Testing is an Island – a Software Testing Dystopia

EuroSTAR 2008, v1.0.3

Abstract – Software testing is complex. In attempts to make it is easy to make decisions about testing, there are many trends today that move the profession more heavily towards precision, objectivity, specialists, metrics, presentation, standardization, short-term, independence, uniformity, verification, speed, ease. This will disregard the accompanying aspects serendipity, subjectivity, generalists, judgment, result, creativity, long-term, organic, diversity, validation, depth and complexity. In the future I am afraid we will have a superficial, fragmentized and soulless testing; without joy, participation, optimism, creativity, trust, tolerance, intrinsic motivation.

Rikard Edgren
TIBCO Spotfire Division
Första Långgatan 26
S-41328 Göteborg
Sweden

rikard.edgren@thetesteye.com

Introduction

This paper is written because I become upset every time I read things like *"people are the weakest link"*[1], *"all testing is verification testing"*[2], *"you can't manage what you can't measure"*[3].

Testing is a lot more than comparing a product against its requirements; and there is no way that standardization/certification/specialization can capture the creativity and communication that blossom when developing a product with great value to users.

Software testing is complex; and the usage of software is even more complex. There is not one easy solution; so I get sceptic when any silver bullet is presented. We need to consider numerous approaches and angles, preferably at the same time.

The industrial world of today is hierarchical and software is often focused on projects. The hierarchy generates managers that need measurements in order to manage; and the management itself can get more important than the content; at least for the managers, that more than usually decide.
It is difficult to make decisions that benefit long-term things like the product, and the people making the product.

That there are other ways; e.g. by treating your employees as adults, believing that they will do good things, and trusting their skills and capabilities, is often disregarded, but has been tried by a large Brazilian manufacturing company with very good results[4].

I have not used many references for the trends I'm seeing, rather I'm hoping you can see and feel some of the issues in your actual context.

A Tester in a Future Project

Let's jump a hundred years into the future.

I am a professional tester, and work as a consultant, as all other testers. This is because testing is separated from development; it is a service that can be bought depending on how much money you want to invest in testing.

Money is the key thing; and all contracts includes a bonus for accomplishing a test level, and sometimes also for bug reports[5]. The most important thing in my company is to have a good insurance, if I get sued by a customer for not finding a defect.

At the current project, I am testing the front-end for the login mechanism at a web store. It is the same as usual, I run the fancy tools I usually do, and generate automatic reports of the functionality. It's pretty easy actually; especially since I have nice leasing contracts with the respected tool vendors.

But there are some things I don't like about the way I work:

- I see a lot of interesting things outside my functionality area, but I am not allowed to report on that; and if I do, no one listens
- When testing the functionality manually, there are several things that I don't like, but I can't state any objective facts, so the issues won't be acknowledged
- I only test login functionality, so I no longer understand how the whole systems work
- Many times my defects won't get fixed, since my reports have metrics that are acceptable to the customers
- I get a lot of great feedback on the nice look of my extensive reports, but never any comments on the actual content
- I have used the same test approach for five years now, and I'm getting bored
- I am never involved in a project for a longer time than two weeks
- I don't get involved with the other people making the software, I actually never talk in person when I'm working
- I was at a conference for testers specializing in performance for login mechanisms on the web, and we all had the same suit and haircut
- I know that the functionality behaves according to the requirements, but I have no idea if the users like it
- I never have the time to dwell in interesting details, because I won't get paid for it
- The job is becoming too easy, it's no longer challenging

In the next chapter, we will see how this started.

Trends and Risks

Let's generalize: When we, as humans, think that we have a good thing going, we tend to focus too much on that seemingly good thing. This is good and natural, but there is a risk that it gives a tunnel vision, and loss of related aspects.

Not that precision, objectivity et.al. are necessarily bad; but when doing too much of them, they are really hurting us and our work in the sense that we don't do what is most important.

If these trends continue, we will have a future over-usage where testing has measurable activities in a very limited scope, but only sometimes contributing to a better product.

I also see a risk that we will lose important things that aren't measurable[6]: joy, participation, optimism, creativity, trust, tolerance, intrinsic motivation.

