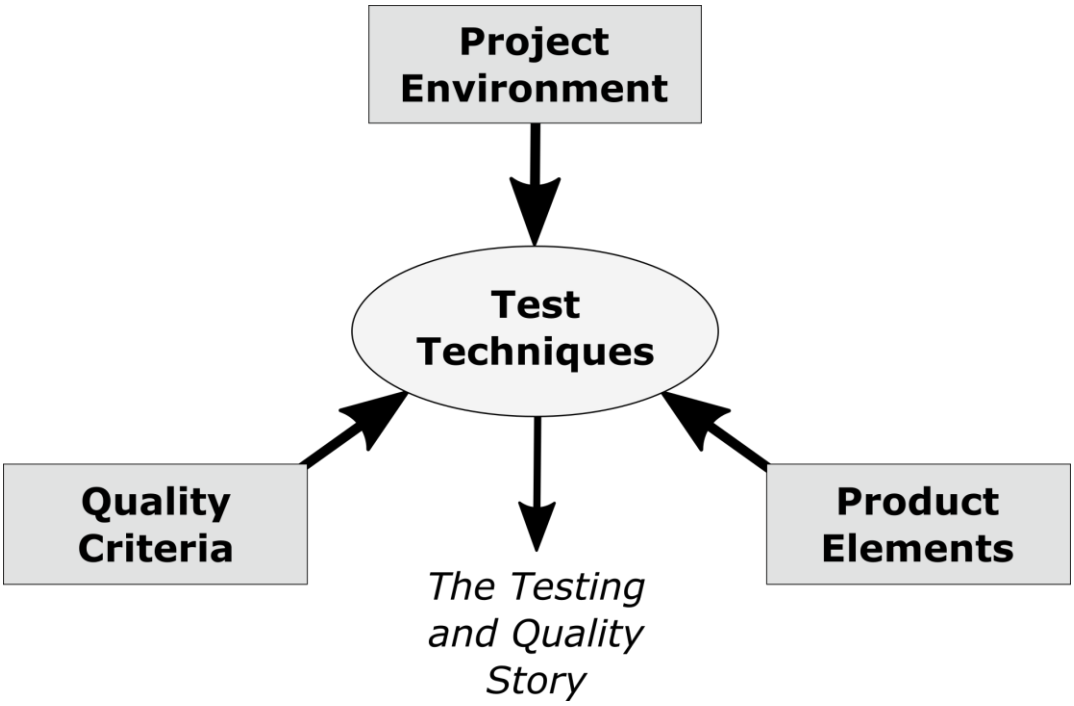
Designed by James Bach
james@satisfice.com
www.satisfice.com
Version 5.7
11/1/2018

Heuristic Test Strategy Model

The **Heuristic Test Strategy Model** is a set of patterns for designing and choosing tests to perform. The immediate purpose of this model is to remind testers of things to think about when they are designing, creating tests, and performing. Ultimately, it is intended to be customized and used to facilitate dialog and direct self-learning among professional testers.



Project Environment represents a set of context factors that include resources, constraints, and other elements in the project that may enable or hobble our testing. Sometimes a tester must challenge constraints, and sometimes accept them.

Product Elements are aspects of the product that you consider testing, including aspects intrinsic to the product and relationships between the product and things outside it. Software is complex and invisible. Take care to cover all of it that matters, not just the parts that are easy to see.

Quality Criteria are dimensions by which people determine the value of the product. Any threat to a quality criterion is a basis for suspecting a problem with the product. Quality criteria are subjective and multidimensional and often hidden or contradictory.

Test Techniques are heuristics for designing tests. The choice to apply a particular technique requires some sort of analysis of project environment, product elements, and quality criteria.

The Testing and Quality Story is the result of testing. You can never know the "actual" quality of a software product— you can't "verify" quality— but by performing tests, you can make an assessment, and that takes the form of a story you tell (including bugs, curios, etc.).

General Test Techniques

A test technique is a heuristic for designing tests. There are many interesting techniques. The list includes nine families of general techniques. By "general technique" we mean that the technique is simple and universal enough to apply to a wide variety of contexts. Many specific techniques are based on one or more of these families. And an endless variety of specific test techniques may be constructed by combining one or more general techniques with coverage ideas from the other lists in this model.

Function Testing

Test what it can do

1. Identify things that the product can do (functions and sub-functions).
2. Determine how you'd know if a function was capable of working.
3. Test each function, one at a time.
4. See that each function does what it's supposed to do and not what it isn't supposed to do.

