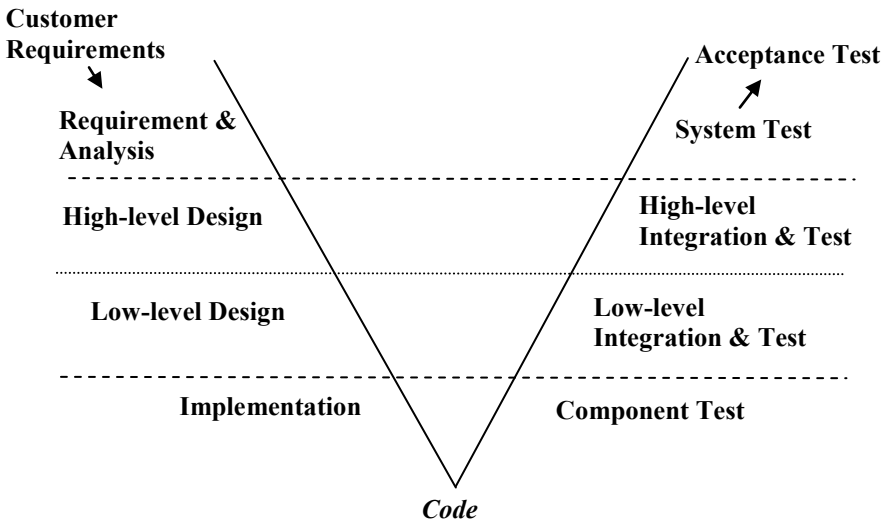

Figure 1. V-model simplified

Requirement & Analysis phase
In the Requirement & Analysis phase, the testing is static using review and inspection techniques. There is no hindrance in creating test cases already at this point, based on requirements or other information about the system. Creating executable tests as early as requirement phase is referred to as "test-driven development". Depending on analysis techniques used, it might also be possible to derive test cases based on the technique: e.g., using user-scenarios. These can be used directly to define the end to end test techniques, creating use-cases according in a more formal sense, e.g. using UML-diagrams.

In this phase, it is also important to create the test plan according to IEEE STD 829 [14], including phases, test criteria, resources, what to test and not to test etc., as well as to gather requirements particular for test, such as test tool planning, test environment planning etc. For our research, we suggest that this phase is a good time to analyze the failure and fault distributions in a manner described in Paper D, based on earlier versions and compare to earlier test strategies.

Design Phase
The design phase is of course where the architecture of the system is designed, and testability must at this point be built into the structure. This is also the time when test techniques should be applied to create test specifications according to the test strategy. At the design phase, where it is possible to use modelling, test cases can automatically be generated from the model or created semi-automatically. Testing in the implementation phase is crucial in all aspects; this is the time when the code is created. We have in Paper C given some information on phases of the test automation to take into account. At the implementation phase, static tests in the form of design and model reviews can be performed.

Implementation Phase
The implementation phase is where the design is implemented into code, and structured into programs according to the architecture. At this phase, the test teams are also preparing the tests, either by describing how they are manually performed in test procedures or by defining test scripts to be executed by a test tool. Here also techniques like static tests, e.g. desk checks and code reviews, can be performed, in addition to using static analysis tools.

Component Test Phase
Often the component test phase is hidden in the implementation phase. Therefore, it is a good idea to highlight component test, so that it does not get lost. The component test phase is a crucial step to make sure software parts are reliable. In paper A the benefit of Software Quality Rank (SQR) is described and how to perform it is described in Paper A and B. SQR consists of steps for areas like static analysis and dynamic analysis [24], dynamic execution in addition to review, but also defines implicit quality improvements like demanding sufficient documentation.

Integration Test Phase
Applying good integration tests are difficult and require careful analysis. Here, tests should already have been prepared for execution. One way to minimize the impact of complexity is to integrate and test bottom up. It is possible to minimize late integration problems by performing early integration with daily builds since large complex systems are then always tested in the right context. This requires a comprehensive component test, which we showed in Paper A and B. The problem with daily builds is that it can make the system completely unstable, but it opens the possibility to early debug and trace the system. The advantage is that now you have the entire system at your hand and integration problems will be found early, and you will not only address a limited part of the system with a limited effect. Note that these system might have more faults hidden if not component test is done sufficiently.

System Test Phase & Acceptance Test Phase
These phases test the entire system, both functional and non-functional aspects. The difference between the system test phase and the acceptance test phase is in the goal of the testing and the depth of testing. In each test execution phase, any group of test cases can be performed: structural, functional and non-functional tests.

At all above test levels, test results are collected, analyzed and reported, which serves as a measurement of the quality of the software, and also aids in improving the quality by targeting areas to correct. No specific test project management aspects have been discussed either.

