

Measurement & Analysis

L

O

O

▶▶ QUICK LOOK

- **Methods to collect and store data**
- **Enhancing your test plan with the information you gather**

Analyzing software documentation to avoid poor test coverage

by Kenneth Lengel

Before You Test

K

When I fly, I usually prefer a window seat with a view of the landscape. Some flyers, of course, take that penchant for panoramas one step further—leaping out into the clouds with a parachute and

a prayer, thriving on the adrenaline rush provided by such a bold feat.

Even these daredevils will tell you that before they board that aircraft they have to review many things

in preparation for their leap. And while poor test coverage hardly has the same impact as a chute that fails to open in flight, both can result in—to say the least—a bumpy landing. The good news is that *both* situations can usually be avoided by proper preparation. Just as skydivers should look before they leap, testers should get used to looking before they test.

Test coverage is about insuring that test plans and test cases include information vital for successful testing of the program in the areas of functionality, performance, and the overall quality of the software. In addition, test managers who prepare test plans that provide proper test coverage can avoid the wrath of a project manager whose implementa-

tion has just gone sour or an angry customer whose system has just crashed.

(Note that test coverage is not the same thing as *code* coverage. Code coverage measures how the tests have exercised the code, e.g., which lines of code have never been executed. But you can exercise every line of code and still miss something important—like figuring out that the program doesn't work at all on Windows 2000.)

Test coverage requires information—about how the program installs, how fast the program accesses and processes data, and how the program appears on the monitor. These are just a few examples of the kinds of things a tester needs to know

about a program's functionality and performance in order to provide appropriate test coverage. This article is about gathering and using that information.

Think this documentation is relevant only in formal test plans? Ad hoc testers, too, can benefit from this careful information gathering—even when they're not following formal testing processes and procedures. Ad hoc testers must still understand how the software is supposed to work. If they don't use supporting documentation, they'll rely on guesses or on their memory of how the program *used* to work. Guesses can be wrong, and the way the program worked last year may not be the way it's supposed to work today.

However, gathering information about a program just for the sake of collecting information does *not* improve your test coverage. Adequate test coverage involves a systematic approach that includes analyzing the available documentation for use in test planning, execution, and review. In order to come up with a successful strategy to improve test coverage, you'll need to do three things:

- create a plan of attack to provide strong test coverage
- determine the scenarios for the test plan
- manage the changes made to information used by testing

Implementing this strategy requires that you, the test manager, think about *people* as much as about documents—taking into account the interests of the rest of the project team, and using people skills to encourage cooperation between teams. To this end, part of the manager's role is to serve as “knowledge manager”—demonstrating how to share and store the team's knowledge, and then using that knowledge to improve the organization's methods.

A Plan of Attack

In that role, your first step toward strong test coverage is figuring out how to gather the most information you can about the functionality and

performance of the program—documentation that will determine what and how to test. Let's say you're the test manager for Acme Printing, charged with testing a new print utility program that will print flyers for marketing companies. In order to test the program correctly, you must evaluate various documents for information that should be included in Acme's test plan. Your plan of attack should address several questions:

- Which documents do you need to create your test plans?
- Who owns these various documents, and what's the best way to access them?
- How will you collect and store this data?
- What is the best method to update plans as documents change?

Let's look at ways to address a few of these issues.

Required Documents

There are various types of documents that you should compile in order to create test plans and test scripts. While we do not have the time within this discussion to outline in great detail these four documents, let's briefly describe them and their usefulness in preparing for testing efforts.

1 Requirements are statements that describe the method by which software will meet the business needs of the user. These statements cover the functional and system specifications, as well as the business rules and quality attributes of the program. Here are some sample requirements for Acme's flyer printing program:

- operating system: must work with any Windows 9X platform
- appearance: must print company logo at the top and bottom of each flyer
- time completion performance: must take less than thirty seconds to print any flyer
- printers: must work properly on recommended list of Hewlett-Packard and Epson printers

Outlining and understanding these requirements enables a test manager to know what areas to test and cover to insure software quality (we know, for example, that there should be at least one performance test). Reviewing these requirements will also guide the test manager in acquiring software (versions of Windows) and hardware (printers).

2 Use Cases define a sequence of actions completed by a system or user that provides a recognizable result to the user. A use case that defines what happens when a user attempts to log into the Acme flyer printing system—and what actions will be taken by the system in response to the user's input—might start like this:

- 1.1 The program will ask for user name and password
- 1.2 The user will enter a six-character alphanumeric username
- 1.3 The user will enter a six-digit numerical password
- 1.4 The program will provide a list of available flyers to print
- 1.5 ...