Domain Testing

Divide and conquer the data

1. Look for any data processed by the product. Look at outputs as well as inputs.
2. Decide which particular data to test with. Consider things like boundary values, typical values, convenient values, invalid values, or best representatives.
3. Consider combinations of data worth testing together.

Stress Testing

Overwhelm the product

1. Look for sub-systems and functions that are vulnerable to being overloaded or "broken" in the presence of challenging data or constrained resources.
2. Identify data and resources related to those sub-systems and functions.
3. Select or generate challenging data, or resource constraint conditions to test with: e.g., large or complex data structures, high loads, long test runs, many test cases, low memory conditions.

Flow Testing

Do one thing after another

1. Perform multiple activities connected end-to-end; for instance, conduct tours through a state model.
2. Don't reset the system between actions.
3. Vary timing and sequencing, and try parallel threads.

Scenario Testing

Test to a compelling story

1. Begin by thinking about everything going on around the product.
2. Design tests that involve meaningful and complex interactions with the product.
3. A good scenario test is a compelling story of how someone who matters might do something that matters with the product.

Claims Testing

Challenge every claim

1. Identify reference materials that include claims about the product (tacit or explicit). Consider SLAs, EULAs, advertisements, specifications, help text, manuals, etc.
2. Analyze individual claims, and clarify vague claims.
3. Test each claim about the product.
4. If you're testing from an explicit specification, expect it and the product to be brought into alignment.

User Testing

Involve the users

1. Identify categories and roles of users.
2. Determine what each category of user will do (use cases), how they will do it, and what they value.
3. Get real user data, or bring real users in to test.
4. Otherwise, systematically simulate a user (be careful—it's easy to think you're like a user even when you're not).
5. Powerful user testing is that which involves a variety of users and user roles, not just one.

Risk Testing

Imagine a problem, then look for it.

1. What kinds of problems could the product have?
2. Which kinds matter most? Focus on those.
3. How would you detect them if they were there?
4. Make a list of interesting problems and design tests specifically to reveal them.
5. It may help to consult experts, design documentation, past bug reports, or apply risk heuristics.

Automatic Checking

Check a million different facts

1. Look for or develop tools that can perform a lot of actions and check a lot of things.
2. Consider tools that partially automate test coverage.
3. Consider tools that partially automate oracles.
4. Consider automatic change detectors.
5. Consider automatic test data generators.
6. Consider tools that make human testing more powerful.

Project Environment

Creating and executing tests is the heart of the test project. However, there are many factors in the project environment that are critical to your decision about what specific tests to create. In each category, below, consider how that element may help or hinder your test design process. Try to exploit every resource.

Mission. *Your purpose on this project, as understood by you and your customers.*

- Why are you testing? Are you motivated by a general concern about quality or specific and defined risks?
- Do you know who the customers of your work are? Whose opinions matter? Who benefits or suffers from the work you do?
- Maybe the people you serve have strong ideas about what tests you should create and run. Find out.

Information. *Information about the product or project that is needed for testing.*

- Whom can we consult with to learn about this project?
- Are there any engineering documents available? User manuals? Web-based materials? Specs? User stories?
- Does this product have a history? Old problems that were fixed or deferred? Pattern of customer complaints?
- Is your information current? How are you apprised of new or changing information?
- Are there any comparable products or projects from which we can glean important information?

Developer Relations. *How you get along with the programmers.*

- *Rapport:* Have you developed a friendly working relationship with the programmers?
- *Hubris:* Does the development team seem overconfident about any aspect of the product?
- *Defensiveness*: Is there any part of the product the developers seem strangely opposed to having tested?
- *Feedback loop:* Can you communicate quickly, on demand, with the programmers?
- *Feedback*: What do the developers think of your test strategy?

Test Team. *Anyone who will perform or support testing.*

- Do you know who will be testing? Do they have the knowledge and skills they need?
- Are there people not on the "test team" that might be able to help? People who've tested similar products before and might have advice? Writers? Users? Programmers?
- Are there particular test techniques that someone on the team has special skill or motivation to perform?
- Who is co-located and who is elsewhere? Will time zones be a problem?

Equipment & Tools. *Hardware, software, or documents required to administer testing.*

- *Hardware:* Do you have all the physical or virtual hardware you need for testing? Do you control it or share it?
- *Automated Checking*: Do you have tools that allow you to control and observe product behavior automatically?
- *Analytical Tools*: Do you have tools to create test data, design scenarios, or to analyze and track test results?
- *Matrices & Checklists*: Are any documents needed to track or record the progress of testing?

