Testing

STEP-BY-STEP TEST DESIGN

How to quickly produce thorough tests for individual product features

by Kathleen A. Iberle

QUICK LOOK

- A simple method for creating test cases
- Using brainstorming and tables to produce test scripts

"Hey Joe…They just added a frambibulator to the Web site—we need some testing for that!" Testers hear this sort of thing pretty often. When faced with short development cycles and partial specifications, how can we come up with a reasonably thorough set of tests in a reasonable amount of time?

I've used one basic test design method on a variety of products over the last ten years, on everything from cardiography products to printer drivers to e-business applications. The method is intended to produce a decent set of tests—not for a whole product, but for an individual feature. It can be used for either black box or glass box testing, as you will see.

This basic test design method consists of the following six steps:

1. List test requirements based on the specifications (written or unwritten)
2. Add test requirements for a range of inputs
3. List a test type for each test requirement
4. Review test types and fill in the holes
5. Write a test case for each test requirement
6. Group test cases into test scripts

Key principles:

- Lists and tables are easy for the designer to handle
- Reviewing test requirements against test types forces the designer to look at the feature from a variety of perspectives
- A variety of perspectives leads to a variety of interesting tests

I have found that even neophytes using this method can get satisfactory test coverage. By "coverage" I'm really referring to the number of nasty post-release surprises you never even thought of testing for. This doesn't mean that all the bugs are found, and it doesn't even mean that all the code has been executed. It means that the problems most hated by your customers have been found prior to release. (Life-critical software would, of course, need additional testing using more deterministic methods.) This method can be used by an individual or by a small group; the small groups tend to cover more ground because their members encourage each other to get creative.

The Step-by-Step Method

The goal of this method is to start by generating as many test ideas as you can, with as much diversity in approach as you can muster. Later the tests can be cut back to a reasonable number—but right now let your imagination run wild.

Focus on the test *idea*, not the actual test itself. Don't worry about precisely stating inputs or outputs, or about possible overlap. Just get as many test ideas as you can from as many perspectives as you can manage.

1) List test requirements based on the specifications

Start by listing the most obvious test requirements. Test requirements, as defined by Brian Marick in his book *The Craft of Software Testing*, are useful sets of input that should be tested. Remember that test requirements are not exact statements of input and expected output, but are *ideas* of what should be tested.

The first step in generating test requirements is to ask yourself, "What is this feature supposed to do? How would I know if it did that?" Sometimes the answers come from written specifications and other times from interviewing the developers.

Here's a sample interview for the *Mirrored Printing* feature of a typical inkjet printer.

Tester: "So what's this mirrored printing supposed to do?"
Developer: "It makes T-shirt transfers. Print something you like on this special paper and it can be ironed onto a T-shirt."
Tester: "How do I turn it on?"
Developer: "Just check the *Flip Horizontal* checkbox in the printer settings when you print."
Tester: "How does it work?"
Developer: "Oh, it just flips the entire bitmap over."
Tester: "Is there any interaction with other printer settings?"
Developer: "Not really."
Tester: "Is it the same for all three operating systems?"
Developer: "Well, the flipping module is, but the user interface part is different for each operating system."
Tester: "You mean the UI for Win95 and Win98 are different?"
Developer: "Oh, no. There's one for Win3.1 and Windows for Workgroups, another one for both Win95 and 98 versions, and one for NT. Then, of course, all of them get localized."

The first sentence of the interview defines user requirements for the feature—what does the user want to do, and why? The rest of the conversation centers around finding out how the feature is used and some of the possible interactions of the feature. The tester generally follows this up by playing with the feature itself to determine more of the probable specifications.

From this interview, it is apparent that there are at least two modules—the flipping module and the user interface module. The flipping module appears to be pretty independent of the rest of the settings, aside from *Flip Horizontal*.

So, what *is* this feature supposed to do, from the user's point of view? Simple: Print a reversed bitmap on transfer paper. What did the engineers say that the product would do? They described a checkbox for *Flip Horizontal*, so we'll add that to the list, as seen in Table 1.