1.7 Terminology

Terminology in Software Testing is confusing, since many different standards have conflicting or just different definitions. We are following the terminology of IEEE

standards of glossary (and in particular IEEE Std 829 [14], BS 7925-1, and ISTQB Glossary [25]) except for the area of fault – error –failure, where we are following the IFIP definitions by Avižienis and Laprie [18]. In each paper, we have defined and clarified the terminology used within the context of the paper.

1.8 Research in an industrial setting

We can confidently say that we understand the need to simplify, minimize and abstract information from real systems to be able to perform research. The scale of industrial and commercial software systems surmounts the time and detail possible to encompass within one PhD time frame.

For our type of research, we need a long term focus, since creating an industrial guideline requires thorough research. To perform experimental research on "real" software from industry creates a lot of difficulty. Introducing failures in commercial systems to study them is just not an option, and software needs to be separated to allow experiments. An alternative would be to use a small fraction of the software to experiment on. Unfortunately, "small" software do not execute sufficiently without its surrounding, and using simulators will not yield the same test results, which makes the results dubious. This meant that we needed to have some form of a subsystem that can execute real industrial and commercial software. In telecom, these systems are often large and complex to use.

Our preparations of creating this controlled research experiment have led us to move an entire telecom node to a secluded environment to allow us to perform our experiments. It is difficult to use the node and also to utilize it to its full potential. We have learned that we need extensive knowledge of the actual system and this makes the work even harder. A second important factor when performing research is the "observability" in the software. Observability is how easy it is to "see" the characteristics you are looking for. For us, this means if we introduce faults into the code, we want them to be "visible" in some form at execution of the software. We are experimenting further in this direction, trying to establish different type of systems' "observability" factors. We want to see what makes a fault to propagate into a visible failure. We conclude that it is very easy to understand why most research in this area is using the prepared Space programs or the Siemens suite, which exists in a combined set with additions, in the system SIR [6]. We could use SIR to compare our research results, but our main concern is that in the Siemens suite, there is only one fault injected in each program, which does not reflect the reality of most programs. We also feel that the type of faults in the SIR system are too few, and too simple in semantics, compared to our experience, as we partly have described in Paper D. Another motivation for not using SIR is that we wanted to base our research in a real industrial setting, and not in a pre-prepared environment.

Chapter 2 - Research Motivation

2.1 Main Motivation

The main motivation for this research is the need for guidelines in industry to determine what test techniques are efficient, effective and applicable. In addition, we hope to gain insight in what level or phase of testing these techniques are applicable, how much human interaction is needed – and what can be automated in all phases of applying a test technique in test design. To gain this insight, it has been obvious that for a fair juxtaposition of test techniques, all types of faults should be available in the experiment to give all test techniques a chance to show its applicability. Equally important is that by analyzing faults in real systems and understanding their frequency and distribution, we will be able to obtain more generic results transferable to other systems. Therefore, we have spent more than a year trying to create an experiment that has a wide variety of faults injected in real (industrial) software. As a bi-product, we have also learned a lot about footprints of faults, fault-injection, and fault and failure distribution.

2.2 Hypothesis for PhD and Ongoing Research

The main hypothesis for the Ph.D. research can be stated as:

'Given a failure-fault history of a system, it is possible to determine the efficiency, effectiveness and applicability of a test design technique at a specific level and specific phase for similar systems'.

This main hypothesis can be divided into several problem areas, as explained below, where the first part is addressed within this licentiate thesis work.

2.3 Problem Areas

Through the main hypothesis, we are trying to show that it is possible to compare test techniques given some information, but does not say how. Herein lays the problem. It is relatively straightforward to make results for one specific code in an instance of a specific application, but our ambition is to try and make results more general and scalable. Below are the sub-hypotheses or "problem areas" we need to address and answer, describing what we need to provide to say we have proved or disproved the hypothesis above.

1. *Fault-failure history analysis*

To be able to draw conclusions from the fault-failure analysis, we need to know distribution and occurrence. The main question is: What are the interesting faults to inject into software to base the experiment on, and be able to draw conclusions based upon, that makes it possible to generalize the result to other software? The work in Paper D, have given us insights in some of the questions that needs to be answered.

2. *Domain independence*

To be able to draw conclusions on whether our result has domain independence, we need to understand what factors makes the results useful for other domains, i.e. generalized and scalable. We aim to approach this problem in a three-phased manner starting from a single system to across systems within a domain, and then across domains.

It should be possible to analyze software and system through its fault-failure history and answering the above questions and drawing conclusions on how that affects the result in testing. We need to understand which part of our results can be generalised and which cannot. Based on experience we can see that the observability is different in different type of systems as well as how easily some faults propagates into failures in some types of systems compared to others. At this point in time, we are not willing to draw any conclusions or make any proposals regarding in what way the domain will impact the result.

