Heuristics for Evaluating Testing Tools

Created by James Bach and Michael Bolton

In one sense, evaluating a test tool is like evaluating any other software product; there are common notions of quality criteria, models for covering the product with testing, and techniques to apply. Those can be found in the Heuristic Test Strategy Model[1]. In this document, we'll present some special considerations that you may want to apply to evaluating tools focused on and marketed towards testing.

In this document, "tool" will refer to a piece of software or a suite that is intended to aid you in testing. "Product" will refer to the product under test.

Capabilities and Power

	Features and algorithms. What does the tool actually do, and how does it do it? Do the tool and its documentation make those things reasonably clear and transparent? How does the tool help you find problems in your product?
	Performance and reliability at scale. Can the tool handle all the data and all of the operations you throw at it from a full-blown project? Is it stable? Does it recover gracefully from failure?
	Logging and reporting. Are there usable, parseable output logs from the tool? Can you easily discover what the tool actually did, after it did it, so that you can debug its behaviour? Have you reviewed the reports it can produce?
	Multi-user / social features. How would this tool enable a team of testers to work together collaboratively, either locally or off-site? Are there commenting or voting or chat features?
	Modeling and Test Design Support. Does the tool assist you in analyzing and modeling the product, then creating tests from those models? Can it help you with special forms of test design such as combinatorial testing, state-based testing, data flow testing, etc.?
	Test Oracles. In what ways can it help you detect real bugs? What sort of verification methods or parameters does it allow you to use?
	Creating and wrangling data. Can the tool help you to generate data? To vary it in useful ways? To store and curate it? Does it help you import and reformat large data sets?
	Support for data-driven testing. Can you connect tables, files, or databases to the same procedure to perform many interesting variations of that test?
	Modifying and extending test procedures. Can the flow of a task be easily edited and adapted to fit new or revised functions and features in your product? Can coders get under the hood and modify test behaviour easily—in non-proprietary and reasonably common programming languages?
	Version control and history. Does the tool allow you to put test artifacts under version control? Does the tool allow you to examine the history of each artifact? Can earlier versions of each artifact be recovered as needed? Does it autosave?
	Coverage analysis and traceability. Does the tool help you understand what has been covered? Can its coverage reports be exported?
	Instrumentation. Does the tool help you to probe or monitor elements of your product that might be hard to detect or analyze via experiential testing alone?

[1] https://www.satisfice.com/download/heuristic-test-strategy-model

Effort to Operate

	Test design and development. How hard is it to create real tests? How hard is it to build suites of related tests without too much duplication?
	Test repair and maintenance. Can procedures or data be reviewed and changed easily if change is needed? What happens when trivial changes to the product invalidate a lot of tests?
	Monitor and analyzing tool behaviour. Can you tell what the tool did? Does it provide you with enough data to investigate bugs?
	Interoperation with other tools. Does the tool play well with other tools you use? Does it work with your pipeline?
	Mass editing. Sometimes you need to make little changes in a lot of test artifacts. What facilities does the tool have for doing so?
	Administrative features. Have you reviewed the user management and settings features? What if testers leave or move around the organization? Are you able to reassign their work?

Feasibility of Adoption

	Availability of users. Are there willing and ready users for the product? Do they have skills necessary to use the tool? Will they need coding skill? Will the tool be welcomed by people who want to do serious testing? Will users feel coerced into using it?
	Adoption curve. Can you incrementally evaluate and adopt the tool, or will it force you to jump in all at once?
	Flexibility and configurability. Can the tool be adapted quickly and easily to fit your product, your workflow, and your culture?
	Compatibility with product technology. Does it support the platforms necessary to your product? Is it portable to new ones that you may adopt later on?
	Compatibility with current test processes and tools. Does the tool support your development setup? Does the tool support your current models of testing? Might it influence them in productive or not-so-productive ways?
	Hosting / availability / data security. Where is the tool's data kept? Are privacy rules being maintained? Is your testing being used to train someone else's machine learning model?
	Adoption and provisioning costs. Tools often come with hidden costs of getting started related to learning, training, data migration, equipment, and infrastructure. Not all of those costs may be monetary or technological; there may be psychological and social costs as well.
	Offboarding cost (in case we bail out) You need to know how to abandon the tool if that becomes necessary.

Learning, Troubleshooting, and Support

	Transparency and simplicity. Do the design and documentation of the tool allow you to know what it can and can't do? Beware of magic, proprietary algorithms that claim to test everything.
	Reference documentation. Are there rich, detailed references available for each feature in the tool? Does the documentation provide examples? Is there a troubleshooting guide?
	Tutorial documentation. Have you been through the tutorial?
	Community of users. How active is the user community? Will it be able to help you?
	Availability of training. Is there training available for the tool?
	Availability of consulting. Is there a marketplace where you can hire help if you need it?
	Availability of technical support. How well does the vendor's own technical support work?
	Responsiveness to bug reports. It's hard to know this except later—but will the vendor take your bug reports seriously and fix the problems you might complain about?

Common Syndromes to Beware

	The 20/80 Rule. Most tools seem to focus on 20% of the testing problem that is easy to solve, as long as you devote 80% of your time and energy to operating their tool—leaving you 20% of your time accomplish the remaining 80% of the work.
	Tools that discourage you from experiencing your own product. Automated checking at the GUI level tends to discourage testers and teams from actually using their own product as part of testing it (i.e. experiential testing). This means you will miss bugs that can't be found with an automated oracle.
	Claims of increased efficiency. When a vendor claims to reduce testing time by any specific percentage, that should have no meaning to you. Whatever meaning it might have, you can't know without detailed explanation, but it will be based on some claim about *your* testing which they can't possibly know is true. Are efficiency gains comparing your oranges to the vendor's apples, or to someone else's *rotten* apples?
	Claims without evidence. Some vendors hide their claims behind a demo wall, neither showing you the tool in action nor describing it in detail. Instead, the pitch is expressed in terms of word magic. Step right up!
	Executive-only demos. Tool salespeople love showing tools to people who don't actually do the work, and who aren't disposed to be critical of the claims and the demonstration. Make sure the people doing the work are in the room, and that they're asking challenging questions.
	Sunk cost bias. When a tool has become entrenched in your process, it may be hard for you to abandon it. When a tool has become entrenched in the organization's culture *and it is expensive*, it may be very hard to convince management to abandon it.
	Deprecating deep testing. Deep testing tends to take time; effort; determination; preparation; rich and varied data; a requisite variety of activities. Beware of tools that undermine any of those things and that direct you toward shallower testing.
	Claims of "AI". Much of the time, "AI" is just a synonym for "software". But someone claims that the tool uses or apply machine learning, how does the training data used to create the ML model relate to your product or data? It's possible that AI can be helpful, but it cannot make qualitative evaluations of the software, and it cannot be responsible. Do not rely on "AI".
	Procrustean Tools. It is common for vendors to force you to use their vocabulary and theories of testing when you adopt their tools. This may lead to systematically alienated and demoralized testers.
	What kind of testing will it NOT support? No tool does everything you might need to do.
	What kinds of bugs will it NOT help you to find? No tool or technology can help you to find every bug. On the other hand, respect vendors who frame expectations in reasonable ways.