Schedule. *The sequence, duration, and synchronization of project events*

- *Test Design:* How much time do you have? Are there tests better to create later than sooner?
- *Test Execution*: When will tests be performed? Are some tests performed repeatedly, say, for regression purposes?
- *Development:* When will builds be available for testing, features added, code frozen, etc.?
- *Documentation*: When will the user documentation be available for review?

Test Items. *The product to be tested.*

- *Scope:* What parts of the product are and are not within the scope of your testing responsibility?
- *Availability*: Do you have the product to test? Do you have test platforms available? Will you test in production?
- *Interoperable Systems:* Are any third-party services required for your product that must be mocked or made available?
- *Volatility:* Is the product constantly changing? How will you find out about changes?
- *New Stuff:* Do you know what has recently been changed or added in the product?
- *Testability*: Is the product functional and reliable enough that you can effectively test it?
- *Future Releases*: What part of your testing, if any, must be designed to apply to future releases of the product?

Deliverables. *The observable products of the test project.*

- *Content*: What sort of reports will you have to make? Will you share your working notes, or just the end results?
- *Purpose:* Are your deliverables provided as part of the product? Does anyone else have to run your tests?
- *Standards:* Is there a particular test documentation standard you're supposed to follow?
- *Media:* How will you record and communicate your reports?

Product Elements

Ultimately a product is an experience or solution provided to a customer. Products have many dimensions. Each category, listed below, represents an important and unique element to be considered in the test strategy. Testers who focus on only a few of these are likely to miss important bugs.

Structure. *Everything that comprises the physical product.*

- *Code:* the code structures that comprise the product, from executables to individual routines.
- *Hardware:* any hardware component that is integral to the product.
- *Non-executable files:* any files other than multimedia or programs, like text files, sample data, or help files.
- *Collateral:* anything beyond software and hardware that is also part of the product, such as paper documents, web links and content, packaging, license agreements, etc.

Function. *Everything that the product does.*

- *Application:* any function that defines or distinguishes the product or fulfills core requirements.
- *Calculation:* any arithmetic function or arithmetic operations embedded in other functions.
- *Time-related:* time-out settings; periodic events; time zones; business holidays; terms and warranty periods; chronograph functions.
- *Security-related:* rights of each class of user; protection of data; encryption; front end vs. back end protections; vulnerabilities in sub-systems.
- *Transformations:* functions that modify or transform something (e.g. setting fonts, inserting clip art, withdrawing money from account).
- *Startup/Shutdown:* each method and interface for invocation and initialization as well as exiting the product.
- *Multimedia:* sounds, bitmaps, videos, or any graphical display embedded in the product.
- *Error Handling:* any functions that detect and recover from errors, including all error messages.
- *Interactions:* any interactions between functions within the product.
- *Testability:* any functions provided to help test the product, such as diagnostics, log files, asserts, test menus, etc.

Data. *Everything that the product processes.*

- *Input/Output:* any data that is processed by the product, and any data that results from that processing.
- *Preset:* any data that is supplied as part of the product, or otherwise built into it, such as prefabricated databases, default values, etc.
- *Persistent:* any data that is stored internally and expected to persist over multiple operations. This includes modes or states of the product, such as options settings, view modes, contents of documents, etc.
- *Sequences/Combinations:* any ordering or permutation of data, e.g. word order, sorted vs. unsorted data, order of tests.
- *Cardinality:* Numbers of objects or fields may vary (e.g. zero, one, many, max, open limit). Some may have to be unique (e.g. database keys).
- *Big/Little:* variations in the size and aggregation of data.
- *Noise:* any data or state that is invalid, corrupted, or produced in an uncontrolled or incorrect fashion.
- *Lifecycle:* transformations over the lifetime of a data entity as it is created, accessed, modified, and deleted.

Interfaces. *Every conduit by which the product is accessed or expressed.*

- *User Interfaces:* any element that mediates the exchange of data with the user (e.g. displays, buttons, fields, whether physical or virtual).
- *System Interfaces:* any interface with something other than a user, such as other programs, hard disk, network, etc.
- *API/SDK:* Any programmatic interfaces or tools intended to allow the development of new applications using this product.
- *Import/export:* any functions that package data for use by a different product, or interpret data from a different product.

