

Testing and requirements

Jukka Talvio

Director, SW processes and tools

(Long time ago trained by Hans Schaefer, the Norwegian God of testing.)



Software development beliefs

- You need documented requirements to test the system
- Quality is the conformance of the system against the documented requirements
- If the system does not meet the customer's expectations, it is often because the requirement document was bad.
- Testing is its own project, separate from development. There must be a separate organization for testing.
- Some even believe in "Document driven development".



My claims

- Requirement documents can *never* be complete
- Requirement documents do not *need* to be complete
- If the requirement documents were complete, they would be *useless*
- You must test things that were not in the requirement documents and therefore test against something else than the requirement document. What is it then that you test against?

Testing efficiently on system level requires understanding of the business model, product positioning and customer behavior. Reading the requirement document is not enough. Conformance to documented requirements is theoretical, not enough and is not the best way in practice.



Agenda

- Prove my claims
- What *should* be written down in good requirements but often isn't (and that can still be OK).
- Why is it normal that everything is not written down and does not need to be.
- What should be communicated to the testers (and other developers) but often isn't. (And it is *not* ok.)
- Instructions for making Quality Engineer asbestos pants that can withstand a lot of heat.



F-Secure has pretty good requirements...

- The requirements documents at F-Secure are quite good, even though there is a lot to improve...
- We have analysed thousands and thousands of bug reports the root cause of which was requirements.
- What should be improved in our requirement (and design/code too)?
- Answer: Exceptions
 - Unexpected situations
 - Integration with other 3rd party products
 - Strange network situations
 - Outcome of incorrect user behaviour.
 - Etc.



Complete requirements are unrealistic nonsense

- Since quality is said to be conformance against “Complete” requirements, the document would need to have...
 - For all possible input, allowed and not allowed inputs
 - What happens when you try to give the product not valid inputs
 - For some non valid input combinations, special behavior that apply to that combination only.
 - For all possible outputs, their valid range.
 - All 3rd party products that we should work with in the same computer.
 - For all features their minimum performance requirement.
 - Minimum usability requirements for all use cases.
 - What happens when the product is used, installed or updated incorrectly.
 - Planned performance, usability, reliability, supportability requirements.
- The requirements would be so “complete” that no one would read them or understand them.



And still, the best complete requirements miss something very important

For some reason, the “complete” requirements often miss the most important reason for making the product: *the business case*.

- How much effort is allowed in producing the product and how much revenue is expected.
- What is the major value adding feature of this product.
- How does the customer choose between this and other solutions.



So you say we don't need requirements?

- No, I am not saying that. Of course we need requirements.
- Not only do we need requirements from the users but also sales, support, marketing, evaluators, buyers, administrators,
- But instead of trying to make your requirements complete, you should do *risk based requirements analysis*.
- All requirements were not created equal. Not every requirement is worth writing down by itself.
- The greater the risk of getting the requirement wrong, the more we should spend time and money in communicating it. Including writing it down.
- For example, if the usability of the product is essential for its business success, you should invest more time and energy into analysing the usability requirements. And testing them.



Jukka's definition of excellent requirements

- All actions a user or implementer would consider separate are documented in a *use case* (functional coverage). *Abstraction level fairly high.*
- All new features that are important to *differentiate the product from competition.*
- *Usability* for key actors and *performance* related to the most common use cases.
- All other requirements that any stakeholder considers to have *direct effect on business, stakeholder satisfaction, etc.*
 - E.g. Sales may ask for a user interface that leaves a positive impression on 90% of prospective customers
 - Support should require that the user can easily identify the version and build she is using when on the phone with technical support
- All most common risks, hazard handling, and **error handling situations** as use case exceptions/alternative flows.
- Design (and testing) constraints
 - Compatibility with operating systems, devices, other software, etc. that appear in more than X% of the user base
 - The most important and common “things” with which the system is planned not to be compatible.



Other communication tools in addition to documents

- Face to face
- Meeting minutes
- Teleconferences
- Digital pictures of drawings made during a meeting
- White boards
- Internet Relay Chat
- News groups
- Twiki
- Web pages
- Email
- Phone
- Meetings



Define 3 most important test areas for the following application

Product name: Work time tracker

Description: A web based application that is used for tracking the absolute work time of employees

Features:

1. Login in to work
2. Log out from work
3. Keep track of the employee work time balance against their required work time and allocated vacation days and show that to the employee.
4. Show reports to company management about the percentage of sick days, spent vacation days, business trips, etc.
5. Person working at the phone switch (anyone else with access privileges) can use the system to see who is in the office and who is out on sick leave, business trip or vacation.
6. The product usability level must be such that the people accept using it.
7. Manage the system on one central system so that many companies can use it without seeing each others information.
8. Manage licenses for each company using the system and their users.