TEST REQUIREMENT
Prints a reversed bitmap on transfer paper
Flip Horizontal turns on from the user interface (UI)

TABLE 1 Users' and engineers' requirements

2) Add test requirements for a range of inputs

Next, take a close look at the inputs. Do you know what the inputs are?

This can take some thinking for a feature. For mirrored printing, the file to be printed is obviously an input. It is less obvious that the checkbox *also* provides an input.

Now add test requirements to cover the domain of each input adequately. Usually this means:

- a sample average value
- values at the boundary conditions
- values beyond the boundary conditions
- illegal inputs
- error conditions

The idea of boundary conditions as "biggest and smallest values" often confuses the tester when the input isn't a simple number. Look for parameters that are big, small, or somehow unusual or unique. In the case of a bitmap, what are the "biggest and smallest values"? Both the size and the resolution of the image define the size of the bitmap. Bitmaps that fill the page completely or are too big for the page may create problems at the edges. This leads to several new test requirements.

The checkbox has only two values, so the concept of "biggest and smallest" doesn't yield any additional test requirements. But is there any way that the *checkbox* could cause error conditions? One of the common failures with switches of all types is that they switch to a new state, but don't

switch back again—which creates another test requirement. The list of "Common Software Errors" in Appendix A of Kaner, Falk, and Nguyen's *Testing Computer Software, Second Edition* is a good source of ideas on error conditions, if somewhat dated.

Some people may argue that this is not truly an error condition. They may be right—but it doesn't matter. The object of the game is to have lots of good test ideas, not to generate them in any specific order or relationship. If your train of thought leads to any other interesting test requirements, add those too.

It is important during this stage to brainstorm freely. Avoid thinking about exactly how you are going to turn each test requirement into a real test (see Table 2). Try to make notes and let it go.

TEST REQUIREMENT
Prints a reversed bitmap on transfer paper
Flip Horizontal turns on from the UI, and prints reversed OK
Flip Horizontal turns on, then off
Works OK with a bitmap less than a full page
Works OK with 8.5 x 11-inch bitmap
Works OK with 11 x 8.5-inch bitmap
Works OK with least number of bits in bitmap
Works OK with most number of bits in bitmap

TABLE 2 Further brainstorming on test requirements

3) List a test type for each test requirement

The next trick in thinking of many different test requirements is to widen your perspective. Different kinds of testing are needed to find different kinds of bugs. Experienced test designers recognized this long ago, and started to keep lists of different kinds of useful test types to help them in future projects; those lists reminded them to look for a wide range of bugs and test the product from a variety of angles. The test types are used heavily in designing system testing and are thus often called *system test types* or *test types* (see sidebar). I use the test types to force me to look at the item to be tested from a variety of angles.

Reliability testing is a typical example of why this test type approach is useful. Reliability testing looks for performance failures to become evident under long-term use. For instruments, printers, and computers, this usually means leaving the system operating for hours or days—running sequential jobs or tasks without rebooting, restarting, or otherwise "cleaning up." This is exactly opposite to good testing practice, which entails getting a "clean start" for each test so that failures are easily diagnosed. This is one reason reliability testing is often overlooked. But reliability testing is important, and it is notorious for revealing memory leakage problems which go unseen in the usual test process.

TEST REQUIREMENTS	TEST TYPE
Prints a reversed bitmap on transfer paper	Functionality
Flip Horizontal turns on from the UI, and prints reversed OK	Functionality
Flip Horizontal turns on, then off	Functionality
Works OK with a bitmap less than a full page	Functionality
Works OK with 8.5 x 11 bitmap	Functionality
Works OK with 11 x 8.5 bitmap	Functionality
Works OK with least number of bits in bitmap	Functionality
Works OK with most number of bits in bitmap	Functionality

TABLE 3 Listing a test type for each test requirement

At this point in the test design process, I usually find that most or all of my test requirements are ordinary run-of-the-mill functionality tests (see Table 3). The next step, however, will force me to expand my test coverage from a variety of angles.

4) Review test types and fill in the holes

Using your favorite list of test types, go through each test type and consider whether it is applicable to this particular feature. Try to generate as many test requirements as you can. You might also go back to the developers and ask their opinions on how well the feature should do in these areas.