3. *Efficiency, Effectiveness, and Applicability*

This problem area has three major issues to be resolved. First of all, we need to define precisely the meaning of efficiency and effectiveness in the context of testing. Next we need to decide or design appropriate measurement techniques/approaches for evaluating them. Finally, we need to perform experiments and ascertain their value in real industrial settings. We have suggested a set of measurements in Paper C that would be possible to use to measure test techniques' efficiency and effectiveness. We have not yet had the possibility to test these measurements awaiting our experiments to be available. But as discussed in future work, we have also identified certain approaches for conducting this.

The area of applicability – or usability of a test technique - opens a wide set of questions that needs to be answered to be able to continue our research. We believe this approach is novel and will cast light on test techniques in a new way. Not only have we in Paper C identified a set of measurements to be used, but also we hope that partitioning the test design and execution process will aid in focusing on the right problems.

Chapter 3 Research Results

This licentiate thesis is focused on analyzing the industrial state of practice, and then defining a process to evaluate test techniques for efficiency, effectiveness and applicability. In principle, the contribution of creating such a process may not appear novel at the outset, but on a detailed level, the applicability aspect of test techniques is a new dimension, which has been unexplored hitherto. In particular, how to create guidelines out of research, so results are both scalable and re-usable, is not an easy task – and the approach to applicability will contribute to better use of the techniques. The focus of this thesis is not so much on finding a lot of data and producing ready to use results, but rather to ask the right questions that will take us closer to the answer.

The knowledge needed to be able to ask the right questions is the main achievement, which challenges most previous test technique evaluations to the core. Our results are defining the areas that need to be evaluated further, to be able to perform better evaluations, where the result would hold outside the scope of the study. In particular, we define a series of questions in different areas that needs to be answered to create a solid controlled experiment that have the potential to scale and yield results that are more general.

Our approach in research has opened up a series of key areas for further studies. In the following sections, we summarize the research results and what needs to be conducted to fulfil the goal of the area. The first area describes eight main questions to answer in the area of fault-failure analysis and its relation to the controlled experiments. The next area is the domain independence, where we have described six main questions. These two areas are the main focus for the time being. We have also explored efficiency, effectiveness and applicability as three major areas and formulated a series of 9 questions to answer. We explored these questions in Paper C, but are here exploring them further.

We realise that the scope of the proposed PhD thesis might not fit the proposed time-frame, and that we have to take some short cuts to achieve partial goals within the given time.

3.1 Fault-Failure analysis and construction of controlled experiments

This section describes our work to set up the controlled experiment in a way that improves our evaluation of test techniques, and possibly help us make guidelines that work outside a particular domain and a particular code. We hope that this line of research would help us better understand how software needs to be prepared to be used for our evaluation. This work will also result in a series of tools and a number of program versions with injected faults that could be used. We will not focus on faults that can easily be found by a compiler, i.e. syntactic faults. We are more interested in semantic faults, simple or complex. The questions below need to be answered, to be able to reach our research goal. Our main research result for this licentiate thesis is the understanding that these questions are the central questions to ask and our attempts to find right answers to them.

1. *How do we qualify and classify faults and failures?* In this area, we have already made extensive investigations [9] and we are realizing the difficulty of making

an “all purpose” classification. We have concluded that it is important to distinguish the fault that causes the failure, and be able to group them in different ways, which calls for a multi-layered classification. Our research in Paper D has shown that faults are distributed – and might be a combination of code (in different places) that together causes a failure. We are still working on providing a better classification that aids us in discussing the different types of faults in a more structured manner.