A complete use case would also define the messages that will appear if the user enters an incorrect username or password. Use cases provide a clearer picture of what the customer is expecting the product to accomplish. Employing these use cases can reduce ambiguity and vagueness in the development process and can, in turn, be used to create very specific test cases to validate the functionality of a program.

3 Change Requests are documents that refer to modifications requested by a user of the program (and that user could be a developer, tester, or the end user for whom the product is ultimately created). A change request for our flyer printing program might be to print the company logo at only the top of the flyer—rather than at both the top and bottom as stated in the requirements. Change requests should be tracked by a test manager to make sure that

PROGRAM DOCUMENTATION	PRODUCT	MOST CURRENT	LAST REVISED DATE
Requirements Document	Flyer Print Utility	1.2	6/10/99
Use Case Document	Flyer Print Utility	1.01	7/14/99
Change Request	Flyer Print Utility	FPU 10	7/5/99
Incident Report	Flyer Print Utility	none	none
Test Plan	Flyer Print Utility	2.1	7/19/00

FIGURE 1 Sample database grid

testers are not relying on their past knowledge of the program’s functionality, and that the testers are aware of the changes to make to their test plans and scripts. Test coverage without this complete knowledge invites failure of your testing efforts.

4 Incident Reports (more commonly known as “bug reports”) are vital to the success of any testing exercise. These reports can help testers know where bugs have been found previously during testing. Review these reports to make certain that known issues do not recur when future changes are made to the program. To avoid these lapses in testing (known as regression gaps), use the knowledge gained from previous bug reports to modify the test plans. This will insure that future test efforts are always able to locate similar errors.

Compiling the list of these documents is a crucial step to insuring that you’ve given a product proper test coverage. However, the only way these documents can be used effectively to create test plans and test scripts is if they are collected, maintained, and made accessible to those who need them.

Collecting and Storing the Data

Your method for collecting and maintaining changes to these documents for your test plan is another ingredient in providing great test coverage. There are both technical and personal issues at stake when discussing the collection of information.

From a technical standpoint, there are a few issues that need to be addressed in the collection of these documents—such as where to put them, how to keep track of them, and who will have access to these documents.

The easiest place to maintain these records is in a shared folder on a network drive. For example, a test manager can create a public folder on the network that includes the latest test plans being used for testing purposes. A development team can give the testers access to requirements documents, use cases, and change requests for test plan development.

In order to track these documents, a simple database can be used to designate only the *location* of the needed documents for testing. Do not take ownership of a document that is not created by your team. In the database (see Figure 1,), designate the fields to include the types of documents available to you, the most current version number of the document, and the last date the document was revised. If you create your test plan based on this information, a quick look at the information in the database for the program in question will reveal whether or not you need to make changes to your test plans.

Writers will be more willing to share if they are assured that the document won’t be modified. Use your normal procedures for version control, file protections, or word processor document protection features to provide that assurance.

There are also personal issues. Documentation is a funny thing...we never have the right amount of it, no one ever has time to complete it, and even fewer people want to let every-

one see *theirs*. Let’s look at three areas that can be addressed to minimize the reservations people might have to sharing their documentation.

Research There is nothing worse than asking people to provide information that has already been made available. In order to minimize the tensions regarding documentation, search through any and all intranet sites within your organization for the information you need. Chances are that the person you would ask for this information has been asked many times before and has provided a means for anyone to access the information.

Negotiation Diplomatic skills are needed to drive away the reluctance people have to share information regarding their work. All of us would admit that most of the technical documentation we create is not something we would submit to a professor for a final exam grade. Negotiation requires a test manager to find ways to provide advantages for the testers in the gathering and sharing of information. One way to negotiate for additional information for testing is to ask the project manager for any information outlining the system, business, or performance requirements of the software. In return, offer the created test plans to the project manager for review after its completion. This accomplishes two distinct tasks. First, the testers get the needed information to create a test plan with proper coverage. Second, the project manager has confidence in the work being done by the testers. A true win-win situation. Test managers must encourage document creators (whether they are pro-

ject managers, business analysts, programmers, or testers) that there is added value to sharing the same information with everyone who may need to understand the product.

Helping Hand In our group, we have a tradition of “system expert presentations.” In these presentations, testers with expertise about a product provide other testers with information—product overviews, operational and functional uses of the product, technical information about the product, troubleshooting techniques, and frequently asked questions. Increased knowledge leads to better test plans. In particular, knowledge of all the programs that make up a system is important for integration testing. We open these presentations to other teams as well, allowing testers to offer assistance outside of the normal testing effort. Lending such a helping hand goes a long way in making sure teams work together.

Research, negotiation, and extending a helping hand are just a few of the techniques that help test managers coordinate the knowledge that is helpful to testers, programmers, and help desk and training staff alike to produce a better product and proper test coverage.

