Test Framing

To test is to tell two parallel stories: a *story of the product*, and the *story of our testing*. Test framing is a key skill that helps us to compose, edit, narrate, and justify the story of our testing in a logical, coherent, and rapid way. The goal of test framing is to link each testing activity with the testing mission.

Elements of Test Framing

The basic idea is this: in any given testing situation

- You have a testing mission, a search for information. Your mission may change over time.
- You have information about requirements. Some of that information is explicit, some implicit; and it will likely change over time.
- You have risks that inform the mission. Awareness and priority of those risks will change over time.
- You have ideas about what would provide value in the product, and what would threaten value. You'll refine those ideas as you test.
- You have a context in which you're working. That context will change over time.
- You will apply oracles—principles or mechanisms by which you would recognize a problem. You will refine some oracles and discover others as you go.
- You have models of the product that you intend to cover. You will refine and extend those models throughout the project.
- You have test techniques that you may apply. You also have choices about which techniques you use, and how you apply them. You will develop techniques along the way.
- You have lab procedures that you follow. You may wish to follow them more or less strictly from time to time.
- You configure, operate, and observe the product (using test techniques, as mentioned above), and you evaluate the product (by comparing it to the oracles mentioned above, in relation to the value of the product and threats to that value).
- You have skills and heuristics that you may apply. Those skills and heuristics will develop on this project, and in your overall career as a tester.
- You have issues related to the cost versus the value of your activities that you must assess.
- You have tests to perform. These are usually fewer than the tests you would like to perform. In any case, you choose your tests from infinite number of possible tests.
- You have time in which to perform your tests. Your available time is probably severely limited compared to the time needed for tests you'd like to perform. The time available to perform tests asymptotically approaches zero, relative to the time required to perform all possible tests.

Test framing is the capacity to follow and express, at any time, a direct line of logic that connects the mission to the tests. Such a line of logical reasoning will typically touch on elements between the top and the bottom of the list above.

Purpose of Test Framing

The *purpose* of test framing is to be able to provide clear, logical, credible answers to questions like

- Why are you running (did you run, will you run) this test (and not some other test)?
- Why are you running that test now (did you run that test then, will you run that test later)?
- Why are you testing (did you test, will you test) for this requirement, rather than that requirement?
- How are you testing (did you test, will you test) for this requirement?
- How does the configuration you used in your tests relate to the real-world configuration of the product?
- How does your test result relate to your test design?
- Was the mission related to risk? How does this test relate to that risk?
- How does this test relate to other tests you might have chosen?
- Are you qualified (were you qualified, can you become qualified) to test this?
- Why do you think that is (was, would be) a problem?

The Form of Test Framing

The form of test framing is a line of propositions and logical connectives that relate the test to the mission, touching on the elements of testing as listed above.

A *proposition* is a simple statement that expresses a concept. The statement may be true or false. In test framing, we typically use propositions as affirmative declarations or assumptions. Occasionally, we will use propositions as the basis of hypotheses to be tested or falsified.

Connectives are word or phrases that link or relate propositions to each other, generating new propositions by inference. Examples include “and”, “not”, “if”, “therefore”, “and so”, “unless”, “because”, “since”, “on the other hand”, “but maybe”, and so forth.

The kind of language used in test framing need not be a *strictly* formal system, but one that is heuristic and reasonably well structured. Here are a couple of fairly straightforward examples.

Example 1:

Mission: Find problems that might threaten the value of the product, such as program misbehaviour or data loss.

Proposition: There’s an input field here.
Proposition: Upon the user pressing Enter, the input field sends data to a buffer.
Proposition: Unconstrained input may overflow a buffer.
Proposition: Buffers that overflow clobber data or program code.
Proposition: Clobbered data can result in data loss.

Proposition: Clobbered program code can result in observable misbehaviour.

Connecting the propositions: IF this input field is unconstrained, AND IF it consequently overflows a buffer, THEREFORE there's a risk of data loss OR program misbehaviour.

Proposition: The larger the data set that is sent to this input field, the greater the chance of clobbering program code or data.

Connection: THEREFORE, the larger the data set, the better chance of triggering an observable problem.

Connection: IF I put an extremely long string into this field, I'll be more likely to observe the problem.

