How to Write a Great Software Test Plan

by Robert Japenga

With complex software systems, you can never test all of the functionality in all of the conditions that your customers will see. Start with this as a fact. You will never test enough!. And that at least, should cause a certain amount of discomfort. But if this fact makes you a little uncomfortable, you have taken the first step at creating a great test plan.
Step 2 in getting started is to read and re-read The Art of Software Testing by Glenford Myers. This classic will set the stage for understanding some of the strategies and methodologies for testing software. He provides a good definition of a number of terms we will be using in this article. Myers makes a simple statement that most of us miss when we test:
[Software] Testing is the process of executing a program with the intent of finding errors.
There is big difference between testing to show that something works versus testing to show that something fails.
Step 3 is to follow these guidelines and you will be home free.

Why Write a Software Test Plan?

A better question is why plan anything? You plan those things that you know are more complicated than what you can do by the seat of your pants. You plan things for which order and completeness are important. Even the simplest software system is so complex and is so prone to failure, that planning to test is just as essential as planning your design.
The second major reason is that, as an industry, we are doing a lousy job at releasing quality products. Current industry standards are that tested and released software averages more than ten significant bugs per 1000 lines of code. How many hundreds of thousand of lines was that last project? Thinking ahead and planning your testing is one way to cut that down.
All of these are good starting reasons for having a software test plan.

What is the difference between a Test Procedure and a Test Plan?

Most good software development models call for both a Test Plan and a Test Procedure. Although in principle this sounds good, test procedures are extremely costly to develop, costly to maintain and don't catch enough of the bugs in a short enough time.
Simply put - a test plan tells you what you are going to test while a test procedure tells you how you are going to test it. And a quality test plan, as outlined in this article, will cover a large percentage of systems being delivered today without the need to develop a test procedure.

What should the Test Plan address?

Generally there are two phases of test plan creation.
The first should be done at the time the system is defined. The second should be created in parallel to the Software Requirements Specification.
Software Test Plan Part 1
This part falls under the more conventional definition of software test plans and includes:
1. Definition and Objectives of the Phases of Testing - This should answer the questions:
•Will there be Module Testing? If so, what are the objectives?
•Will there be Integration Testing? If so, what are the objectives?
•Will there be Systems/Acceptance Testing? If so, what are the objectives?
•Will there be Beta Testing? If so, what are the objectives?

2. Schedules - This should answer the questions:
•When will Part 2 of the Test Plan be written for each Phase?
•When will each of the Phases be scheduled?
•What are the dependencies of each Phase?
•How long will each Phase last?

3. Responsibilities - For each phase, the people who will design, write, execute, and verify test

cases as well as the people who correct the problems should be identified.
4. Target Equipment Required - All of the target equipment required for testing should be identified. This is a much neglected part of the process. If two prototypes are required, they need to be planned very early. Will you be testing on one prototype and one pre-production piece of hardware? On how many shifts will testing take place?
5. Test Equipment Required - All of the test equipment required for testing should be identified. Any off the shelf test equipment should be identified as well as any test equipment that will need to be created. The plan should identify what test equipment is already owned and what will need be procured either by purchasing or renting.
Software Test Plan Part 2
This is the part that we have found can replace the Test Procedure. The goal in this phase is to define exactly what should to be tested. And what needs to be tested but the requirements from the great SRS you created (as a result of reading our "How to Write a Great SRS").
Here are the guidelines we follow in creating this important phase.

1.Create a document which is structured identically to the SRS
2.Take every requirement from the SRS and create one or more test cases from that document.
3.Provide a check box beside each test case. This allows the Test Plan to easily become a Test Report.
4.Create stress testing scenarios that step outside of the SRS and look at the system as a whole. For example, if the system has TCP/IP Ethernet connection on it, perform certain tests with large amounts of data being sent to the system. Or, if the system has a keyboard as input, perform testing with the keyboard being pounded with both valid and invalid data.
5.Provide off-design test scenarios outside of the SRS that test the system in conditions for which the SRS provides no definition. Showing that the software does what it's supposed to do is only half the battle. The other half is determining if the software is doing what it is not supposed to do.

Test Case Generation

The following are some general guidelines for creating test cases:

1.Every test case should define expected results as a result of the defined inputs.
2.Every requirement should be tested at and beyond its boundary conditions (for example: high and low limits) as well as mid points.
3.Re-read Myers on Test Case generation.Then re-check your test cases.
4.Mix black box and white (or glass) box testing. This means that some test cases are written without knowing what is inside the box (black box testing). Other testing should include testing based on knowing what is inside. For example, if you know that an algorithm for generating 10,000 different outputs is done with a formula, you will not need to test all 10,000 outputs. However if you know that it is done with a table with some break points in the table where different algorithms are used at each break point, you will want to include this in your testing. Also, white box knowledge can reduce the number of actual tests necessary through equivalence partitioning. (Don 抉 know what equivalence partitioning is? Go To Step 3 and repeat!)

Who should execute the Test Plan?

We have found that it is essential to have a different person or team execute the Test Plan from those who created the software. Find individuals who like to break things! Develop a creative tension between development teams who test each other 担 software.