Below are listed a bunch of terms that aren't opposites, but rather things that software testing can have more or less focus on. There needs to be a balance.

Precision vs. Serendipity[7]

Most development departments have requirements that they should be predictable, and deliver at expected date with expected features. This is not bad in itself, but if we focus too much on that, it's a big risk that we will lose many other things that we didn't know we were looking for. We might be precise and specific, but we don't produce a lot of value in the long run.

Software testing differs from most other things in that there is no harm if something goes wrong. On the contrary, it is good to see what happens when things go wrong, since that will happen to end users, either intentionally, or by mistake.

As a manual tester (or a tester creating automated tests), it is common that while looking for something very specific, you stumble on something else that is very important; this is the everyday serendipity for software testing.

As a tester it is important to look carefully at many aspects of the software at the same time.

Objectivity vs. Subjectivity

Objectivity works best when there is completeness about the objects. E.g. science can be objective when there is information about all important parts. For software testing, we know that all tests can't be performed, and this is a main reason why subjectivity fits well in our discipline.

That software quality can be objective is an illusion. "Quality is value to some person"[8], and quality is something that resides in the relation between users and software. Zero security defects don't mean that the software is, or feels, secure.

It is with subjectivity we know what is most important. Sören Kierkegaard acknowledged this in the 19th century when saying "the truth is the subjectivity"[9]. Being subjective is being human.

Subjectivity makes sense when we acknowledge that it is people that use software, it is people that make software.

Specialist vs. Generalist

The scope for testers seem to get smaller and smaller. It is good to get into details; that's an important part of software testing, but it is bad for "the whole picture"; which is what we deliver to customers. The reason for this could simply be that it is easier to metrics-manage small chunks of employees with specialities.

A tester that looks at many parts of the software can see how features and attributes interact, and can put seemingly unrelated pieces together. By knowing the whole system it is also possible to grasp the value of features and defects, to know where more testing is needed.

The risk of specializing too much is that you become alienated from the product as a whole. Different persons perform performance, security, stability, calculation, environment testing. Different persons perform unit testing, integration testing, system testing, acceptance testing. We must be aware that the specialization and fragmentation probably reduces communication and the sense of ownership of the product.

Metrics vs. Judgment

Measurements have become increasingly important the last couple of hundred years. Without this we wouldn't have come as far as we have for many things

But the measurements are often very blunt, e.g. "Bug counts capture only a small part of the meaning of the attributes they are being used to measure." [10] And when adding value to an uninterpreted measurement, we get a metric that disregards the actual context[11].

Usage of metrics seem more appropriate for dead things like manufacturing objects, than for complex things like software, that involves people. This over-simplification gives a big risk that we focus on the measurable things, and ignore the more important stuff. At many times, sound judgement not only is enough; it is better.

Measurements combined with knowledge of details can give an analysis that is fruitful.
But metrics only, uses numbers to reduce the complexity and thereby the "truth" disappears.

Presentation vs. Result

It sometimes seems more important to present a result, than to reflect on what the actual result is. It can also seem that it is more important which tools that are used, than what the results are[12].

It could be argued that it is impossible to control what you do without a valid presentation, but many times you know enough already, and could spend time on actually improving your testing or the product.

A parallel to this is people that think that CMM or TMM says anything about how good the testing is. All those three-letter acronyms say is that you have documented the way you work, i.e. the process is defined, but the content is unknown.

In the dystopic future it is probable that a measurable Return On Investment needs to be seen for each activity; so the effort to present (not achieve) this will be the most important part.

Standardization vs. Creativity[13]

By using standards and best practices, we know what is done, and it is easier to estimate required time. Creative methods can be faster or slower, and we can't for sure know what the result will be. Creative ideas are by nature outside the process, and might therefore be banished, even if the process is flawed.

Standardization is a good thing especially when you are doing the same thing, e.g. producing paper towels; but software project are always different, and the testing effort (that never is complete) needs a lot of creativity.

We must also differentiate between standards regarding behaviour, e.g. how a software is supposed to work in Windows Vista, and standards regarding how to perform testing generally. The former has context, and is therefore more valid.