- **Compatibility** Will mirrored printing act differently if different applications are used to generate the bitmap? (No, because it really doesn't start until the bitmap has already been created.)
- **Configuration** Would it make any difference if there were different hardware attached? (I don't think so.)
- **Conformance** Does mirrored printing need to conform to any external standard? (Nope.)
- **Error Conditions** What could go wrong? What if it runs out of memory—will it fail gracefully? (This, then, is a new test requirement.)
- **Localization** What could go wrong in different countries? Mirrored printing is not affected by two-byte fonts, because it's already in bitmap form. What about a full-page bitmap on A4 paper? (That's a bit longer than A-size paper, and strange edge conditions may show up when you print.)
- **Performance** Is there a minimum time in which the flip must occur?
- **Recovery** If we really abuse the system, like turning the power off, will it still work? (Nothing special about this feature, so no test requirement.)
- **Reliability** Will the system continue to run, day after day? What if the modules in this feature leak memory or resources? (That's a good test requirement.)
- **Security** Are there security features for mirrored printing? (No.)

- **Stress** What about running huge files through this? (That would be the same test as "most number of bits.")
- **Serviceability** Any special concerns for the technical support staff? And the help files should be correct. (There's another test requirement.)
- **Usability** Does the user interface make sense? (How about trying to print a T-shirt by following the Help files?)
- **User Scenario** Act like a real user. (Print a real bitmap on real transfer paper and iron it onto a real T-shirt.)

Notice in Table 4 that we now have fourteen test requirements! This is pretty impressive, given that we started with only a few sentences of verbal specifications. Here we see these fourteen requirements matched to test types.

It is common to find that only three or four of the test types are applicable to a given feature. Test requirements for these test types usually increase the total number of test requirements by about 25 percent. (Mirrored printing is a useful example because it has lots of educational test requirements that are not strictly "functionality," and fewer than usual functional test requirements.)

Kaner, Falk, and Nguyen present some additional ways to think of a lot of test requirements quickly in the "Lists" section of the chapter on "Components of Test Planning Documents" in their book *Testing Computer Software, Second Edition.*

5) Write a test case for each test requirement

The next step is to turn your test ideas into actual test cases. For each test requirement, consider how to force the conditions. What are the inputs and outputs? Will you need special input files? What kind of configuration should the test be run in? How will the tester be able to tell whether the test has passed or failed? This precise description is usually referred to as a test specification, test case specification, or simply a test case.

I find that it is easiest to continue working with the table at this point. It's still too early to be attempting to write a "push-this-button" test script (if indeed that's even appropriate), because you will lose some opportunities to combine tests for efficiency.

The steps to follow:

- Specify inputs and expected outputs for each test requirement.

TEST REQUIREMENTS	TEST TYPE
R1. Prints a reversed bitmap on transfer paper	Functionality
R2. *Flip Horizontal* turns on from the UI, and prints reversed OK	Functionality
R3. *Flip Horizontal* turns on, then off	Functionality
R4. Works OK with a bitmap less than a full page	Functionality
R5. Works OK with 8.5 x 11 bitmap	Functionality
R6. Works OK with 11 x 8.5 bitmap	Functionality
R7. Works OK with least number of bits in bitmap	Functionality
R8. Works OK with most number of bits in bitmap	Functionality
R9. Runs out of memory while flipping	Error Conditions
R10. Full-size A4 bitmap on A4 paper	Localization
R11. 8.5 x 11 bitmap on A4 paper	Localization
R12. *Flip Horizontal* checkbox works OK in different languages	Localization
R13. Doesn't leak memory or resources during flip	Reliability
R14. Uses mirrored printing by following *Help* instructions	Usability

TABLE 4 Even more test types for test requirements

Test Types

The earliest account of system test types appears to be in Glenford Myers' *The Art of Software Testing* (1979). Similar lists are found in almost every testing book. A well-written and somewhat more recent description of the test types is found in Chapter 11, "Software Testing," of Watts Humphrey's *Managing the Software Process.*

The test types are basically a compendium of rules of thumb from experienced test designers, and as such vary somewhat from author to author. The useful test types also vary according to the type of product. Pre-1990 books almost exclusively cover large custom business applications, which differ in some important ways from other types of products. I've created a test type list for each major type of product I've worked on, and they've all been slightly different. For instance, many of the classic texts include "installability" as a test type. Large custom business applications were typically installed only once and the installation was likely done manually, so a simple functionality test of the installation often sufficed. For printer drivers, installation is considered to be a feature—and we find that configuration testing, recovery testing, and localization testing of the installer are all necessary in addition to functionality testing.

