More and Better Test Ideas

EuroSTAR 2009, v1.0.3

Abstract – Software is complex, so test ideas (or test conditions) should be broad and use many aspects of testers' knowledge, experience, creativity and subjectivity.

Using test specifications consisting of a lot of one-liner test ideas is a good way to give and get information.

The most important test ideas can either be used as basis for test cases, or as a guideline for other types of testing.

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

rikard.edgren@thetesteye.com

Test Ideas

Software testing is a multi-dimensional discipline and testing can be performed in many different ways. Common to most successful tests is that there is an idea behind the executed test, ranging from thoroughly documented in detailed test cases, to a sub-conscious thought in the mind of the exploratory tester.

Brian Marick has a good definition of test ideas: "a brief statement of something that should be tested."[1]

International Software Testing Quality Board use the term 'test condition', defined as "An item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element."[2]

The main difference between these two is that Marick's definition is vaguer, and doesn't state that verification is the only way[3]. But in essence, I think they mean the same, e.g. when ISTQB mentions "using test conditions to focus unscripted testing"

I like that test ideas can be expressed in many different ways, e.g. "verify that program is closed when selecting File, Exit", "play around with start/pause/resume/stop/extend functionality", "ask frequent users (including yourself) for usability improvements"[4] are equally interesting as test ideas for me. Test ideas can focus on what to test, or how, or both. They can be input to many things, from automated unit tests to exploratory scenario sessions.
They can be open, in the sense that you don't really know what will happen with them, you rather rely on the subjectivity, knowledge and experience of the testers.

My personal definition sums up as "test ideas are brief notions about something that could be tested."

James Bach uses "Test Idea: an idea for testing something."[5], which indicates that you don't really need a definition, you immediately understand what a "test idea" is.

Test ideas can be one-liner test cases, or a summary of a technique (investigate all state transitions), or a reminder of good testing things (perform regression testing for the most important bugs from last release), or whatever is suitable.

Test ideas can be created by brainstorming around every part of the system, including its interactions and quality attributes.

For me, test ideas are the most important part of software testing.

Generating Test Ideas

Software testers need to think in many different ways, because the end users will (intentionally or unintentionally) use the product in many different ways. The more test ideas that are thought of or executed, the greater chance of selecting the ones that will be finding information that is important for the product.

There are many methods for generating test ideas, and the best ways are probably as many as there are people working with testing multiplied with the number of software projects. Not having one true solution allow us to try different approaches, use different methods as it suits us, and change according to risks.

If you aren't developing the first version, your support organization is a huge inspiration. Your bug database is equally good, and the user's perspective might be found on the Internet. Learning the technologies in use will generate ideas; understanding how customers are working can yield the most important ideas. Looking at details will give different ideas compared to when looking at the whole picture, where your software only is a part of a big system.
By creating or understanding a model of the software, you will know where to apply your testing knowledge; by considering possible interactions between sub-parts you will identify things that might go wrong; by considering relevant quality criteria (CRUSSPIC STMPL + Accessibility) you will see parafunctional perspectives; talking to developers is a good source for information; if it is possible to play around with a version, test ideas probably come automatically.
And of course, you could also use the requirements[6] and specifications as well.

Creativity[7] and hard work is needed, and the problem might rather be when to stop; for each test idea, there are two more, and there are so many things that can go wrong.

Details about the test idea might not be interesting at this stage, so vaguer test ideas like "perform tests according to boundary value analysis for XX, YY, ZZ" can be useful. This will also make sure that you don't stop after just applying one technique, since one test idea will look like too little.

If you want more test idea triggers, you can use:

Blog posts like Michael Hunter's "Did I remember to"[8],
Cem Kaner's appendix of error types in Testing Computer Software[9]
James Bach's Heuristic Test Strategy Model[10] (SFDPOT[11], CRUSSPIC STMPL[12] and more)
Michael Bolton's HICCUPPS[13],
Michael Kelly's tours: FCC CUTS VIDS[14]
Elisabeth Hendrickson's Cheat Sheet[15]
Giri Vijayaraghavan's & Cem Kaner's Bug taxonomy for Shopping Cart[16]
Vipul Kocher's Quality Patterns approach[17]
Testing techniques from any book, web site, training or discussions
Bach's and Kaner's description of test types[18]
Other lists like Brian Marick's[19] and Jonathan Kohl's[20]

There are so many tips and tricks and questions[21] and heuristics[22] and methods that sometimes it works best to just think for yourself: what do I think is important?

Improving Test Ideas

Different persons look at software in different ways, so don't expect your test ideas to be complete or the best. Use tester colleagues, developers, support people etc. in order to get feedback and make the test ideas better, and remind them that it is difficult to see the things that are missing.