Conclusion: THEREFORE, as a test, I will try to paste an extremely long string in this input field AND look for signs of mischief such as garbage in records that I observed as intact before, or memory leaks, or crashes, or other odd behaviour.

Example 2

Mission: Evaluate our product and its interaction with a related product.

Proposition: Our product, MobileMoolah, allows users to record financial transactions at the point of sale, and to upload batches of those transactions to a compatible products.
Proposition: PC ChequeBook has the second-largest market share of desktop finance products, where CompuCash is the market leader.
Proposition: PC ChequeBook allows localized date formats (mm/dd/yyyy, dd/mm/yyyy, yyyy-mm-dd, and so forth).
Proposition: MobileMoolah also allows localized date formats.

Connection: If we set the date formats to the same setting in both products, we can reasonably expect to import data from MobileMoolah to PC ChequeBook without having to modify the date manually.

Proposition: A goal of any software product is to save time and increase convenience for a customer.
Proposition: One oracle that would point to a problem in the product is inconsistency with its explicit or implicit purposes.
Proposition: Another oracle that would point to a problem in the product is inconsistency with its reasonable user expectations.

Proposition: Upon importing data from MobileMoolah into PC Chequebook, with both date formats set to dd/mm/yyyy, we observe that the imported transactions contain the format mm/dd/yyyy.

Connection: If, after the transfer of data between PC ChequeBook and MobileMoolah, the month and day fields appear switched for any or all transactions, the product is inconsistent with a reasonable user expectation, and inconsistent with the product's implicit purpose. THEREFORE we have reason to suspect a bug.

Proposition: Based on this test alone, we cannot determine without further investigation whether the bug is in PC ChequeBook or MobileMoolah.

Proposition: The new update to MobileMoolah is slated to ship in four days.

Connection: IF our mission is to find as many bugs as we can before ship time, we should quickly reproduce this bug, and report it immediately (along with the steps to reproduce it) without further investigation.

Connection: IF, at this time, this is the most serious problem that we've found, AND IF we believe our mission includes discovering whether this is a general problem rather than a problem specific to PC ChequeBook, we should reproduce this problem with a different desktop program (probably the market leader) before moving on.

Proposition: We have a CompuCash test system available to us, on which we can run this same test immediately and have a result within two minutes.

Connection: IF the problem were to reproduce with CompuCash, THEREFORE we would be able to infer that we're seeing a general problem with MobileMoolah.

Conclusion: SINCE we can get some more useful information quickly and specific information quickly, we might as well.

Now, to some, the thought process in these examples might sound quite straightforward and logical. However, in our experience, some testers have surprising difficulty with tracing the path from mission down to the test, or from the test back up to mission—or with expressing the line of reasoning immediately and cogently.

Our approach, so far, is to give testers something to test and a mission. We might ask them, given the mission, to describe a test that they might choose to run; and to have them describe their reasoning. As an alternative, we might ask them why they chose to run a particular test, and to explain that choice in terms of tracing a logical path back to the mission.

Unframed Tests

If you have an unframed test, try framing it. You should be able to do that for most of your tests, but if you can't frame a given test right away, it might be okay. Why? Because as we test, we not only apply information; we also *reveal* it. Therefore, we think it's usually a good idea to alternate between focusing and defocusing approaches. After you've been testing very systematically using well-framed tests, mix in some tests that you can't immediately or completely justify.

One of the possible justifications for an unframed test is that **we're always dealing with hidden frames**. Revealing hidden or unknown frames is a motivation behind randomized high-volume automated tests, or stress tests, or galumphing, or any other test that might (but not certainly) reveal a startling result. The fact that you're startled provides a reason, in retrospect, to have performed the test. So, you might justify unframed tests in terms of plausible outcomes or surprises, rather than known theories of error. You might encounter a "predicatable" problem, or one more surprising to you. In that case, better that *you* should say "Who knew?!" than a customer.

Structures for Test Framing

Part of the skill of test framing involves identifying structures that inform testing, and the elements of those structures. The following quick references might be helpful in constructing the testing story.

