37 Sources for Test Ideas

We recommend collecting test ideas continuously from a variety of information sources.
Consider the following, and think about values, risks, opportunities; find shortcuts to cover what is important.

Product	**1. Capabilities.** The first and obvious test ideas deal with what the product is supposed to do. A good start is requirements, examples and other specifications, or a function list created from actual software. But also try to identify implicit requirements, things users expect that haven't been documented. Be alert to unwanted capabilities. **2. Failure Modes.** Software can fail in many ways, so ask "what-if" questions to generate test ideas that expose the handling of internal/external, anticipated/unexpected, (un)intentional, realistic/provoked failures. Challenge the fault tolerance of the system; any object or component can break. **3. Models.** A state model helps identify test ideas around states, transitions and paths. A system anatomy map shows what can be tested, and can highlight interactions. Create a custom model using structures like SFDPOT from Heuristic Test Strategy Model. A visual model is easier to communicate, and the modeling activity usually brings understanding and new ideas. Many and rich models give better test ideas. **4. Data.** By identifying intentional and unintentional data (there is always noise), you have a good start for a bunch of test ideas. Follow easy and tricky data through the application, be inside and outside boundaries, challenge data types and formats, use CRUD (Create, Read, Update, Delete), exploit dependencies, and look at the data in many places. **5. Surroundings.** No product is an island, so environment compatibility (hardware, OS, applications, configurations, languages) is one of many important testing problems, but also investigate nearby activities to your product. By understanding the big system you can get credible test ideas that are far-fetched when looking at functionality in isolation. **6. White Box.** By putting a tester's destructive mindset on developers' perspective of architecture, design and code, you can challenge assumptions, and also find mistakes that are cheap to fix. Pay special attention to decisions and paths you might not understand from a black-box perspective. Code coverage is not worthless, it can be used to find things not yet tested. **7. Product History.** Old problems are likely to appear in new shapes. Search your bug/support system or create an error catalogue, remember critical failures and root cause analyses. Use old versions of your software as inspiration and oracle. **8. Rumors.** There are usually lots of talk about quality and problems. Some hurt the product and the organization. Use the rumors themselves as test ideas. It is your mission to kill them or prove them right. **9. Actual Software.** By interacting with the software, you will get a lot of ideas about what is error-prone, connected, interesting. If you can eat your own dog food (euphemism: sip your own champagne), you are in much better position to understand what is important about the software. If "Quality is value to some person", it is pretty good if that person is "me". **10. Technologies.** By knowing the inner workings of the technology your software operates in, you can see problematic areas and things that tend to go wrong; understand possibilities and security aspects; which parameters to change, and when. You can do the right variations, and have technical discussions with developers. **11. Competitors.** By looking at different solutions to similar problems you can get straightforward test ideas, but also a feeling of which characteristics end users are interested in. There might be in-house solutions (e.g. Excel sheets) to be inspired by, and often there exists analogue solutions for similar purposes. Can you gain any insightful test ideas from your competitors support, FAQ or other material?
Business	**12. Purpose.** The overall purposes of the product will give you goals for your test ideas. Ask a couple of extra why? to find out the real purposes. These can give you the broadest benevolent start that can find very important problems, fast. **13. Business Objectives.** What are the top objectives for the company (and department break-downs) ? Are there any requirements that contradict those objectives? Do you know the big picture, the product vision and value drivers? **14. Product Image.** The intended behavior and characteristics of the product might be explicit or implicit, inside the walls and minds of the people producing or consuming the software. You will be able to write compelling problem reports if you know and can show threats to the product's image, e.g. by pointing to a violation of marketing material. **15. Business Knowledge.** If you know the purpose of the software, and the context it operates in, you can understand if it will provide vale to customers. If you can't acquire this knowledge, co-operate with someone who knows the needs, logic and environment. **16. Legal aspects.** Do you need to consider contracts, penalties or other legal obligations? What would cost the company the most in a legal issue? Do you have a lawyer that can give you hints on what must be avoided?
Team	**17. Creative Ideas.** All products are unique, and require some test ideas not seen before. Try lateral thinking techniques (e.g. Edward De Bono's Six Thinking Hats, provocative operators, the opposite, random stimulation, Google Goggles) to come up with creative test ideas. Metaphors and analogies is a good way to get you started in new directions. **18. Internal Collections.** Use or create lists of things that often are important in your context, some call these quality/test patterns, others have product-specific quicktests. **19. You.** You are a user. You might be a stakeholder. You matter. Take advantage of your strengths from experiences, skills, knowledge and familiarity with problems. Use your subjectivity and feelings to understand what's important. And don't forget to acknowledge your weaknesses and blind spots.