Platform. *Everything on which the product depends (and that is outside your project).*

- *External Hardware:* hardware components and configurations that are not part of the shipping product, but are required (or
- optional) in order for the product to work: systems, servers, memory, keyboards, the Cloud.
- *External Software:* software components and configurations that are not a part of the shipping product, but are required (or
- optional) in order for the product to work: operating systems, concurrently executing applications, drivers, fonts, etc.
- *Internal Components:* libraries and other components that are embedded in your product but are produced outside your project.
- *Product Footprint:* The resources in the environment that are used, reserved, or consumed by the product (memory, filehandles, etc.)

Operations. *How the product will be used.*

- *Users:* the attributes of the various kinds of users.
- *Environment:* the physical environment in which the product operates, including such elements as noise, light, and distractions.
- *Common Use:* patterns and sequences of input that the product will typically encounter. This varies by user.
- *Disfavored Use:* patterns of input produced by ignorant, mistaken, careless or malicious use.
- *Extreme Use:* challenging patterns and sequences of input that are consistent with the intended use of the product.

Time. *Any relationship between the product and time.*

- *Input/Output:* when input is provided, when output created, and any timing relationships (delays, intervals, etc.) among them.
- *Fast/Slow:* testing with "fast" or "slow" input; fastest and slowest; combinations of fast and slow.
- *Changing Rates:* speeding up and slowing down (spikes, bursts, hangs, bottlenecks, interruptions).
- *Concurrency:* more than one thing happening at once (multi-user, time-sharing, threads, and semaphores, shared data).

Quality Criteria Categories

A quality criterion is some requirement that defines what the product should be. By thinking about different kinds of criteria, you will be better able to plan tests that discover important problems fast. Each of the items on this list can be thought of as a potential risk area. For each item below, determine if it is important to your project, then think how you would recognize if the product worked well or poorly in that regard.

Capability. *Can it perform the required functions?*

Reliability. *Will it work well and resist failure in all required situations?*

- *Robustness:* the product continues to function over time without degradation, under reasonable conditions.
- *Error handling:* the product resists failure in the case of errors, is graceful when it fails, and recovers readily.
- *Data Integrity:* the data in the system is protected from loss or corruption.
- *Safety*: the product will not fail in such a way as to harm life or property.

Usability. *How easy is it for a real user to use the product?*

- *Learnability:* the operation of the product can be rapidly mastered by the intended user.
- *Operability:* the product can be operated with minimum effort and fuss.
- *Accessibility:* the product meets relevant accessibility standards and works with O/S accessibility features.

Charisma. *How appealing is the product?*

- *Aesthetics:* the product appeals to the senses.
- *Uniqueness:* the product is new or special in some way.
- *Necessity:* the product possesses the capabilities that users expect from it.
- *Usefulness:* the product solves a problem that matters, and solves it well.
- *Entrancement:* users get hooked, have fun, are fully engaged when using the product.
- *Image:* the product projects the desired impression of quality.

Security. *How well is the product protected against unauthorized use or intrusion?*

- *Authentication:* the ways in which the system verifies that a user is who he says he is.
- *Authorization:* the rights that are granted to authenticated users at varying privilege levels.
- *Privacy:* the ways in which customer or employee data is protected from unauthorized people.
- *Security holes*: the ways in which the system cannot enforce security (e.g. social engineering vulnerabilities)

Scalability. *How well does the deployment of the product scale up or down?*

Compatibility. *How well does it work with external components & configurations?*

- *Application Compatibility:* the product works in conjunction with other software products.
- *Operating System Compatibility:* the product works with a particular operating system.
- *Hardware Compatibility:* the product works with particular hardware components and configurations.
- *Backward Compatibility:* the products works with earlier versions of itself.
- *Product Footprint:* the product doesn't unnecessarily hog memory, storage, or other system resources.

Performance. *How speedy and responsive is it?*

Installability. *How easily can it be installed onto its target platform(s)?*

- *System requirements:* Does the product recognize if some necessary component is missing or insufficient?
- *Configuration:* What parts of the system are affected by installation? Where are files and resources stored?
- *Uninstallation:* When the product is uninstalled, is it removed cleanly?
- *Upgrades/patches:* Can new modules or versions be added easily? Do they respect the existing configuration?
- *Administration:* Is installation a process that is handled by special personnel, or on a special schedule?

Development. *How well can we create, test, and modify it?*

- *Supportability:* How economical will it be to provide support to users of the product?
- *Testability:* How effectively can the product be tested?
- *Maintainability:* How economical is it to build, fix or enhance the product?
- *Portability:* How economical will it be to port or reuse the technology elsewhere?
- *Localizability:* How economical will it be to adapt the product for other places?