2. *In what way does the fault manifest itself in the code and how does the fault propagate into a failure?* This question of propagation is central, since it will aid in understanding the *observability* of a failure – and at what level the error or failure can be found. If we decide to work further with automatic fault injection (seeding or mutation), this could aid us in better understanding automation techniques. We have already invested in exploring the option by studying mutation test tools to be used to inject faults, and are currently trying to decide if we should create our own tool to aid us in our experiments.
3. *If we have problems of observability in a system, is there a way to improve the observability – in the purpose of making a better evaluation without changing the semantics of the program?* This question is very difficult depending on what approach is taken. If the system already exists, the solution would be to re-factor the code into something more testable, which is a research area in its own. Re-factoring of code without the goal of improving testability is hard as it is, and we are not sure exactly what it means to improve observability. For the latter part of the question, we hope to learn something about this area as a part of this research - in particular by conducting our test technique evaluation on different type of systems. The other way would be to improve testability as a part of the design at the systems architectural level during construction. We will not pursue this line of questioning at this time, even if it is intriguing.
4. *Is the fault system dependent, language dependent or a possible construct in any system using that language?* We must understand if the fault and its corresponding failure would be usable in the controlled experiment to evaluate test techniques in the sense that the result will be possible to generalize to other software or not. Dependencies of any kind will limit this ability. In this area we have already looked through our existing database of faults from our experiment in Paper D, and concluded dependencies to domain is low, and that the language dependencies exists, but the most faults could easily be transformed into a similar fault in another procedural language. We cannot yet determine if sticking to faults that are not system or language dependent makes us assume the program construct exists in all code. This area must be explored further – and might require a semantic model to be constructed, or a way to determine semantics of a fault.
5. *Which faults are already removed from the system before testing, and which are not found?* We plan to make a survey to find out which faults have already been removed from the system before testing. We reason that not all of these faults are removed consistently, and this set would be a valuable representative of faults to inject back to the system. This survey will also add knowledge on how to conduct a study where humans are involved. We hope to conduct this survey when we have at least a decent classification available, so that we at the same

time can test if the classification is understandable. What faults remain in the system is a much harder question to answer. We still have no idea how to tackle this, except conducting full path coverage – which in our experiment system is still unfeasible to do except for some few samples of code at component level. Therefore, the context and type of fault and its propagation in the software is central.

6. *What does a fault distribution, class and type, and fault occurrences tell us about the system, the testing that has been conducted, and what test techniques have been used?* This question is by no means easy to answer, and we hope that our work will give us insight in these problems. We hope by replicating our failure-fault analysis in another code or another domain, we will be able to understand how and in what way they result differ. This is only possible if we have a good classification and type understanding – a tool that we can inject faults with, and then some other software to experiment on. This will aid in assuring that our approach and result can be regarded as generic and also that our approach is working.
7. *Are there special events such as fault masking features of the system that disturb the analysis, and what impact they have on the result in distribution?* This question will not result in any special measures on our behalf other than making sure that we understand the implications and the threats of validity to our findings.
8. *What impact on the result has "constructed" faults injected compared to "real faults" re-injected?* This is a subject that many researchers have strong opinions about [10, 11], and an area that needs to be explored further. We feel that there is a range from very simple changes, "mutants", to very complex, multi-dependent faults in real systems that needs a "better" classification system to be able to determine if they have any impact. From a research point of view, this could be a rather philosophical question that in practice dismisses all research with injected faults of any kind, i.e. all constructed experiments. We are not willing to dismiss injected faults. However, anything that can better help us compare results is strengthening the construction validity of our experiment.

3.2 Domain independence

To be able to draw conclusions on if our result has domain independence, we need to understand what factors makes the results useful for other domains, i.e. generalized and scalable. We aim to approach this problem in a phased manner starting from one single system to across system within the same domain, and then across domains. This is depending on factors such as

1. *What fault distributions, frequency and type are typical in a particular system?* The best ways to explore this is by creating a classification and then try to repeat our work on Paper D in several different types of systems. We would also like to analyse if the faults found in one system are language or domain dependent or not.
2. *How the fault propagates and manifests itself in different types of domains/systems?* This would be easier to determine if an automatic fault

seeding (based on the results above) could be used.

3. *How does the programming language impact the domain, and what domains are using what languages? And – will this impact the types of faults that are present?* We are focussing on procedural languages, and in particular C. We hope to expand into C++ and Java, which we still have in our mind when constructing our approach. We believes that the procedural languages still dominates software in the industry, even if "new" powerful languages exist, that are more suitable to handle concurrency, multi-threading, functional and logical problems, such as Erlang, Prolog, Lisp and generated languages etc. We hope an overview of domain and languages used (and typical faults) could be created, but fear that this could be outside the scope of the PhD-work.
4. *What type or class of system has what particular characteristics? Which of these characteristics in a systems impacts testing for that type or class?* This is a wide question, which is related to the architecture of systems. We could apply a text-book approach, and then review the typical type of failures and fault to be found. We could also utilize our knowledge about "specific" fault/failure classifications and draw conclusion in what domain they are used in.
5. *What levels exists in different domains?* This questions points to the strategy of testing, and are sometimes easy to determine, and sometimes more organisation or process focused. The aim is to identify what is typical for a certain domain to make the guidelines comparable.
6. *What development process is used, and how does the development process impact what type of faults are constructed in the system?* It should be possible analyze software and system through its fault-failure and answering the above questions and draw conclusions on how that affects the result in testing. We need to understand which part of our results can be generalised and which cannot. Based on experience we can see the relation of observability is different in different types of systems as well as how easily some faults propagates into failures in some types of systems compared to others. At this point in time, we are not willing to draw any conclusions or make any proposals, since we have not yet started to explore this line of questions and in what way the domain will impact the result.