System test types are very closely related to customer requirements that are not strictly about desired functionality. The same classifications are frequently used for both system test types and customer requirements. Some authors refer to these as *attributes,* which you hope your feature possesses. The IEEE standards consider the term *feature* to contain the idea of performance, reliability, compatibility, and so forth—a definition which is rarely used in practice. Frequently, attributes are listed as part of the "quality" of the product. Capabilities and *"ilities"* are also terms in use for the same idea. Whatever the term, customers have shown again and again that they care deeply about properties beyond simple functionality.

- If specific setups or tools are needed, specify those (see test requirement R13 in Table 5).
- Where there are no conflicts, combine test requirements into a single case. Often user interface switches can be tested while testing the feature that they switch.
- Don't over-combine these test requirements—you'll lose track of what the test is supposed to test. Three test requirements per test case is usually the upper limit.
- Vary the inputs. If several test configurations work equally well, alternate between them.

In Table 5, it's clear that printing a reversed bitmap (test requirement R1) will involve using the *Flip Horizontal* checkbox (test requirement R2), so the same test case will cover both. The test case needs a specific input, so I chose an ordinary, average case: a less than full-size bitmap. Now several requirements (R1, R2, and R4) are covered by one test case: T1. (In real life, I would choose a sample file that produced a less than full-size bitmap and list the sample file in this table as well.)

In test case T3, I was able to combine test requirement R8 (most number of bits in a bitmap) with test requirement R5 (an 8.5x11 bitmap), since obviously a full-sized bitmap is necessary to get the highest number of bits. It's not actually sufficient to force the condition—a high-resolution print mode must also be chosen when printing the test file, as is noted in the Input.

6) Group test cases into test scripts

The final step is to group test cases into scripts or procedures. There are also some opportunities here to make efficient choices.

- Group test cases with common setups and inputs together into test scripts.
- Keep manual test scripts short enough to execute in two to three hours; the goal is to avoid having unfinished scripts at the end of the day.
- If you think of any more interesting test ideas during this process, go ahead and add them into your tests.
- If using manual test scripts, include the test requirements in the document so the tester knows what the test is trying to test.

In Table 6, you can see that I grouped test case T1 (Prints on transfer paper with realistic bitmap) and test case T10 (Uses mirrored printing by following *Help* instructions) into the same test (test script A). This is the test that most closely mimics real customer use, and the only one that will use the expensive transfer paper. The ideal test would also require an ironing board, an iron, a T-shirt, and a tester not already familiar with this feature.

TEST REQUIREMENTS	TEST TYPE	INPUT	EXPECTED OUTPUT	TEST CASE
R1. Prints a reversed bitmap on transfer paper	Functionality	Less than full-size bitmap	Reversed bitmap	T1
R2. *Flip Horizontal* turns on from the UI, and prints reversed OK	Functionality	Less than full-size bitmap	Reversed bitmap	T1
R3. *Flip Horizontal* turns on, then off	Functionality	Different bitmap	Reversed bitmap	T2
R4. Works OK with a bitmap less than a full page	Functionality	Less than full-size bitmap	Reversed bitmap	T1
R5. Works OK with 8.5 x 11 bitmap	Functionality	Full-size bitmap	Reversed bitmap	T3
R6. Works OK with 11 x 8.5 bitmap	Functionality	Full-size bitmap in "medium res" or "normal" print mode	Reversed bitmap	T4
R7. Works OK with least number of bits in bitmap	Functionality	Tiny bitmap in "low res" or "econofast" print mode	Reversed bitmap	T5
R8. Works OK with most number of bits in bitmap	Functionality	Full-size bitmap in "high res" or "best" print mode	Reversed bitmap	T3
R9. Runs out of memory while flipping	Error Conditions	Low memory machine	Fails gracefully	T6
R10. Full-size A4 bitmap on A4 paper	Localization	Full-size A4 bitmap	Reversed bitmap	T7
R11. 8.5 x 11 bitmap on A4 paper	Localization	Full-size 8.5x11 bitmap	Clips map	T8
R12. *Flip Horizontal* checkbox works OK in different languages	Localization	Full-size A4 bitmap	Reversed bitmap, UI OK	T7
R13. Doesn't leak memory or resources during flip	Reliability	Needs leakage tool	No leakage	T9
R14. Uses mirrored printing by following *Help* instructions	Usability	Medium-size bitmap	No user confusion	T10

