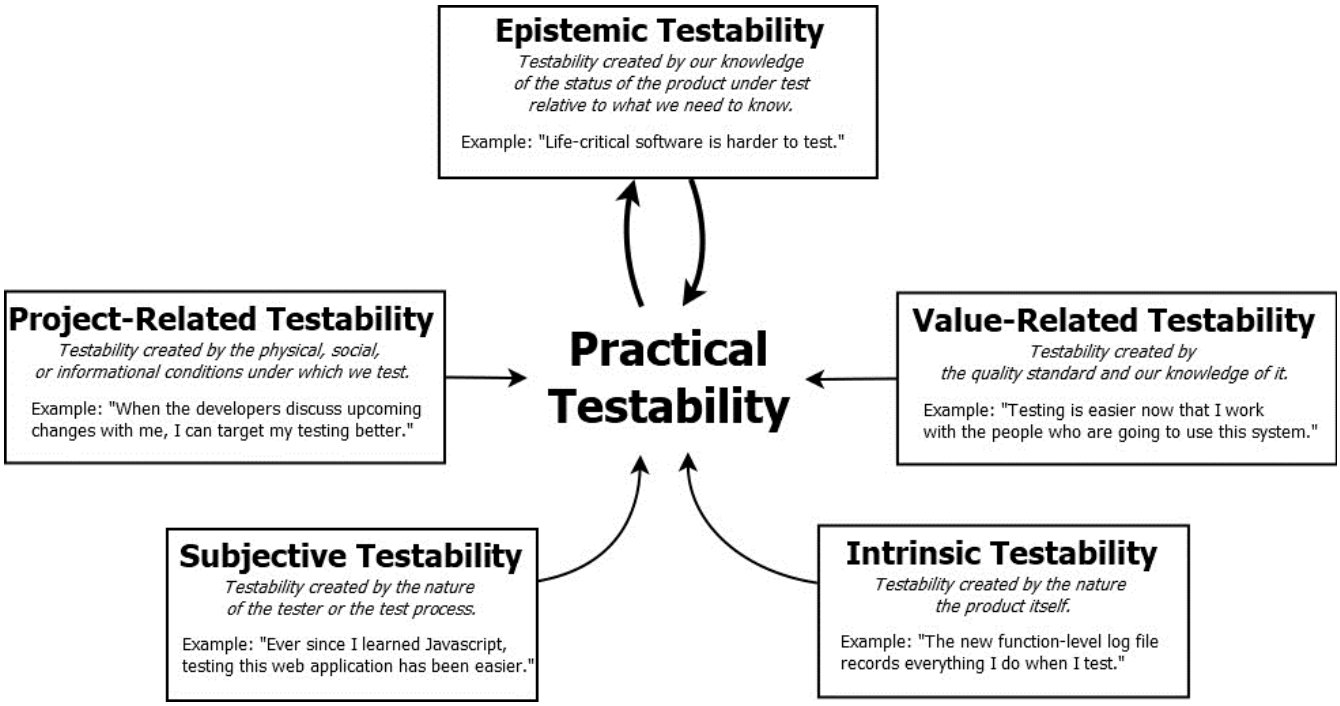
Heuristics of Software Testability

Version 2.7, 2024, by James Bach, Satisfice, Inc.

The *practical testability* of a product is how easy it is to test* by a particular tester and test process, in a given context†. Practical testability is a function of five other "testabilities:" *project-related* testability, *value-related* testability, *subjective* testability, *intrinsic* testability, and *epistemic* testability (also known as the "risk gap"). Just as in the case for quality in general, testability is a plastic and multi-dimensional concept that cannot be usefully expressed in any single metric. But we can identify testability problems and heuristics for improving testability in general.



Interesting Testability Dynamics

Changing the product or raising the quality standard reduces epistemic testability. The difference between what we know and what we need to know is why we test in the first place. A product is easier to test if we already know a lot about its quality or if the quality standard is low, because there isn't much left for testing to do. That's epistemic testability. Therefore, product that changes a lot or in which we can't tolerate trouble is automatically less testable.

Improving any other aspect of testability increases the rate of improvement of epistemic testability. Efforts made to improve any other aspect of testability, by definition, increases the rate at which the gap between what we know and what we need to know closes.

Improving test strategy might *decrease* subjective testability or vice versa. This may happen when we realize that our existing way of testing, although easy to perform, is not working. A better test strategy, however, may require much more effort and skill. (Ignorance was bliss.) Beware that the opposite may also occur. We might make a change (adding a tool for instance) that makes testing seem easier, when in fact the testing is worse. (Bliss may be ignorant.)

---

* *Testing* is evaluating a product by learning about it through experiencing, exploring, and experimenting.

† *Context* is all of the factors a situation that should be considered when solving a situated problem.

Increasing intrinsic testability might *decrease* project-related testability. This may happen if redesigning a product to make it more testable also introduces many new bugs. Or it may happen because the developers spend longer stabilizing the product before letting independent testers see it. Agile development done well helps minimize this problem.

Increasing value-related testability might *decrease* project testability. You can dramatically increase value-related testability by embedding yourself with real users in the field, and even participating in their work. You will be testing under the most realistic possible conditions. However, this can be difficult, expensive, slow, and lead to accidental retention or mishandling of confidential personal data as you test, which hurts project-related testability.

Increasing practical testability *also* improves development and maintenance. If a product is easier to test then it is also easier to support, debug, and evolve. Observability and controllability, for instance, is a tide that floats all boats.

The tester must *ask* for testability. We cannot expect any non-tester to seriously consider testability. It's nice when they do, but don't count on it. An excellent tester learns to spot testability issues and resolve them with the team.