3.3 Efficiency, Effectiveness and Applicability

To compare test techniques and determine which ones are best, we must find means to measure and compare them. Measuring efficiency and effectiveness is common practice in research aiming to compare test techniques. Our contribution is that we have expanded these two measurements to encompass information that make them useful in an industrial setting. The importance of applicability is a new and novel concept when comparing techniques. If a test technique is to be applied in an industrial setting, this will be the most important area to explore as an addition to efficiency and effectiveness. Central questions to be answered include the following:

1. *In what way does the system context (programming language, design principles, domain classification) influence the efficiency, effectiveness and applicability of finding a fault/failure?* This question clarifies that any measurement performed,

is always measured in a context. It is important to understand, clarify in what way, and at what impact this context have. If we are able to clarify this, we have a better chance of making our measurements hold when we transfer them to another context. This investigation either could be performed in a large variety of system context, or as we suggest here, could be viewed as a result of the fault/failure analysis in conjunction with defining the system, which in addition implies observability of the system.

2. *What are good and bad measurements that evaluate efficiency and effectiveness?* We have suggested a set of measurements in Paper C that would be possible to use to measure the efficiency and effectiveness of test techniques. We have not yet had the possibility to test these measurements awaiting our experiments to be available. How the system context influences the result is also an open question at this point. We hope our fault-failure analysis, and our understanding of observability and different domains will cast some more light on these areas.
3. *What test techniques, groups of techniques are we focussing on?* It is clear that the test techniques have not yet been structured enough and definitely by the classification we propose in Paper C. Since we are separating the techniques abilities, we believe we will find a better understanding of which test techniques are useful at different levels, different systems, and how they could be used (manual or automated). This structure will aid us in a better taxonomy for discussing them, but also aid us in comparing them.
4. *Are there test techniques that are easier to understand than others are?* This is a fundamental question for the applicability of a technique. The ease of understanding a technique does not only concern grasping it at a conceptual level, but by asking this, we aim to find "traps" when defining them, using them and using them in different contexts. This concerns particularly techniques that have a human decision impact in its outcome.
5. *When is a technique applicable and when is it not? – Meaning, what type of system and what level (component, integration and/or system) is it useful to compare the test design technique with another?* This is an expansion of question 3 that will provide us with taxonomy for test techniques and question 4, which addresses one aspect of usefulness. Providing insight in this question will sharpen the guidelines to industry. We suspect this will open up for an entire new set of insights. One can further specify this into several sub-questions (not with the intent to be complete):
    a. *When are they comparable (within a group, across groups)?*
    b. *By number of failures found in a particular phase of the of the test design technique-concept?*
6. *What qualifies and defines itself as a test technique? What are variants of the same test technique?* Defining and limiting the definitions of a test technique in the taxonomy could be one approach. We do not aim to limit us too much, but would like to approach techniques as groups, and variants. Unfortunately, grouping on too high level will cause problems. Take the area of "random" test techniques as an example. There is a multitude of ways and approaches to perform random testing, and any element of "random" in a technique could also cause a big variation. We need to be careful when describing this, since the

naming could add to confusion if breaking traditional patterns, but also clarify aspects by bringing in new aspects.

7. *Is a technique applicable for manual or automation execution?* This relates to the deployment of the technique, which we view as one aspect of the applicability. When measuring efficiency, automation will outperform manual testing during execution. This is an unjust comparison, since one must take into account the time to automate, the number of times performing the test, as well as maintainability aspects of the test cases. Our novel approach is the expansion of the measurement to encompass these aspects. We must better understand what we mean by efficient and "fast" in different phases of execution of test techniques, which we clarify by the questions below.
    a. *How do we judge "fast" when it comes to automated testing?*
    b. *What do we mean by "fast" when it comes to learning?*
    c. *What do we mean by "fast" when we are using the technique on a particular software or system?*
8. *How much is dependent on manual judgement – and how much of a technique is rule-based and therefore in principle possible to automate?* These are fundamental questions when comparing applicability of techniques. Not only does it tie into the possibility to automate them, but also what value could be expected when using them.
9. *Is it possible to use and vary existing test cases by applying a test design technique and compare its outcome?* We have already attempted to answer this last question, but feel that this can only show results that are not generic, since there exist no way to currently define how to evaluate test cases in industry. This last question opens up an entire new area of exploration, well beyond the scope of our focus. Regardless of the research difficulty, this seems to be a way forward for any industry to expand and improve their testing.