At the same time, your test ideas can be very good inspiration for developers. If they read your specification, they will surely come up with a good solution for a problem they hadn't thought about.[23]

It is also an opportunity for stakeholders to confirm that testing is going in the right direction, or not. And a way to let other people know a bit more about what software testing means at your company.

Communicating test ideas can take many forms, and my recommendation is to try written one-liners for test ideas. They are fast to read and review for testers, developers, managers; it is easy to re-use the essence of other test ideas; it is a way to see how complete the testing will be; it is a format that allows test ideas to be formulated in more or less vague fashion, where granularity (level of detail) can be adjusted for better readability.

I believe "good" test ideas capture importance. To identify really good test ideas, you can use the same criteria as Cem Kaner has identified in "What is a good test case?"[24]:

> Powerful; Yield significant results; Credible; Likely; Easier to evaluate; Useful for troubleshooting; Informative; Appropriately complex; Giving Insightful Information

Good test ideas should also be easy to understand, since many people with different background hopefully will read and understand. Good test ideas could be aligned with quality objectives, if such exist.

...and remember Kaner's advice that "Test cases can be "good" in a variety of ways. No test case will be good in all of them."[25]

Difficulties with getting feedback on test ideas can be that people don't really care; they don't take the time needed to understand the product or its testing or both.
It could be stressed that you probably will learn something by reviewing test ideas; that you don't have to understand everything, just a comment like "this is not especially important" is worthwhile.
It seems to me like test ideas is a good concept for some people, but not for all.[26]

Thinking in the opposite direction is often useful, don't be afraid to remove test ideas; time is limited, and the fastest and most fruitful test ideas are the ones to go with.

On the other hand; you don't want to spend too much time on writing and discussing, when you could be executing tests.

Execution of Test Ideas

So what do you do when you have these really good test ideas?
Well, you can use them as they are and test for all your worth (for small projects, I just enter the results in a new column next to each test idea.)
Or you can create other artefacts based on the test ideas; each test idea might become one or several detailed test cases, some test ideas might be automated at unit or system level, and some might be discarded because something changed, or they were more difficult/time-consuming to execute than you thought.
Each test idea could be a title in Session-Based Test Management[27]

A main drawback with exploratory testing is that it often is performed by a few testers after features have been implemented.
A main drawback with scripted testing is that it is text-savvy and not necessarily compliant with what was actually implemented.
One-liner test ideas could be in the middle of this.

As you learn more, you will realize that some test ideas aren't worthwhile, and you will come up with new ideas that are more important.
Often you can just execute these tests at once, and sometimes you might want to update your test idea list.

We also need to keep in mind that generating test ideas and executing them only is a part of software testing, and an even smaller part of software development.

When you see many test ideas at once, you should take the opportunity to use several of them at the same time. A good example is platform configuration things like Large DPI, operating system, language settings, which is appropriate to test on-the-fly.[28]
Some quality attributes, e.g. Stability, Usability, Accessibility is best evaluated after quite some testing has been performed.

The Future of Test Ideas

At least within a company, there are similarities within projects or desired quality attributes. If a test idea can't be copied directly, many of them can be translated to the new context[29] in a much easier way than if you have detailed test cases. Even better might be to have generic checklists of the company's important test idea triggers.

The lightness of test ideas open up possibilities of more easily interchanging interesting test information between testers, companies or industries. In the appendix there is a list of test ideas that I hope is fruitful for testers of completely different systems. This list is not a template, there is no need for that, since it is how you and your colleagues think that matters.

However, we also must be aware of the limitations: one-liner test ideas cannot help a lot regarding how to perform the tests, so they might not be helpful if test executors have little knowledge about the product and/or testing. They might also be too vague to mean something, e.g. if a test idea is a summary of a long scenario where it is the details that matter.

That test ideas don't have expected results is a good thing, since all tests have many results, and the tester should look for many of them, not just one thing.[30]

There are also several risks attached with just brainstorming for test ideas[31], you might miss a lot.
To make it easier, you could create a checklist with your favourite methods, or collect your own set of heuristics[32].

Categorizing, or tagging, of test ideas (e.g. in a database) could be very powerful, e.g. by identifying generic test ideas for re-use.
Ability to search for test ideas (e.g. based on rating) in an effective way is interesting.

I think that "test idea triggers" could be created by many others than software testers.
Each troublesome support issue should be possible to map to a vague test idea.
Stakeholders are allowed to point at important things that didn't make it to the requirements documents.