TABLE 5 Creating test cases to cover all test requirements

Test case T2 (Flip checkbox works repeatedly) turns into a short test script that focuses on proper behavior of the user interface.

Test cases T3, T4, T5, and T9 are grouped together into test script C—a script that consists of printing various bitmaps with a U.S. paper tray and then checking for resource leakage. Similar test cases that involve special equipment (A4 paper and paper tray, a low-memory computer) are grouped around the equipment needs.

When This Method Doesn't Work

If the item under test is a specific software module rather than a feature, this method can still be used with some changes. The inputs are the parameters (and global variables and data read from files or databases, etc.) and the outputs are the return variables (and global variables and data read from files or databases, etc.). If you are doing glass box subsystem testing, Brian Marick's *The Craft of Software Testing* describes a technique very similar to this one which is better tailored to glass box testing.

This method should not be used by itself for life-critical or safety-critical software. Thorough unit and module testing prior to system test is necessary in these cases, and a more sophisticated approach to system test may well be warranted.

This method is also not appropriate for situations in which there are a large number of independently varying inputs that are believed to interact with each other, because this method doesn't offer any advice on how to cut the possible tests down to a reasonable number. There are other techniques that should be used for combination testing. This method is much more focused on ferreting out interesting tests from a wide variety of perspectives.

A small number of configurations can be handled in this method by adding another column to the table for test configurations. For our mirrored printing example, a typical configuration issue is the different operating systems in the Windows family. The test configurations are considered as part of the test requirement, before grouping into test scripts. This generally requires putting both the test case and the test script column into the table, which can get confusing if the table is very big. (Table 7 shows an attempt to synthesize all of the information we've used in our previous tables.)

Choosing a subset of all of the possible configurations usually requires at least some research into the structure of the program. In this case, we were already told by the developers that the flipping module itself was believed to be independent of the operating system. The user interface module is different for three classes of operating systems:

Version A: Win 3.1, Windows for Workgroups
Version B: Win 95/Win95B/Win98
Version C: WinNT4, WinNT5

Because of this, test requirements that are testing the flipping module can be run on only one sample from any of the three classes. Test requirements that deal with the UI must be run on one sample from each class. It's important to keep an eye out for test requirements that test more than just the module in question. For instance, the memory leakage can be affected not only by whether the module cleans up after itself, but whether operating system calls clean up after themselves. Therefore, it ought to be run on one sample from each of the three classes.

The end result in this example is that the test script E can be run on just one operating system with relatively low risk, and the test scripts A, B, C, and D must be run three times each—once for each operating system family.

Tools for Success

I've used this test design method with tables, with spreadsheets, and with a

TEST CASE	INPUT	EXPECTED OUTPUT	TEST SCRIPT
T1. Prints on transfer paper with realistic bitmap	Less than full-size bitmap	Reversed bitmap	A
T2. Flip checkbox works repeatedly	Different bitmap	Reversed bitmap	B
T3. Largest number of bits	Full-size bitmap in "high res" or "best" print mode	Reversed bitmap	C
T4. Landscape orientation	Full-size bitmap in "medium res" or "normal" print mode, landscape orientation	Reversed bitmap	C
T5. Small number of bits	Tiny bitmap in "low res" or "econofast" print mode	Reversed bitmap	C
T6. Run out of memory while flipping	Low memory machine	Fails gracefully	D
T7. Full-size A4 paper with localized UI	Full-size A4 bitmap	Reversed bitmap, UI OK	E
T8. 8.5 x 11 bitmap on A4 paper	Full-size 8.5x11 bitmap	Clips map	E
T9. Doesn't leak memory or resources during flip	Needs leakage tool	No leakage	C
T10. Uses mirrored printing by following *Help* instructions	Medium-size bitmap	No user confusion	A