These nine questions are by no means a complete set, but serve as an indication of our current insights in the problem. We conclude that our main contribution is that we have created a set of important questions that needs to be answered to fulfil our initial goal. We realize that the amount of work needed to answer them goes way beyond the scope of a PhD thesis. We have provided a structure, and an approach to define research by evaluating test techniques for the future.

Chapter 4 Contribution

Included Papers

Paper A: Eldh, S.: How to Save on Quality Assurance – Challenges in Software Testing, Jornadas sobre Testeo de Software, p 103--121, ITI, Universidad Politecnica de Valencia, Valencia, Editor(s): Tanja E.J. Vos (2006)

Paper B: Eldh, S., Punnekkat, S., Hansson, H.: Experiments with Component Test to Improve Software Quality**,** *IEEE International Symposium on Software Reliability Engineering (ISSRE)-Industrial track*, Trollhättan, Sweden (2007)

Paper C: Eldh, S., Hansson, H., Punnekkat, S., Pettersson, A., Sundmark, D.: A Framework for Comparing Efficiency, Effectiveness and Applicability of Software Testing Techniques. *Proc. TAIC*, IEEE, London, UK (2006)

Paper D: Eldh, S., Punnekkat, S., Hansson, H., Jönsson., P.: Component Testing is Not Enough - A Study of Software Faults in Telecom Middleware, 19th IFIP International Conference on Testing of Communicating Systems TESTCOM/FATES, Springer LNCS, Tallinn, Estonia (2007)

Sigrid Eldh is the main author for Paper A, B, C and D. Professor Hans Hansson and Professor Sasikumar Punnekkat contributed actively to Paper B, C and D in their role as supervisors. In addition, Paper C has contributions in the form of review and feedback on the written paper, and general discussions with the colleagues Licentiate Anders Pettersson and Licentiate Daniel Sundmark. In Paper D, Peter Jönsson as a part of his Master Thesis work created the data collection and class distribution, but he did not actively participate in the creation of the paper. Sigrid Eldh defined this Master Thesis, planned the work and performed supervision throughout the Master Thesis.

Paper A

The main contribution of Paper A is capturing the industrial challenges and explaining examples of successful (cost-saving) improvements conducted in the telecom industry as a case study. It provides state of the art in testing, and assesses Ericsson as a practical example, discussing the main questions to be answered with further studies. The paper also gives an overview of the area and available software – and presents test concepts that describe

- How do you test and make sure that the time you use in development and testing is effective and efficient?
- What quality you need to have in your product, at different levels?
- Consequences of bad quality
- Where we should spend our effort in the software life-cycle, and gives some examples describing how important testing is
- Software testing as an expanding area
- How does quality become a part of the software?

- Testing within Ericsson, Levels
- Software Quality Rank – And improvement in Component Test level
- Test Automation within Ericsson

Paper B is based on an attempt to improve component test, where we collected data from a large design organization with 23 design teams. The paper describes a method we call Software Quality Rank (SQR) and how it was deployed. The software quality was assessed before and after the deployment and how the different teams succeeded with the implementation of the SQR scheme. The work in Paper B emphasizes the importance of component test – developers own testing, and touches areas like review, static analysis and defines what is sufficient testing at different SQRs and how to measure at a component level.

Paper C is a research position paper – which outlines and presents the specific research area. The contribution is in particular a framework for how to compare test techniques, that in addition to the more common efficiency and effectiveness measurements, also takes into account how to evaluate the applicability of the test technique – to aid its practical usage in industry. The applicability addresses the ease of learning and applying the technique on a particular domain, but also takes into account the amount of human intervention needed in contrast to possibility of automating parts or the entire test technique. Another contribution is that the paper clearly defines the process and steps to be taken to make an evaluation with the aim to make it scalable and possible to generalize.

Paper D is focuses on the first step of the Paper C test framework with the aim to contribute to resolve the question on how to create a sound controlled experiment on a real live application. The contribution is that a fault and failure distribution is created. This is conducted with a special selection process making the failure analysis more efficient, and is conducted for a real industrial application for a period of three years involving 65 developers and targeting a 200 KLOC large component, encompassing 367 of 1096 Failures reported on the system. The analysis focuses on failures found by integration, system test and by the customers – but does not capture the component test level.

Chapter 5 Related Works

Research on Test Design Techniques has been performed for more than 25 years, and a good overview can be found in [16]. In parallel, a Master Thesis by Larsson, presents a collection of test techniques [13] 2004. Beizer [29], Basili et al [12] and Hetzel [27] have performed seminal work.