I don't think a wiki with generic test ideas for all testers in the world would work. What is important in each situation will be obscured; not only is the difference between Notepad and an elevator too big, there will also be different users, and different quality attributes that matter.
On the other hand, such a wiki[33] could be a great asset of test idea triggers.

I will continue to think about and use test ideas, they help me while testing.

Appendix

Session Tester v0.2 Test Ideas v0.3[34]

Session Tester - http://sessiontester.openqa.org/ - *is an exploratory testing tool for managing and recording Session-Based Testing.*
It is recommended to use many test ideas at the same time.

1. use the product while performing exploratory testing (not necessarily on Session Tester)
2. review the list of Test Primers (validity, duplicates, spelling, further suggestions etc.) - (priming.txt available in install directory)
3. review the Cheat Sheet by using it as inspiration for ad hoc testing

4. install the product on supported Windows versions, as Administrator and 'Restricted' user
5. install the product on supported MAC versions
6. install on a different drive than default
7. look at installed files, start menu items (more?)
8. verify ability to upgrade to a newer version
9. verify that uninstallation removes all files, except the saved sessions
10. do exploratory testing around install/upgrade/uninstall behavior

11. read the manual in one sequence
12. verify that the Help matches the functionality
13. verify that Help can be launched from Session Tester
14. look at appearance of the manual in applicable browsers (IE, Firefox, Safari, Opera, iPhone??)

15. use Session Tester on an error-prone machine (Large DPI, moved User catalogue, two monitors etc.)
16. check for memory leaks & leaking objects while testing
17. check all dialogs for conformance with standard behavior (access keys, tab order, alignment, Escape etc.)
18. verify that About dialog displays correct version number, correct license, and a web site address

19. verify handling of all types of characters for "Tester" and "Mission" entries (extended characters, japanese characters, empty, long strings)
20. verify a few samples for session length, reminder and warning
21. verify handling if system time is set back or forward

22. verify that warning appears on specified time before end
23. verify that warning appears, and re-appears on specified time before end, after extending the session
24. verify that reminder appears on specified interval
25. verify that warning appears on specified interval, after extending the session
26. verify that mission reminder can display extended characters, Unicode characters, long strings etc.
27. listen for sounds for warnings and reminders

28. verify that entered text is saved in XML and corresponding HTML file
29. verify that all different tags and their text is saved
30. verify handling of mis-spelled/new tags
31. check if tags are trimmed for leading and trailing whitespaces

32. use a large amount of different text inside a tag (imagine a log excerpt), including difficult strings like '@data'
33. do white-box review of the regular expression that grabs the session data
34. use same tag several times, and see that they are merged correctly
35. validate that available tags are really good
36. check that save is done at the appropriate moments (stop, end of session, closing session tester)
37. verify that all types of characters can be saved (extended characters, japanese characters, empty, long strings, newlines etc.)
38. verify escaping of xml code in session notes

39. play around with the start/stop/pause/resume/extend functions (or calculate all possible state transitions if you prefer that)

40. verify that Generate Web Report produces correctly formatted, fully viewable .html files
41. look at appearance of the Web Report in applicable browsers (IE, Firefox, Safari, Opera, iPhone??)
42. check the naming of web reports and web files
43. investigate the logic behind the creation of web report, then use what you learned
44. generate a Web Report before any session has been started
45. browse the Web Reports and see if it can give you an overview of the testing performed
46. check Performance of reporting by using thousands of session .xml files

47. see how easy to use the software is for someone un-familiar with the program using it all by themselves, without reading the manual
48. ask frequent users (including yourself) of Session Tester for usability improvements
49. look at all file deliverables for too little or too much information
50. scenario: several testers are working on the same project, saving session notes on a common network drive, gathering results and presenting to a manager
51. compare Session Tester to a (resemblance) of a competitor
52. check how the application deals with power saving schemes (hibernation, standby, shut down hard disks, shut down the monitor, etc) , locking computer (Windows+L on Windows) and waking up after this
53. use this test idea list as input for new ideas

[1] Brain Marick, in description of tool Multi at http://www.exampler.com/testing-com/tools.html: "A *test idea* is a brief statement of something that should be tested. For example, if you're testing a square root function, one idea for a test would be "test a number less than zero"."

[2] *ISTQB Standard Glossary of Terms used in Software Testing V.2.0, Dec, 2nd 2007* http://www.istqb.org/downloads/glossary-current.pdf

[3] It is also easy to think that True/False statements are needed. See ISTQB definition of "condition: A logical expression that can be evaluated as True or False, e.g. A>B.", *ISTQB Standard Glossary of Terms used in Software Testing V.2.0, Dec, 2nd 2007* http://www.istqb.org/downloads/glossary-current.pdf

[4] These examples indicate that this paper is grown from system testing, hopefully also useful for other types of testing

[5] James Bach, blog post *Some Useful Definitions*, http://www.satisfice.com/blog/archives/43

[6] The problem with using only requirements is that they aren't complete, and you also have implicit requirements, unspoken requirements, and incorrect requirements.