Using check lists to be creative is a good way of looking in both directions.

Short-term vs. Long-Term

Many professional activities are done as projects, which focus is to reach the deadline. The maintainability and benefits for the product long-term are secondary. And for projects with a tight schedule, secondary means that it doesn't exist at all. Matthew Heusser touches on this area when talking about technical debt[14]. We should focus more on the long-term products, and less on the short-term projects.

Thinking long-term doesn't mean we need to plan for everything; it means that we need to do simultaneous learning, communication, and execution; without cutting corners all the time, since corners also are important.

There is also a long-term aspect of the software testers. If you know more about the systems, you can test faster and more advanced. So by having, and taking care of testers, you will get better testing in the long run.

Independent vs. Organic

The independence of testing is based on the fact that it is very difficult to see your own mistakes, probably because you don't want to see them. But letting testers form a separate and isolated group can be a mistake based on fear of bias[15]. And when outsourcing testing (or development) only; you lose too much of the interaction and information sharing with developers.

With technical opportunities, and transportation energy problems; it is probable that a tester will work at home, being very specific, independent, fragmentized; but maybe not of too much value.

By treating development and testing in a more organic/holistic way; communication and information sharing will be a key element to understanding and producing software with subjective value.

Uniformity vs. Diversity

There are a lot of things said about what skills a good tester needs: basic programming, critical thinking, technical knowledge, fast learners etc. But it should also be acknowledged that it is good if different testers have different backgrounds, and thereby can achieve a greater coverage together, since they are looking at the product differently.

If all testers have the same background, they will look at the software in the same way; and thereby repeating similar tests. Since we know that all tests can't be performed, this isn't very good.

And different background and knowledge will also be positive for the ability to provide fruitful feedback on each other's work.

Humans are unique, and diversity is a good ground for creativity.

Verification vs. Validation

Verification can be simplified as "building the thing right"; and validation as "building the right thing". According to acronyms that like objectivity, i.e. CMM[16], this means that verification is checking at low-level, and validation is checking at high-level. In the future, I think most verification will be done programmatically, and validation by customer groups[17].

But the combinations and grey areas in between will be ignored; even though a manual tester can do both at the same time. There is no need for a new role called Product Investigator[18], because that role would create its own limitations.

Simultaneous verification and validation is very powerful, and is probably the reason why software testing is an important part of most software engineering; projects become better and faster. But simultaneous verification and validation can't be done by automation, and is very difficult to measure.

Speed vs. Depth

Everything becomes faster, easier to chew, and thereby superficial. An interesting counter-reaction is The Long Now Foundation[19] which I believe expresses the view that faster means cheaper, but slower means better.

Automated tests are faster, but narrower. The future will give a lot better automated testing tools, probably even GUI tools that give more value than the maintenance cost. But there is a risk that manual testing is disregarded; which might give technically sound products that no one wants to use.

Both speed and depth are needed, but sometimes one is more important than the other. As a software tester it is difficult, and important, to decide when to focus in more detail, and when to move on.

Time is needed, and it always runs out.

Easy vs. Complex

Acronyms might be popular because they are easy to remember. That the world is more complex is forgotten, for instance when efforts should be SMART:

keyword	Negative side-effect
Specific	we become narrow-minded
Measurable	can't use fluffy, good things
Attainable	can't aim really high; no holistic stuff
Relevant	no objections, but I'd prefer Important
Time/bound	gives short-term thinking

Isn't it strange that in order to get an acronym that is easy to remember, we omit Important, Non-De-Motivating, Wanted…

Endnotes

To summarize: testing (and software development as a whole) will be more and more like a fast-food chain[20]: superficial, fragmentized and soulless.

I want to be pessimistic so we all can get optimistic together. There is no need to go down this route; but it requires us to inform our managers and each other about this many, many times.

I would like to propose a subjective-holistic approach, where testing isn't an island.