Test Design Techniques research is a lively area, where currently different search algorithms are explored, since it is possible to transform the test design problem to a search problem (e.g. genetic, hill-climbing etc [26]). Other areas that are the focus of research attention are; data generation testing approaches, DeMillo and Offutt [30], constraint testing, model based testing [23], regression testing [32], and slicing [31]. Basili & Elbaum eloquently explained how to do research based on test [19]. Hyonsook, Elbaum et. al has combined the Siemens-suite [21] of programs, packaged and added faults into a program called SIR [6] – available for research purposes and widely used by research. Rothermel and Elbaum's research [22], in conjunction with Weyuker and Ostrand [15], Vegas and Basili [28], and Harrold [20, 31] are among the pioneering works that consistently drives test design technique and its evaluations of them forward.

Another major area of research is fault injection, Voas [17]. There is a fine line between fault injection and fault seeding (which is the automatic injection) and its use in mutation testing, by DeMillo and Lipton [8]. The area of faults and failures has been explored by many since the first bug "a Moth" was found in a relay in one of the first computers 1947. Harrold and Offutt [20] are for instance looking at fault injection. The terminology of faults and failure and its references are explored in Paper D, and in Jönsson's Master Thesis [13].

Chapter 6 Conclusions and Future Work

The insights concluded so far in this research provide strong foundations and act as the major stepping stones for the future PhD phase of the research. Our goal is to create experiment files that are filled with different types of faults to start the next step, viz., 'Evaluation of test techniques'. This research has given valuable insight in to the problem we are trying to solve. We have grouped these insights into questions, and we have explained in Chapter 3, why they are important and how we aim to address them.

The summary of these above questions are:

A. What should a test technique evaluation experiment look like to get a general applicable result outside the actual software used?
B. What faults should be injected to give a fair representation of different types of faults, and how do we select them?
C. What are the different measurements that should be used to compare test techniques in an industrial setting?

For these main questions, we have explored the area, which have resulted in the following series of insights:

1. How important the fault and fault injection are for a fair evaluation and a solid result. Creating and constructing faults to be injected does not give reliable results at all times, since faults in real commercial software are much more complex in nature, than earlier believed. Constructed faults are still very valuable if we understand how they behave. Constructed faults could be used in addition to real faults, and are particularly useful for understanding how faults propagate into failures, and our insight is that faults (and their corresponding, or lack of, failure) can expose the observability of a system.
2. The difficulty of relating failures into faults in an industrial system. Not only does it require a good documentation, but sometimes the debugging also has to be repeated. We have through this process come to understand why many researchers avoid using industrial software for their experiments, since extracting faults to re-inject is currently a painstaking process
3. It is difficult to re-inject faults. We have started to understand what the nature of faults really look like. How semantically complex they are and the vast number of faults possible to make in any system. Our observation is that they are not so language dependent or domain dependent as often indicated.
4. It is difficult to separating failures from faults, and sometimes even separating them from their causes is unclear when reviewing different classifications.
5. The difficulty of predicting failure appearance when only knowing the fault has also become clear. The faults that cause a crash to the system when executed are easier to debug and identify, but our analysis says that most (approximate 90%) of the faults do not cause failures in a fault-tolerant system. Does this fact have anything to do with the observability of the system? It might. Systems with more visible consequences when executed are easier to observe. Crashes are very visible. We hope our further studies will help us understand this better.

6. Failure distributions are not easy to predict. We were rather surprised that our system had the failure distribution it had. That the largest influences are either missing code or spurious code, indicating problems of specification and context knowledge in large complex systems. This confirms an old result [15] that this might be typical for this type of systems considered, complex middleware and operating systems.
7. The cost of poor quality – and how important thorough component test is in a large and complex industrial system.
8. The deployment of a technique impacts its quality. It is not enough to achieve 100% statement coverage, since this could mean both good quality in some cases, and not nearly enough quality in others.

There are many more insights worth mentioning, our greatest being that this line of research is not taken by many, and it seems a very valuable path to take to learn more about how to understand our systems and how to evaluate test techniques with the purpose to provide guidelines for testing of real industrial systems.

Bibliography

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

[2] Dahl, O-J.. et al. (Dijkstra), "Structured programming", Academic Press, London and New York, 1972.

[3] Hierons, R. Brunel University, Keynote speech at TTCN-3 conference, Stockholm, Sweden, June 2007

[4] Eldh, S., Hansson, H., Punnekkat, S.; "Improving Functional Test Cases in a Automated Test Suite", *submitted to ICST 2008*, Lillehammer, Norway

[5] Whittaker, J.: Invited Keynote at European Int. Conference on Software Testing, Analysis and Review, Copenhagen 2000

[6] Hyunsook, D., Elbaum, S., Rothermel, G.: "Infrastructure support for controlled experimentation with software testing and regression testing techniques", *Proc. Int. Symp. On Empirical Software Engineering*, ISESE '04, pp. 60 – 70, ACM Aug. 2004

[7] Schütz, W: "The Testability of Distributed Real-Time Systems", Springer Verlag, 1993

[8] DeMillo, R., Keynote at IEEE Workshop on Mutation Testing and Analysis, at TAIC PART, Windsor, September 2007

[9] Jönsson, P.: "Analysis and Classification of Faults and Failures in a Large Complex Telecom Middleware System", IDE, MDH, Master Thesis, Nov 2007

[10] Reid, S.C., "Module Testing Techniques – which are the most effective? Results of a Retrospective Analysis", 5th European Int. Conference on Software Testing, Analysis and Review, Edinburgh, November 1997.