Project	**20. Project Background.** The reasons for the project are driving many decisions, and history from previous (similar) projects are well worth knowing about in order to make effective testing. **21. Information Objectives.** It is vital to understand the explicit and implicit purposes of the testing effort. If you can't get them, create your own quality objectives that steer test ideas for any feature. **22. Project Risks.** Some of the difficult things in a project can be addressed by testing. You want to know about which functionality developers are having trouble with, and you will adjust your schedule depending on risks that need mitigation first. **23. Test Artifacts.** Not only your own test ideas, logs and results can be used for sub-sequent tests, also try out test results from other projects, Beta testing reports, usability evaluations, 3rd party test results etc. What questions do you want to be able to answer in status reports? **24. Debt.** The shortcuts we take often give a constantly growing debt. This could be project debt, managerial debt, technical debt, software debt, testing debt or whatever you wish to call it. If the team keep track on what is on the debt list, you can map a set of test ideas against those items. **25. Conversations.** The informal information you get from people may contain things that are more important than what's included in specifications. Many people can help you with your test design, some are better judges of importance, what can you gain from MIP:s (Mention In Passing)? If developers know you can find interesting stuff, they will give you insider information about dubious parts of the software. A set of questions to a developer might be an innocent *"what do you think we should test?*" or "*what part of your code would you have liked to do better?*" **26. Context Analysis.** What else in the current situation should affect the things you test, and how? Do you know about the market forces and project drivers? Is there anything that has changed that should lead to new ways of testing? What is tested by others? Which strengths and weaknesses does the project and its people have? **27. Many Deliverables.** There are many things to test: the executable, the installation kit, programming interfaces, extensions, code & comments, file properties, Help, other documentation, Release Notes, readme:s, marketing, training material, demos etc. All of these also contain information you can use as inspiration. **28. Tools.** If something can be done very fast, it is a good idea to try it. Tools are not only the means to an end, they can also be used as the starting point for exploration.
Stakeholders	**29. Quality Characteristics.** Quality characteristics are always important for the project to be successful, although the OK zone can be easy to reach, or difficult and critical. Our definition includes capability, reliability, usability, charisma, security, performance, IT-bility, compatibility, supportability, testability, maintainability, portability, and a plethora of sub-categories. Many of these can be used as ongoing test ideas in the back of your head, executed for free, but ready to identify violations. **30. Product Fears.** Things that stakeholders are really worried about are much stronger than risks, they don't need prioritization, they need testing. Typical hard-to-verify, but useful-for-testing fears are: loss of image, wrong decisions, damage, people won't like the software. Different people have different fears; find out which matters most. **31. Usage Scenarios.** Users want to accomplish or experience something with software, so create tests that in a variety of ways simulate sequences of product behavior, rather than features in isolation. The more credible usage patterns you know of, the more realistic tests you can perform. Also try eccentric soap opera tests to broaden test coverage. **32. Field Information.** Besides knowledge about customer failures, their environments, needs and feelings, you can take the time to understand your customers both in error and success mode. Interview end users, pre-sales, sales, marketing, consultants, support people, or even better: work there for a while. **33. Users.** Think about different types of users (people you know, personas), different needs, different feelings, and different situations. Find out what they like and dislike, what they do next to your software. Setup a scene in the test lab where you assign the testers to role play different users, what test ideas are triggered from that? Best is of course unfiltered information directly from users, in their context. Remember that two similar users might think very differently about the same area.
External	**34. Public collections.** Take advantage of generic or specific lists of bugs, coding errors, or testing ideas. As you are building your own checklist suitable for your situation, try these: • Appendix A of Testing Computer Software (Kaner, Falk, and Nguyen) • Boris Beizer Taxonomy (Otto Vinter) • Shopping Cart Taxonomy (Giri Vijayaraghavan) • Testing Heuristics Cheat Sheet (Elisabeth Hendrickson) • You Are Not Done Yet (Michael Hunter) Learn some testing tricks or techniques from books, blogs, conferences; search for test design heuristics, or invent the best ones for you. **35. Standards.** Dig up relevant business standards, laws and regulations. Read and understand user interface standards, security compliance, policies. There are articles out there that describe how you can break something even if it adheres to the standards, can you include their test ideas? **36. References.** Reference material of various kinds is a good source for oracles and testing inspiration, e.g. an atlas for a geographic product. General knowledge of all types can be handy, and Wikipedia can be enough to get a quick understanding of a statistical method. **37. Searching.** Using Google and other means is a good way to find things you were looking for, and things you didn't know you needed (serendipity).

Rikard Edgren, Martin Jansson and Henrik Emilsson - thetesteye.com v1.0

inspired by Sabourin's 10 Sources for Testing Ideas, Bach/Bolton's HICCUPPS(F) and Kaner's writings.