Test Plan Scenarios

Now it’s time to begin writing your test plan. Considering the volume of documents you have reviewed, creating a test plan must seem like an insurmountable task. The best way to begin writing the test plan is to focus on the scenarios that you have captured from your analysis of the documentation.

For example, you know many things about what the product should do. Here are two sample scenarios that you could use to create appropriate test cases for testing:

- You know from the requirements that the company logo should be on both the top and the bottom of each page. But you also know, from reviewing change requests, that this requirement has been superseded by a new rule that logos should appear only on the top of each page. A test that checked for top-and-bottom printing would incorrectly signal a failure. You should ensure that your test plan now contains a test that checks that the logo only appears on the top.
- One of the documents you examined might have been a bug report about envelope windows. The requirements did not mention that flyers are to be folded and placed in envelopes with clear windows. The bug was that sometimes the street address wasn’t visible through the window. You should ensure that your test plan contains tests that check whether addresses are always visible.

The test plan should include every important statement of fact in all the documents you’ve researched, as well as a test scenario that validates the information.

You can also use documentation to find limits for your testing efforts. For example, from the requirements document, you know that you need only test on Windows 9X, not on Windows 2000. You also know that only certain printer models from Hewlett-Packard and Epson matter. (But suppose you’ve discovered a series of bug reports about NEC incompatibility—and you see that all those bugs have been fixed. You now

know that the requirements document is out of date.)

Managing the Changes

One of the most difficult areas to manage in testing is the never-ending changes made to the product’s requirements and other documents that represent intended functionality and performance.

The best way for a test manager to deal with the changes is to assume the role of a *change management advocate*, insuring the proper current flow of this information into testing documentation. Test managers are natural points of contact for this information, since they are already charged with estimating the amount of time and effort required to fix each change request submitted for approval. The test manager/change management advocate can channel all change requests to the test team, and should provide a copy of the change request to the tester who is responsible for updating the test plan. By doing this, the test manager/change management advocate can accomplish two tasks: keeping on top of the additional time needed for testing if the change is approved, and making certain the changes are added to the test plan.

Changes to the test plans should include the use of several tracking methods. First, each test plan should include a *Revision History Chart* (see Figure 2) that designates when changes have been made, by whom, and why. When writing and changing test plans, it’s a good practice to have the updated plan reviewed by the business analyst or the development team’s project manager—this shows them how the testers believe the document changes have affected the test plan.

TEST PLAN VERSION	DATE REVISED	AUTHOR	CHANGE DESCRIPTION	REASONS FOR CHANGE
1.00	06/23/00	BML	Initial Version	
1.10	06/29/00	BML	Update	Added Requirements
1.20	07/10/99	BML	Revision	Comments from Project Manager
1.30	07/15/99	BML	Revision	Change Request

FIGURE 2 Revision history chart

A tester should also refer to the database of documents to verify, as previously discussed, that the documents used for testing have not been updated since the latest version of the test plan. One way to head off this problem is to create a change request when changes are made to any of the documents that are used for testing. The change request language should be simple and to the point, such as “Added Requirement 1.4.2 to Flyer Print Utility Requirements Document.”

One of the biggest risks for test managers is not planning for the time required for revisions and reviews; these should *always* be accounted for in project planning processes and schedules. Plan for 5% of your total test plan creation time per test cycle as a good estimate of the time needed for revisions. For example, if a tester takes twenty hours to create a test plan and you are planning three test cycles, the

total time needed to make revisions should be approximately three hours. (Note that while three hours seems like a small amount of time, it has increased the total time needed for planning efforts by 15%.) Planning for these revisions can reduce the stress of your testers and increase the success in meeting implementation deadlines.

Closing Thoughts

We all know that documentation is vital to testing an application or system correctly—and that it’s rarely kept up to date. Even worse, some testers avoid using the documents that could help with testing efforts because of team members’ resistance to tracking, updating, and compiling that data. This “avoidance behavior” creates unneeded battles and broken communication between programmers and testers about how the program is to be tested.

We can’t expect anyone to test what they don’t know. But ignoring the documentation because it is incomplete, time-consuming to review, or difficult to access only causes a vicious cycle to continue—resulting in uninformed testing, poor test coverage, and a product with questionable quality. Looking before you test, and creating a test plan that clearly defines your testing scope and goals, will help you avoid rough landings when your team’s product hits the streets. **STQE**

Ken Lengel (klengel@home.com) is Senior Project Manager at Workscope, Inc., an industry leader in HR self-service for large employers. He has over ten years of experience in software process improvement, system integration, and software testing. He enjoys writing, mentoring, and solving problems to improve business processes.