TABLE 6 Grouping test cases into test scripts

Microsoft Access database. As a test planner, the database was by far the most useful. A typical project involves hundreds or thousands of individual test requirements, grouped into hundreds of test procedures. I could search the database by feature or by test type and get a quick feeling for how much testing was planned in a particular area or for a particular test type. I could search for words describing a particular feature in the test requirements, and then retrieve all of the associated test scripts.

The database was a drawback in certain respects. It wasn't seen as "true" test documentation by people more accustomed to working with narrative documents, and thus transferring projects to new staff could get messy. The database also tended to break down every time Microsoft released a new version of any related software, so the maintenance cost was pretty high. Although I've looked repeatedly over the last six years for commercial tools that would support this method of test design, I have yet to find any.

However you apply the framework—tables, spreadsheets, or databases—this is a simple test design method that can be used in a variety of situations for black box feature testing. It's basically a structured method of brainstorming test requirements. The test designer focuses on thinking up lots of interesting test requirements before getting into detailed, precise test design. A checklist of system test types is used to encourage the designer to think about testing and the item under test from a variety of angles. STQE

Kathy Iberle (kathy_iberle@hp.com) *is a senior software engineer at Hewlett-Packard. She has worked in both development and testing, and she currently consults within several HP divisions.*

TEST REQUIREMENTS	TEST TYPE	TEST CONFIGURATION	INPUT	EXPECTED OUTPUT	TEST CASE	TEST SCRIPT
R1. Prints a reversed bitmap on transfer paper	Functionality	3.1, 9x, NT	Less than full-size bitmap	Reversed bitmap	T1	A
R2. *Flip Horizontal* turns on from the UI, and prints reversed OK	Functionality	3.1, 9x, NT	Less than full-size bitmap	Reversed bitmap	T1	A
R3. *Flip Horizontal* turns on, then off	Functionality	3.1, 9x, NT	Different bitmap	Reversed bitmap	T2	B
R4. Works OK with a bitmap less than a full page	Functionality	Any	Less than full-size bitmap	Reversed bitmap	T1	A
R5. Works OK with 8.5 x 11 bitmap	Functionality	Any	Full-size, high-resolution bitmap	Reversed bitmap	T3	C
R6. Works OK with 11 x 8.5 bitmap	Functionality	Any	Full-size bitmap in "normal" or "medium res" print mode, landscape orientation	Reversed bitmap	T4	C
R7. Works OK with least number of bits in bitmap	Functionality	Any	Tiny bitmap in "low res" or "econofast" print mode	Reversed bitmap	T5	C
R8. Works OK with most number of bits in bitmap	Functionality	Any	Full-size bitmap in "high res" or "best" print mode	Reversed bitmap	T3	C
R9. Runs out of memory while flipping	Error Conditions	3.1, 9x, NT	Low memory machine	Fails gracefully	T6	D
R10. Full-size A4 bitmap on A4 paper	Localization	Any	Full-size A4 bitmap	Reversed bitmap	T7	E
R11. 8.5 x 11 bitmap on A4 paper	Localization	Any	Full-size 8.5x11 bitmap	Clips map	T8	E
R12. *Flip Horizontal* checkbox works OK in different languages	Localization	3.1, 9x, NT	Full-size A4 bitmap	Reversed bitmap, UI OK	T7	D
R13. Doesn't leak memory or resources during flip	Reliability	3.1, 9x, NT	Needs leakage tool	No leakage	T9	C
R14. Uses mirrored printing by following *Help* instructions	Usability	3.1, 9x, NT	Medium-size bitmap	No user confusion	T10	A

TABLE 7 Mirrored printing example with operating system configurations included