



Become a better tester!

Sigrid Eldh
Ericsson AB
Stockholm, Sweden
sigrid.eldh@ericsson.com

Abstract:

Learn what to focus on when it matters! In “super-speed” I will share with you a lot of tricks and hints to use at different situation throughout the different phases in the software life-cycle. Are effective and efficient are your testing? With this talk, you will learn ways to handle test and testing. My hope is to inspire you to new ways to succeed with how you approach testing by getting a wider perspective to handle the details. You will learn the secrets of improvements, you will find more bugs, understand automation and you will solve many testing problems, when different test techniques really works, how to make test faster, and why great software testing can really earn yourself and your company a lot of Money!

Keywords:

Life-cycle testing, effective and efficient, test improvements, test automation, test techniques, cost-efficient test

1 Introduction

A major problem for a tester is spending their time right and focus on the right test (making a priority). Creating test cases is the main task – in addition to planning and follow up (measuring the quality) of the software. To make great test cases, you must not only understand the system well, but the best way to approach of how to test this particular aspect of the system. Using and utilizing the software development life-cycle is always beneficial. Different processes also have different testing consequences and hurdles. Being aware of some simple fundamental process and management principles you will better spend your energy where it matters. Test Automation is such an area, that can both save a project quality, but also drain the resources of maintaining the automation suite instead of contributing to the quality. Having a fundamental idea of what test techniques that can be used, and what is effective at different test levels will help you more efficiently will aid you collect information about the quality in the best way.

2 Life-cycle aspects

Depending on the complexity of the system, the number of levels varies (more complex code, more levels or layers in the anatomy of the system). Since testing is also a way to measure the quality of the product – and to find defects, it must exist in every level, exploiting available tools and techniques. Focus is different of testing, depending on which phase of the system is executed, what type of system, who long time each phase takes, and of course skills of people at different levels. Maintenance is often consisting of testing the particular correction, and should also identify side-affects, which means extensive regression testing. Measurement and improvement of good regression suites –

PureConferences

www.pureconferences.com

Pure Conference Solution Pvt Ltd., A-108B, Sector 58, NOIDA 201301, India; Tel: +91 120 4621080

Page 1 of 5

that should be automated, is not as easy as it sounds, and if overlooked, the system will slowly “rot” i.e. deteriorate. In early phases of newly developed system, the focus is often vastly different than in mature systems. Planning is an aid to be effective, and follow up every day activities. If plans are made on weekly schedules, they are also late in weeks. Plans should be regarded as “contracts”, and honored. Honest planning and good measurements are crucial what time is spent on. Here dividing work in small packets with clear responsibility makes work easy to follow up. Again, not too much time should be spent on planning – so all test time should be taken into account. How much is difficulties in preparing systems? Understanding the software? What is the actual cost (time) of writing and reviewing test cases? Answering these questions leads to a better understanding of efficiency in HOW you work. We will describe some of the test techniques, and in what phase they are efficient.

3 Efficiency and Effectiveness of test

3.1 An Effective test finds defects– or has good coverage

One way to measure how effective a test case is – is by how many defects are found. This means that test cases that seldom find defects could be omitted to use in a regression suite but should be used once for contribution. On the other hand, a test case that does NOT find defects adds to coverage. Important test cases should be known, and different types of coverage should be explored, meaning - test cases should be added by actual contribution. Priority of effective tests is always difficult, but should be done. So we will describe some ideas around how to:

- Evaluate the test cases (which find faults, where)
- Explore coverage (structure of system, modules, but also on code level)
- Make priorities
- Understand which test techniques that are “good” for certain types of systems, defects, and coverage

So, how do you finds defects, more defects? Stumble across faults? That is the first step. Understand mistakes you can do – by reading about your existing defects. Also understand what are the problems of your software. I think some languages invite some type of faults, e.g. Memory faults are common... ok, how to expose memory faults? Some architectures invites some types of problems, if requirements are fuzzy, maybe describing a model – states and transitions, will give you new insight in how the system works – and also help you generate good tests. If input is the focus, maybe analyzing the test data, limits, frequency, types etc can help you choose the most effective test technique. Let make an experiment. Try and describe typical structures of the system you are testing – at different “levels”. Code? What are Code structural techniques? Right–Coverage. But testing code is often done through a load module (a built execution).. you will test it through its interfaces. So what are the parameters? What are the important functions? With this you will identify functional test cases. Combining these two will give you the most effective approach on low level/component testing. Ok, what about integration? What are the features that describe entities in an integration? Integration

between what? How are things dependent and related? This will give order, structure and a form of state chart, that probably will describe what test cases are good. Use cases? Particular sub-systems? Functions? User interface – and a database? Embedded code? Are you catching on? The first best trick is to ask the right questions. And find good answers. What is unique? What is similar? What can I compare against? You will be on a good track of test design creation by constantly asking yourself good questions. Not take things for granted and try and “trick” the code.

