

# Test Framing

Version 1.1

Michael Bolton and James Bach

December, 2010

*This paper appears largely as it was presented at EuroSTAR 2010, as a supplement to a tutorial by Bolton. It is based on a blog post(<http://www.developsense.com/blog/2010/09/test-framing/>) by Bolton in September 2010, which in turn is based on work by Bach through 2010.*

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 have methods for test execution, in which you configure, operate, observe, and evaluate the product. The particular methods will likely vary from one test to the next.
- 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 deliverables that you must develop, maintain, present, and archive. These may include reports on bugs or issues, status information, test coverage outlines, risk lists, test data, strategy or design documents, automation code, configuration notes, and more.

- You have time in which to perform your design, test procedures, and reporting. 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, well 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, will be, 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

Test Framing

Michael Bolton

Page 4

12/8/2010

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 "predicable" problem, or one more surprising to you. In that case, better that *you* should say "Who knew?!" than a customer.

## Oracles

The role of oracles is so important that it deserves special note here. An oracle is a heuristic principle or mechanism by which someone recognizes a problem.

If we perceive a problem, it's because an oracle is telling us that there's a problem. Conversely, if we don't see a problem, it's because no oracle is telling us that there is a problem. That doesn't mean that there is no problem, or that there's no oracle for a problem that's there. It simply means that, for whatever reason, we're not applying a principle or mechanism that would identify a problem. We may not be aware of the oracle, or the oracles that we have may be misleading us, or failing to lead us far enough.

Oracles are by their nature heuristic. That is, oracles are fallible and context-dependent. Oracles do not tell us conclusively that there is a problem; rather, they suggest that there *may be* a problem. As Gause and Weinberg define it<sup>1</sup>, a problem is "a difference between things as perceived and things as desired". Perception and desire are both human and subjective, relative to some situation at some time. Consequently, there can be no absolute oracle. Most testers are familiar with applying an oracle which seems to indicate a problem, whereupon a program manager replies, "That's not a bug; that's a feature." The tester and the program manager here are applying different oracles. Neither can be said to be wrong intrinsically; instead, each is using a different principle or mechanism.

*Consistency* is an important theme in oracles. Unless there is a compelling reason to desire otherwise, we generally tend to want a product to be consistent with

- **History:** That is, we expect the present version of the system to be consistent with past versions of it. Naturally, if a product is inconsistent with its history because a bug has been fixed, we likely appreciate the inconsistency. Yet if our programmers have provided a workaround for the old problem, and the fix requires us to change our work habits, we may resent the fix! This underscores the point that all oracles are heuristic. Oracles may give inconsistent indications, and they contradict each other. Oracles should be applied thoughtfully, rather than followed.

---

<sup>1</sup> Donald Gause and Gerald M. Weinberg, *Are Your Lights On?* Dorset House Publishing Company, Inc. (March 1, 1990).

- **Image:** We expect the system to be consistent with an image that the organization wants to project, with its brand, or with its reputation. This can work both ways; for example, a game producer might specialize in strategy games such that the strategy aspect is paramount and graphic design is relatively unimportant. For such a company, problems with graphics receive less attention than problems with the strategic aspect of the game.
- **Comparable Products:** We expect the system to be consistent with systems that are in some way comparable. That might include other products in the same product line, or from the same company. The consistency-with-past-versions (History) heuristic is arguably a special case of this more general heuristic. Competitive products, services, or systems may be comparable in dimensions that could help to discover a problem. Products that are not in the same category but which process the same data (as a word processor might use the contents of a database for a mail merge) are comparable for the purposes of this heuristic. A paper form is comparable with a computerized input form designed to replace it. Indeed, any product with any feature may provide some kind of basis for comparison, whereby someone might recognize a problem or a suggestion for improvement.
- **Claims:** We expect the system to be consistent with what important people say about it. These claims may take the form of reference (documents or products that you can point to), inference (what you believe someone important might say about the system), or conference (what someone important *does* say). The claim may be incomplete or in error, in which case testing may reveal a problem with the claim, rather than a problem with the system. Important people might disagree in their claims about what the product should do. The tester's role is not to decide the matter, but to make people aware of the disagreement.
- **Users' Expectations:** We expect the system to be consistent with some idea about what its users might want. Consider "users" broadly here. A system that will be used in many different ways will have diverse users whose expectations and desires may conflict. Often the direct user of a product is acting as a proxy for the person who receives the bulk of the benefit of the product or service, as a travel agent is operating a reservation system largely on behalf of her client.
- **Product:** We expect each element of the system to be consistent with comparable elements in the same system. A product might afford several means of accessing or observing a particular variable; consider the different ways of setting the margins—via a visible ruler or via a dialog box—in a word processing program, or differences between screen and print output. User interface elements should be broadly consistent with one another, both for consistency of user interaction and consistency of image.
- **Purpose:** We expect the system to be consistent with the explicit and implicit ways in which people might use it. If some aspect of the product is missing, such that it fails to fulfill the user's needs or support the user's task, we suspect a problem. If the product over-delivers, presenting options or features that confuse, overwhelm, or slow down a user, we suspect a problem.
- **Standards and Statutes:** We expect a system to be consistent with relevant standards or applicable laws. Note that compliance with a standard may be voluntary; a development group may choose to violate a point in a standard or may reject the standard entirely. Yet non-adherence to a standard should be conscious, rather than compulsive. It may be the tester's role to draw attention to non-conformance with relevant standards—or unnecessary conformance with irrelevant standards.