[7] The combination of software testing and creativity is investigated in Rikard Edgren, *Where Testing Creativity Grows*, http://www.thetesteye.com/papers/where_testing_creativity_grows.doc

[8] Michael Hunter, MSDN blog, http://blogs.msdn.com/micahel/articles/175571.aspx and http://www.ddj.com/blog/debugblog/archives/2007/05/you_are_not_don_31.html

[9] Cem Kaner, Jack Falk, and Hung Q. Nguyen (1999). *Testing Computer Software, 2nd Edition.* Wiley, and also at http://www.logigear.com/resources/articles_lg/Common_Software_Errors.pdf?fileid=2458

[10] James Bach's *Heuristics Test Strategy Model*, http://www.satisfice.com/tools/satisfice-tsm-4p.pdf ,

[11] James Bach web article *How Do You Spell Testing?* http://www.satisfice.com/articles/sfdpo.shtml I think Interactions would work well to add here.

[12] In James Bach's and Michael Bolton's course *Rapid Software Testing*, and also explained in Bolton's Better Software article *More Than One Answer; More Than One Question*, http://www.developsense.com/articles/2005-11-MoreThanOneAnswerMoreThanOneQuestion.pdf

[13] Michael Bolton, Sticky Minds article *Testing Without a Map* http://www.developsense.com/articles/Testing%20Without%20A%20Map.pdf

[14] Mike Kelly, blog post *Touring Heuristic*, http://www.michaeldkelly.com/blog/archives/50

[15] Elisabeth Hendrickson, *Test Heuristics Cheat Sheet* http://testobsessed.com/wordpress/wp-content/uploads/2007/02/testheuristicscheatsheetv1.pdf

[16] Giri Vijayaraghavan & Cem Kaner, *Bugs in your shopping cart: A Taxonomy* http://www.kaner.com/pdfs/BugsInYourShoppingCart.pdf

[17] Vipul Kocher *Questioning Patterns*, http://www.whatistesting.com/qpatterns/qpatternspaper.zip

[18] Cem Kaner & James Bach, *Black Box Software Testing* course http://www.testingeducation.org/BBST/index.html

[19] Brian Marick, blog post *Test design links (biased toward exploratory testing)*, http://www.exampler.com/blog/2007/11/06/test-design-links-biased-toward-exploratory-testing/

[20] Jonathan Kohl, *Getting Started With Exploratory Testing - Part 4*, http://www.kohl.ca/blog/archives/000188.html

[21] Rikard Edgren, blog post *Multi-Dimensional Software Testing*, http://thetesteye.com/blog/2009/03/multi-dimensional-software-testing/

[22] A lot on heuristics in *Rapid Software Testing* course, http://www.satisfice.com/info_rst.shtml with slides downloadable.

[23] This is one example of how testers can create low-hanging fruit for developers.

[24] Cem Kaner, *What is a good test case?* STAR East, May 2003, http://www.kaner.com/pdfs/GoodTest.pdf The list of attributes is my abbreviation of the content.

[25] Ibid.

[26] Some developers give you new ideas every now and then. Others are not interested at all.

[27] Elaborated by Michael Kelly in web article *Using session-based test management for exploratory testing* http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1352925_mem1,00.html

[28] See examples in Rikard Edgren's blog post *An Error Prone Windows Machine*, http://thetesteye.com/blog/2008/04/an-error-prone-windows-machine/

[29] This might be possible between different companies or areas, but there is a big risk you lose sight of what is important

[30] One of many important observations in Cem Kaner, *The Ongoing Revolution of Software Testing*, http://www.kaner.com/pdfs/TheOngoingRevolution.pdf

[31] Page 5 of Giri Vijayaraghavan & Cem Kaner, *Bug Taxonomies: Use Them to Generate Better Tests* http://www.kaner.com/pdfs/BugTaxonomies.pdf identifies many risks with just brainstorming for test ideas

[32] As described in Lesson 38 of Kaner, Bach, Pettichord *Lessons Learned in Software Testing*, John Wiley & Sons, Inc., New York 2002

[33] There exists an embryo at http://testforge.net/wiki/TestForge but not with a lot of content.

[34] The test idea list is available at Session Tester Forum: http://clearspace.openqa.org/thread/19517