Define 3 most important tests for the following application

Product name: Work time tracker

Description: A web based application that is used for tracking the absolute work time of employees

Business case: Competing products in the market are difficult to use and usually based on special hardware. They are difficult and expensive to take into use. However, there are requirements in the law to keep track of how long people are at work. So every company needs some solution.

The business advantage here is that this application can be taken into use with very little investments. No special hardware is needed. Product is sold as a service to the customer. The web service is maintained by us in our own machine. Because of this, perception of the security of the service is important so that the customers believe it to be safe to use it over public networks.

One of the critical business success criteria is acceptance of the end users. Their main criteria is ease of use and that there are clear benefits to the end user, not just legal requirement.



Use business risks to direct testing.

Testing = Reducing and measuring quality risk

- We must test the functions that the users use most often *to reduce the risk of these functions not working.*
- We must test the functions that are essential for the business success *to reduce the business risk.*
- Conclusion, my suggestion:
 - Test installation and deployment of the system from the customer point of view.
 - Test security of the server side. There must be a description of the security environment and what kind of attacks the server is planned to withstand. Aim to demonstrate to prospective customers how the security is assured.
 - Test usability from employee point of view. Use a pilot.



What do you test against if not explicit requirements?

- ***Development* process is a learning experience.**
- Some of this knowledge is written down into the requirements, some in design, some in code comments, some in code, and some in the test cases.
- *Most* of the knowledge is never written down.
- You *must* talk to people to find out what needs to be tested, if a product passed the test, etc. And it is OK. Phone, meet, talk, Irc, (and even email if you must).
- If the knowledge is then important enough that it is beneficial to write it down, it should be written down. But not all information is or should be documented. Example: streamlined meeting minutes.



Quality Engineer's (=test engineer + more) real life

It is difficult to find a project where testers would have been given enough time to test and regression test the system.

I have never seen an experienced tester say after the project that there are absolutely no undiscovered bugs in the system.

Yet, when we find a bug after product release, it is common to hear from a manager:

Why did we not test this? It is so obvious we should have tested it. We need better testers to get better quality.



Instructions for protecting the behind of a Quality Engineer

- You must accept the financial and organization limits and at the same time keep pushing the organization to a new level.
- You must accept taking risks and it is your job to tell how big the risks are.
- The management needs to decide what risks to take and what not.
- If you haven't communicated the quality risks, the bugs after release *are* more your fault than the management's.
- On the other hand, if the management refuses to tell you the business logic and market situation so that you could do the risk analysis, the bugs and costs are more their fault than yours.
- *"It shall be the duty of managers to make decisions and the duty of engineers to make them informed ones."* – Towo Toivola, Quality Engineer
- Word of warning: it's ok for me to point fingers here but do not do it at work. It is not productive.



Master Test Plan (thanks to Petri Kuikka)

1. INTRODUCTION

PURPOSE, SCOPE, DEFINITIONS, REFERENCES

2. TESTING STRATEGY

How will select what to test and what not to test and how much to test anything.

How do you do test case selection.

If you automate testing, what will you automate, what not and why.

If you have multiple operating systems, how do you select what you test on each platform and what not.

What about regression testing?

3. REQUIRED TEST SPECIFICATIONS

What tests are documented and *which ones are not*

4. REQUIREMENTS FOR TEST LABS

What do I need to do the testing and what are the risks if I don't have it

5. SCHEDULE, RESOURCES AND RESPONSIBILITIES

Schedule and resources are usually dictated. So tell here what you can do with them and what are the risks. If not dictated, give at least two options, the best and the minimum.

5.4 TEST EXECUTION

Who will run tests and when.

5.5 METRICS

6. RISKS

Very important list for the safety of your behind. Remember to communicate them into the project risk management if there is such a thing.

6.1 PROJECT DELAYING PROBLEMS FROM PREVIOUS PROJECTS

7. APPENDIX A: FEEDBACK FROM SUPPORT ABOUT PREVIOUS RELEASES and comparison to their quality at the end of validation.



Summary

4 points you might remember in the afternoon

1. Requirement documents can *never* be complete and need not be complete. Every requirement was not created equal.
2. Requirements *document* is one tool to communicate requirements. There are others. The bigger team you have the more you should pay attention to the other means as well. People are poor readers.
3. *Testing efficiently requires understanding of the business model, product positioning and customer behavior. Reading the requirement document is not enough. Conformance to documented requirements is theoretical and does not work in practice.*
4. Document your testing plan, ask others to review it and have your management to approve it explicitly.



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