Guidewords for Analyzing Testability

Epistemic Testability

- **Prior Knowledge of Quality.** If we already know a lot about a product, we don't need to do as much testing.
- **Tolerance for Failure.** The less quality required, or the more risk that can be taken, the less testing is needed.

Project-Related Testability

- **Prudence.** If the business is desperate to release the product, they will be willing to accept a lot more risk and therefore be much less interested in testing, and will make themselves less available to testers.
- **Supportive Culture.** Testers are highly sensitive to intimidation or disrespect. No one wants to feel like a troublemaker or outsider. Protect the independence and purpose of testers if you want their best work.
- **Developer Availabilty**. The ability to speak with and perhaps influence developers/designers/managers makes a nurturing environment for testing.
- **Change Control.** Frequent and disruptive change requires retesting and invalidates our existing product knowledge. Change control generally improves testability, but can also hurt it if we aren't allowed to update our test environments, tools, and data. Change control may occur on a local or corporate level.
- **Information Availability.** We get all information we want or need to test well.
- **Tool Availability.** We are provided all tools we want or need to test well.
- **Test Item Availability.** We can access and interact with all relevant versions of the product.
- **Sandboxing.** We are free to do any testing worth doing (perhaps including mutation or destructive testing), without fear of disrupting users, other testers, or the development process.
- **Environmental Controllability.** We can control all potentially relevant experimental variables in the environment surrounding our tests.
- **Time.** Having too little time destroys testability. We require time to think, prepare, and cope with surprises.
- **Leanness.** Complexity and magnitude of work products, accumulated over time, must be navigated or maintained to get the testing done. Also, complicated bureaucratic processes reduce time available for tsting. Avoid technical debt and administrative overhead.

Value-Related Testability

- **Oracle Availability.** We need ways to detect each kind of problem. A well-written specification is one example of such an oracle, but there are lots of other kinds of oracles (including people and tools).
- **Oracle Authority.** We benefit from oracles that identify problems that will be considered important.
- **Oracle Reliability.** We benefit from oracles that can be trusted to work over time and in many conditions.
- **Oracle Precision.** We benefit from oracles that facilitate identification of specific problems.
- **Oracle Inexpensiveness.** We benefit from oracles that don't require much cost or effort to acquire or operate.

- **User Stability & Unity.** The less users change and the more harmony among them, the easier the testing.
- **User Familiarity.** The more we understand and identify with users, the easier it is to test for them.
- **User Availability.** The more we can talk to and observe users, the easier it is to test for them.
- **User Data Availability.** The more access we have to natural data, the easier it is to test.
- **User Environment Availability.** Access to natural usage environments improves testing.
- **User Environment Stability & Unity.** The less user environments and platforms change and the fewer of them there are, the easier it is to test.

Subjective Testability

- **Engagement.** A bored or distressed tester is a poor tester. A good tester is intellectually and emotionally engaged in the work; confident of finding any problems that may exist.
- **Involvement.** Testing is easier when a tester is closer to the development process, communicating and collaborating well with the rest of the team. When testers are held away from development, test efficiency suffers terribly.
- **Product Knowledge.** Knowing a lot about the product, including how it works internally, profoundly improves our ability to test it. If we don't know about the product, testing with an exploratory approach helps us to learn rapidly.
- **Technical Knowledge.** Ability to program, knowledge of underlying technology and applicable tools, and an understanding of the dynamics of software development generally, though not in every sense, makes testing easier for us.
- **Domain Knowledge.** The more we know about the users and their problems, the better we can test.
- **Testing Skill.** Our ability to test in general obviously makes testing easier. Relevant aspects of testing skill include experiment design, modeling, product element factoring, critical thinking, and test framing.
- **Test Strategy.** A well-designed test strategy may profoundly reduce the cost and effort of testing.
- **Supporting Tester Availability.** A "supporting tester" is anyone who is not a regular tester and who is not responsible for the testing, yet is available to help test the product. Developers can be good supporting testers.

Intrinsic Testability

- **Observability.** We must see the product. Ideally, we want a completely transparent product, where every fact about its states and behavior, including the history of those facts is readily available to us.
- **Controllability.** We must be able to visit the behavior of the product. Ideally, we can provide any possible input and invoke any possible state, combination of states, or sequence of states on demand, easily and immediately. Any non-deterministic behavior of the product detracts from testability.
- **Algorithmic Simplicity.** We must be able to visit and assess the relationships between inputs and outputs. The more complex and sensitive the behavior of the product, the more we will need to look at.
- **Algorithmic Transparency.** If no one knows how the product produces its output (as is typical with machine learning systems), we will need to sample much more of the input and output space to test it well.
- **Algorithmic Stability.** If changes to the product can be made without radically disturbing its logic, then past test results will not have to be thrown out with every tiny modification. Although any system might suffer with instability, it is especially a problem in machine learning.
- **Explainability.** We must understand the design of the product as much as we can. A product that behaves in a manner that is explainable to outsiders is going to be easier to test. "Explainability" is also a hot topic in AI.
- **Unbugginess.** Bugs slow down testing because we must stop and report them, or work around them, or in the case of blocking bugs, wait until they get fixed. It's easiest to test when there are no bugs.
- **Smallness.** The less there is of a product, the less we have to look at and the less chance of bugs due to interactions among product components. This also applies to the amount of output we must review.
- **Decomposability.** When different parts of a product can be separated from each other, we have an easier time focusing our testing, investigating bugs, and retesting after changes.
- **Similarity** *(to known and trusted technology).* The more a product is like other products we already know the easier it is to test it. If the product shares substantial code with a trusted product, or is based on a trusted framework, that's especially good.