There is one more heuristic that testers commonly apply. Unlike the preceding ones, this one is an *inconsistency* heuristic:

- **Familiarity:** We expect the system to be *inconsistent* with any patterns of familiar problems. Note that any pattern of familiar problems must eventually reduce to one of the eight consistency heuristics.

We can carry this list of consistency heuristics in our heads more easily by applying a mnemonic, based on the first letter of each heuristic guideword: HICCUPPS (F).

These consistency heuristics are subject to Joel Spolsky's Law of Leaky Abstractions ("All non-trivial abstractions are to some extent leaky.") This means that there may be overlap between the heuristics. That's fine; broad, overlapping categories prevent an important problem, or class of problems, from being overlooked by defining our categories too narrowly.

Since oracles are fallible and context-dependent, testers cannot know—and therefore should not decide—the deep truth about any observation or test result. No single oracle can tell you whether a program (or a feature) is working correctly at all times and in all circumstances, so it's important to use a variety of oracles, and to be open to applying new ones at any moment. Any program that looks like it is working may in fact be failing in some way that happens to fool all of your oracles. To defend against that, you must proceed with humility and critical thinking<sup>2</sup>.

Because oracles are fallible, a tester reports whatever seems plausibly to be a problem. How does one decide on plausibility? Testers apply *abductive inference*, cycles of reasoning to the best explanation. We collect data and observations, we hypothesize explanations to account for the data, and we evaluate the hypotheses. Then we make a decision: choose the hypothesis that so far best accounts for the data, and stop; or collect more data—or more hypotheses. This too is a heuristic process, and "heuristic devices don't tell you when to stop"<sup>3</sup>.

Oracles can be used prospectively or generatively; in the moment that they're applied; or retrospectively. We usually have a large number of oracles at our disposal before we start testing. Yet we often do not have oracles that establish a definite correct or incorrect result in advance.

- You may use an oracle to help design a test. ("If the data doesn't get transmitted to the server after I press update, that would be inconsistent with an implicit purpose and an explicit claim, so I'll look for a problem like that.") Cycle through the list of oracle heuristics while you're engaged in test design.
- You may suddenly become conscious of an oracle ("Hey... that account balance is *negative*! I didn't expect *that* to happen! That's inconsistent with what I *would have* expected, had I anticipated that in advance.")
- You may apply an oracle retrospectively. ("Since that particular standard came to my attention, I realize now that what I saw when I was testing two weeks ago was non-standard behaviour. I'm going to investigate that now that I have a reason to suspect it was a bug.")

At any time subsequent to the test, you may cite an oracle heuristic to explain why you believe something to be a problem. A problem (or non-problem) may be more easily recognized with the application of multiple oracles that agree with each other. Oracles may contradict one

---

<sup>2</sup> For an excellent introduction, see David Levy, *Tools of Critical Thinking: Metathoughts for Psychology (Second Edition)*. Waveland Press, 2009.

<sup>3</sup> Gerald M. Weinberg, *An Introduction to General Systems Thinking, Silver Anniversary Edition*. Dorset House, 2001.

another. A product owner's decision on what to do about a problem report may be influenced by choices about which oracles to apply. Therefore, since our role as testers is to provide credible information, we may also choose to use different oracles to temper our test framing or our bug advocacy<sup>4</sup>.

## **References on 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. DeMarco and Lister cover software risk in *Waltzing with Bears*, Dorset House, 2003. For an interesting and provocative 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, (<http://www.satisfice.com/tools/satisfice-tsm-4p.pdf>). See also Donald

---

<sup>4</sup> Cem Kaner and James Bach, “# Bug advocacy: How to win friends, influence programmers, and stomp bugs”, *Black Box Software Testing*. Center for Testing Education and Research, Florida Institute of Technology. <http://www.testingeducation.org/BBST/BBSTbugAdvocacy.htm>

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 Michael Bolton, “Testing Without A Map”, *Better Software Vol. 7, No. 1, January 2005*. <http://www.developsense.com/articles/2005-01-TestingWithoutAMap.pdf>. For more on oracles, see Cem Kaner, “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.satisfice.com/blog/wp-content/uploads/2009/10/et-dynamics22.pdf> (with an introduction at <http://www.satisfice.com/blog/archives/370>).