3.2 An Efficient test is FAST

In addition to finding defects and giving good coverage, speed is essential. Not only do we want to understand what to do – how to create good test cases, we also need them to be prepared and then execute them fast. Finally, this is a process that could partly be automated it will yield thousands of test cases, and the problem then becomes – verdict. Meaning evaluation of the test case – will then become the problem. So, which of the phases should one spend effort on? That is the key to great testing. Understanding all aspects of what takes time, hence what are the time consuming actions, will give more efficient testing. Then we explore how to be more efficient.

4 The Secret of Automation

Is automation home-made (scripting) or tools? ! Is Test Automation only Test execution? These are normal misconception. We have a great slogan within Ericsson “Automation everywhere”. We always ask ourselves if we can automate all tasks that are monotone, repetitive and time-costly. Is it worth the effort? To know this, good measurements and understanding of the automation cost – and maintenance cost must be conducted. If we understood that the first test case costs maybe many weeks, so that never holds in a comparison with testing manually. What must then understood is that the next thousands of test cases will only take a few minutes to write, and suddenly you have written thousands of test cases that can be run automatically. Understand the principle and make sure you should measure and learn yourself. Then there is the problem of maintainability. This is far more costly, if the automation architecture and test cases are done wrong, it usually is difficult to keep up with new versions, and changes. Again, understanding that design, libraries and strict templates needs to be used, are important to keep automation efficient to maintain. But maybe the problem of automation is basic. It is common that the testers do not know enough of how the tool works, no agreed usage of the tool is defined, and no champion exists. Then the tool might not be a contribution but a hurdle. It is easy to become an expert of a tool. Also, write scripts to “ease” any repetitive tasks. Document them. – A good start to get more efficient!

5 Improve your daily work as a tester.

The first task – before taking on improvements of an organization, is improving yourself. You can learn a lot by just learning from your own mistakes. You can measure what you spend your time on. You can quantify (give a value) to what your are good at, and what you need to improve. Challenge your weakness and build on your strengths. Love programming? Focus on making great test automation. Love trouble-shooting?



Understand the system well. Understanding your test object – and test techniques, can also be explored when creating tests. Start a measurement of “how many defects do I find” in one hour of test, one day of test and one week of test (get the numbers up). Then, improve the types of defects you find, how many are fixed, how many are improved? Ok, so designing test cases is one thing. There is also one other thing. By using the system you can explore it. How many defects do you find? How fast? Ok, then compare yourself to other testers in the team. What defects do they find? How? What do they do differently? Discuss this, and learn from each other! Now you are almost ready to use all this data to analyze and draw conclusions to improve your team. Analyzing and improving (learning from our own mistakes) is the most difficult part. But motivation – wanting to learn more – improving and evaluation is the first start. You just have yourself as the “test object”. Good luck! And do not forget to pat yourself – when you know you have done a good job. You probably earned it.

References

- [1] Eldh, S. “*On Evaluating Test Techniques in an Industrial Setting*”, Tech. Lic. Nr 87 MDH, Västerås, Sweden (2007)
- [2] 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)
- [3] 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)
- [4] 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)
- [5] 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)

Biography



**Sigrid Eldh,
Ericsson AB, Sweden**

Sigrid holds a Masters of Computer Science, and a Technical Licentiate degree, and is currently working to finish her PhD in Software Testing. In addition, she works full time as a Verification Expert at Ericsson. For more than 25 years she has brought new insight into the software testing field, by numerous of publications and appearances. She was a co-founder of SAST (Swedish Association of Software Testing), ASTA (Australian Software Testing Association), ISTQB and SSTB (Swedish Software Testing board). She is the chair of SSTB and has contributed to both the Foundation and the Advanced syllabus for ISTQB.