[11] Ishoda, S. "A criticism on the capture-and-recapture method for software reliability assurance", *Proc. Soft. Eng.* IEEE, 1995

[12] Basili, V.R. , R.W. Selby, "Comparing the Effectiveness of Software Testing Strategies" original 1985, revised Dec. 1987, in B. Boehm, H.D. Rombach, Martin V. Zelkowitz (Eds.). *Foundations of Empirical Software Engineering, The Legacy of Victor R. Basili*, Springer Verlag, 2005

[13] Larsson, M.: "Evaluation and overview of test techniques and their applicability in different real-time systems", IDE, MDH 2004

[14] IEEE Standard 829 Test Documentation, IEEE 1998

[15] Ostrand, T. S., and Weyuker, E. (1984). "Collecting and Categorizing Software Error Data in an Industrial Environment." *The Journal of Systems and Software*, 4:289-300.

[16] Juristo, N., Moreno, A.M. and Vegas, S. "Reviewing 25 Years of Testing Technique Experiments", *Journal of Empirical Softw. Eng.*, Springer Issue: Volume 9, Numbers 1-2, pp. 7 – 44, March 2004

[17] Voas, J. M., McGraw, G., *Software Fault Injection*, John Wiley & Sons, 1998.

[18] Avižienis, A., Laprie. J.: "Dependable computing: From concepts to design diversity". Proceedings of the IEEE, volume 74, pages 629–638, May 1986

[19] Basili, V., Elbaum S.: "Better Empirical Science for Software. Engineering", Invited Presentation, Shanghai, China, ICSE 2006

[20] Harrold, M. J., Offutt, A. J., Tewary, K.: An approach to fault modelling and fault seeding using the program dependence graph. Journal of Systems and Software, 36(3):273–296, March (1997)

[21] Hutchins, M., Foster, H., Goradia T, Ostrand, T.: "Experiments of the effectiveness of dataflow- and control flow-based test adequacy criteria". In Proc. of the 16th int. conf. on Software engineering, pages 191–200, IEEE Computer Society Press. Italy, 1994.

[22] Rothermel & Elbaum, "Test case prioritization: a family of empirical studies, Transaction of Software Engineering, Vol.: 28, Issue 2, page(s): 159-182, Feb 2002

[23] El-Far, I. K., Whittaker, J. A.: “Model based Software Testing”, Appears in 2001 This paper appears in the Encyclopedia on Software Engineering (edited by J.J. Marciniak), Wiley, 2001
[24] Zhu, H., Hall, P.A.V., May, J.H.R., “ Software Unit Test Coverage and Adequacy”, ACM Computing Surveys, Vol. 29, No.4 Dec 1997
[25] ISTQB Glossary, International Software Testing Qualifications Board, 2007
[26] Wegener, J. Sthamer, H., Jones, B.F. and Eyres, D.E., “Testing real-time systems using genetic algorithms”, *Journal of Soft. Quality*, Springer, Vol. 6 (2), June 1997
[27] Hetzel, W.C., “An experimental analysis of program verification methods“, *PhD dissertation*, Univ. North Carolina, Chapel Hill 1976.
[28] Vegas, S., Basili, V.R., “A characterization schema for Software Testing Techniques’, *Journal of Empirical Softw. Engineering*, 10, Springer, 2005, pp. 437-466
[29] Beizer, B. *Software Testing Techniques*, International Thomson Computer Press, 2nd ed., Boston, 1990
[30] R. A. DeMillio, A. J. Offutt, "Constraint-Based Automatic Test Data Generation," *IEEE Transactions on Software Engineering* ,vol. 17, no. 9, pp. 900-910, September, 1991.
[31] Larsen, L. and Harrold, M. J. 1996. Slicing object-oriented software. In *Proceedings of the 18th international Conference on Software Engineering* (Berlin, Germany, March 25 - 29, 1996). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 495-505, 1996
[32] Rothermel, G., Harrold, M. J. “Analyzing regression test selection techniques” IEEE Transaction on Software Engineering, Volume 22, Issue 8, pp. 529-551, August 1996