Testing Mission: Testing is not necessarily a simple matter of finding bugs. There are many possible missions for testing, including informing ship/no-ship decisions, competitive evaluation, or finding safe scenarios and workarounds for problems. "Different objectives require different testing tools and strategies, and will yield different tests, different test documentation, and different results." See page 19 of Cem Kaner, "Challenges in the Evolution of Software Testing Practices in Mission-Critical Environments." Software Test & Evaluation Summit/Workshop (National Defense Industrial Association), Reston VA, September 2009. http://www.kaner.com/pdfs/NDIAkanerSept2009.pdf

Information about requirements: On any project, there's always more information available than one might think at first glance. The trick is to be able to find and exploit those sources of information quickly and consciously. Consider *reference*, *inference*, and *conference* as heuristic sources of knowledge. They're all useful, they're all incomplete, and each may contradict, reinforce, or refine the other. See Michael Bolton, "Rock, Paper, Scissors", *Better Software, Vol. 8, No. 11, December 2006*. http://www.developsense.com/articles/2006-11-RockPaperScissors.pdf

Requirements as reference, conference, and inference is also discussed in Kaner, Bach, and Pettichord, *Lessons Learned in Software Testing: A Context-Driven Approach.* Wiley, 2001.

Risks: There's a four-part story to risk, in which a victim—some person—suffers harm or loss or annoyance, due to a weakness in the program that is triggered by some threat. For a brief introduction to risk ideas, consider James Bach, "Heuristic Risk-Based Testing" in *Software Testing and Quality Engineering, 11/99.* Also look at http://www.satisfice.com/articles/hrbt.pdf Michael Bolton, "Test Design with Risk in Mind". *Better Software, Vol. 9, No. 7, July 2007.* For a more detailed list, see Kaner, Falk, and Nguyen, *Testing Computer Software, Second Edition*, Wiley 1999. For a discussion of risk in general, see Nassim Nicholas Taleb, *The Black Swan: The Impact of the Highly Improbable (Second Edition)*, Random House Trade Paperbacks, 2010.

Value: Value in a product is always subjective and multi-dimensional. For a quick reference to some of ways people might evaluate a product, see the "Quality Criteria" section of the Heuristic Test Strategy Model at http://www.satisfice.com/tools/satisfice-tsm-4p.pdf. See also Donald Gause and Gerald M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.

Context: See "The Satisfice Test Context Model" (http://www.satisfice.com/tools/satisfice-cm.pdf) and the Project Environment section of the "Heuristic Test Strategy Model" (http://www.satisfice.com/tools/satisfice-tsm-4p.pdf).

Oracles: See the (previously unpublished) article "Oracles" at http://www.developsense.com/resources/Oracles.pdf. See also Michael Bolton, "Testing Without A Map", *Better Software Vol. 7, No. 1, January 2005*.

http://www.developsense.com/articles/2005-01-TestingWithoutAMap.pdf. Finally, see work by Cem Kaner and Doug Hoffman, “Introduction: The strategy problem and the oracle problem”, *Black Box Software Testing*, Center for Testing Education and Research, Florida Institute of Technology. http://www.testingeducation.org/BBST/BBSTIntro1.html.

Coverage: The first step when we are seeking to evaluate or enhance the quality of our test coverage is to determine for whom we're determining coverage, and why. A mapping illustrates a relationship between two things. In testing, a map might look like a road map, but it might also look like a list, a chart, a table, or a pile of stories. We can use any of these to help us think about test coverage. Yet excellent testing isn't just about covering the "map"—it's also about exploring the territory, which is the process by which we discover things that the map doesn't cover. See Michael Bolton, “Got You Covered”, *Better Software, Vol. 10, No. 8, October 2008*, http://www.developsense.com/articles/2008-10-GotYouCovered.pdf; “Cover or Discover”, *Better Software, Vol. 10, No. 9, November 2008*, http://www.developsense.com/articles/2008-11-CoverOrDiscover.pdf; and “A Map By Any Other Name”, *Better Software, Vol. 10, No. 10*, December 2008, http://www.developsense.com/articles/2008-11-AMapByAnyOtherName.pdf

Test Techniques: See the “Test Techniques” portion of the Heuristic Test Strategy Model. Also see Lee Copeland, A Practitioner’s Guide to Software Test Design, Artech House Publishers, 2003.

Skills: See “Exploratory Testing Skills and Dynamics” at http://www.developsense.com/resources/et-dynamics22.pdf