Oracles

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[1], 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 is wrong; 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.
- **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

---

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

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 choice 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; the object is to 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 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's 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[2].

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 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"[3].

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 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[4].

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

---

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

[3] Gerald M. Weinberg, *An Introduction to General Systems Thinking, Silver Anniversary Edition*. Dorset House, 2001.

[4] 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