With a subjective approach, we would see persons as thinking subjects that has valuable feelings. To be motivated might be the most important factor, in the long run. So instead of creating the most effective test cases, it is sometimes necessary to create the most interesting test cases. Quality is produced by all members of the development team; the testers are not by themselves responsible or un-responsible for end user satisfaction.
By being truly subjective, you can also skip the quasi-objective things, especially measurements/metrics that only are used to make it easier for management to manage. Humans have the unique ability to understand what is important.

The holistic approach applies to many things, for instance time. Don't focus too much on this sprint, or this quarter; know that people should work for a lot longer than this bonus-rewarding year.

By testing all parts of a product, it is possible for a tester to come up with many nice potential problems when sub-systems interact.

With a holistic approach testers should interact with all other parts of developments; give and get help.

There are dozens of quality aspects, both functional and non-functional, and a good tester should be able to look at most of them at the same time. A nice analogy is a conductor that can spot a small mistake by one instrument when many of them play at the same time[21]. It is also interesting that the conductor could do this without having seen the sheet music, but it would be more difficult if only one instrument played at a time.

The combination of subjective and holistic also means that testing is only a part of a tester's life.

"Everything is in a state of flux"[22].

[1] Alon Linetzki has the opposite of my view: "When dealing with projects, one has to think people. In the end, we are the weakest link." p. 18 in *Testing Experience* magazine 08/01

[2] A common view identified, but not held by Michael Bolton, *Two Futures of software testing* - www.developsense.com/presentations/twofuturesofsoftwaretesting.pdf (also presented at EuroSTAR 2008!)

[3] This is an incorrect interpretation of Tom DeMarcos "you can't control what you can't measure"; which gives a measurement hysteria, because people have an assumption that managers must control everything.

[4] Ricardo Semler, *Maverick!* 1993

[5] Actually there exist dubious activities where testers are paid by the number of bugs: www.utest.com

[6] Of course these things could be measured in some way in the future; either by questionnaires or brain sensors. But I believe that measuring subjective things will affect the results in a negative way.

[7] Serendipity isn't the most common word, so for an explanation see http://en.wikipedia.org/wiki/Serendipity

[8] Jerry Weinberg, *Quality Software Management: Systems Thinking*

[9] Sören Kierkegaard, *Afsluttende uvidenskabelig Efterskrift til de philosophiske Smuler* 1846

[10] Cem Kaner & Walter P. Bond, "Software engineering metrics: What do they measure and how do we know?" *10th International Software Metrics Symposium (Metrics 2004)*, Chicago, IL, September 14-16, 2004.

[11] The Context-Driven School of testing says "Metrics that are not valid are dangerous." http://www.context-driven-testing.com/ I wouldn't mind simply saying "Metrics are dangerous"

[12] Henrik Emilsson came up with the comparison with Monty Pythons movie *The Meaning Of Life*: "Ah, I see you have the machine that goes ping"

[13] For more information on creativity in software testing, see Rikard Edgren, *Where Testing Creativity Grows*, http://www.thetesteye.com/papers/where_testing_creativity_grows.doc

[14] "For the most part, North American business is optimized to create short-term results at the expense of the long term" Matthew Heusser, http://xndev.blogspot.com/2008/05/my-position-on-tech-debt-i.htm

also stating: "One easy way to move fast is to travel light."

[15] Cem Kaner, *The Ongoing Revolution of Software Testing*, http://www.kaner.com/pdfs/TheOngoingRevolution.pdf

[16] http://en.wikipedia.org/wiki/Verification_and_Validation_%28software%29

[17] Attempts to perform validation by automatic acceptance testing won't last long.

[18] On the other hand, if the dystopia comes true, I will probably try to get work as Product Investigator

[19] www.longnow.org – an example of their long-term thinking is that years are written with five digits (02008)

[20] The same idea was originally expressed by Michael Bolton, Do You Want Fries With That Test?

http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=9039

'Do you want to be a thoughtful testing cook who expertly uses the tools and ingredients available, or just some guy sitting behind a terminal flipping "testburgers"?'

[21] Analogy from Fredrik Scheja at SAST VÄST 2008 Q1, Gothenburg, see http://blogg.sogeti.se/itkvalitet/2008/02/testyrket-jmfrt.html

[22] "Panta rhei" – common characterization of Heraclitus philosophy