Nine steps to software testing success

Formal training for software test professionals is surprisingly sparse. Classes on testing are generally part of a larger computer curriculum. Because of this, most testers come from a variety of backgrounds, and may not know how to get started. This paper lists some steps, hints, and tips on testing software.

New ideas

It is much easier for new ideas to come from outside of a project, rather than from within. This is because projects tend to be self-contained little islands, where the same ideas are recycled over and over again. Almost any idea that comes from outside of this insular environment is bound to be a new one, and can lead to needed improvements.

One source of these new ideas is books. Read the first four chapters of several books on testing and software project management, while quickly jotting down anything that sounds like an idea you haven't heard before. This will give you some material to work from when starting your test.

Another possibility is talking with friends, colleagues, and conference attendees that are in the software testing or development field. Regular lunches or just meeting for coffee to bounce ideas around can give you many ideas to try. There probably won't be as many ideas here as in a book, but they are likely to be of higher quality. This is because you'll have more opinions on how well the idea is likely to work in your environment and situation.

Establish the goals

The first step is to establish the goals for the test. Identify the areas of the product that need the most testing, what kinds of hardware and operating system platforms will be required, and the available resources for the project.

Areas that need the most testing

These can be new additions, the core functions, or just places where the developers have been having trouble during unit test. To find these areas, you'll need to talk with as many different people as you can. Customers, developers, and other testers are good sources. The documentation group is also an often-overlooked source of information here. They'll know about any product areas that are particularly complex, and these tend to be the same areas that have defects.

Hardware, software, and platform requirements

This is an important area, as these requirements increase the cost of the test. If running the program on ten different operating systems is important, than you'll need to install them, possibly on ten different machines. These machines and the software will all have to be paid for and obtained prior to the test. Installing the operating systems will also require a certain amount of time before the test starts, and some of the testers may need education on them.

It is important to find out these requirements early in the planning cycle, to allow time for all of this.

Available resources

Resources are more than just machines and software. They include lab space, testers, support personnel, and even time. All of these will affect your ability to conduct the test, and may limit the scope or size of it. To create a plan that works with the available resources, you'll need to know what they are first.

Organize

Now you've assembled lots of information and requirements, but it is in an unorganized, unmanageable form. Organizing this information will help you keep everything straight, make the right decisions, and eliminate any unnecessary or duplicate steps.

Prioritize

Now that everything is organized, take a look at which aspects are the most important for the test. These are the areas that align most closely with your already established goals.

It usually isn't possible to test a product without resource constraints, either in time or personnel, so these priorities should be well defined during the planning stages of the test.

By prioritizing early, the test can be focused toward the most important things.

If tradeoff choices are needed after the test starts, they will be easier to make if the priorities are well defined in advance.

Document the plan

By now you should have all the data, organization, and prioritization needed to create a full test plan. The plan is a collection of these items, written in a clear format that allows members of the test as well as other groups to understand the test requirements, goals, priorities, and resources.

Inspect

Testing is like anything else of any complexity; it can't be done in a vacuum. Now that the test plans are defined and written, make them available for inspection, not just by your peers, but also by other groups that are affected by the test like the product development, translation, and documentation groups.

Prepare

Use the plan to set up all of the environments required for your test. Parcel out the test cases and make assignments. Make sure each environment, or test cell, is ready. After the test starts, there will be little time to install operating systems or requisition a needed machine, so be careful and complete.

A spreadsheet or table will help with this. Try something like this:

Test case	Owner	Test cell	Start (planned/ actual)	Finish (planned/ actual)	Percent complete
001: Saving resources to a file	SPL	3	2/5/04	4/15/04	0%
002: Context-sensitive help	JEC	6	1/15/04 1/20/04	3/15/04	25%
003: Limit testing on search window	DER	8	1/5/04 1/4/04	2/15/04	50%
004: Test installation from previous release	DLG	2	11/05/03	12/01/03 12/15/03	100%

By using a table and making it available, the expectations are clear from the beginning. Everyone knows what they are responsible for, which machine they are going to run it on, and when it has to be done.

The percentage column tracks the progress of each test, and can be used to indicate when a test is finished (100%). Never raise the percentage past zero if the test hasn't actively started, and never mark it 100% until all defects found have been fixed and it is really over. The numbers between zero and 100% are a rough estimate, and are not exact. They are there to show approximate progress toward the goal, not to measure things exactly.

Act

Now that you have a fully-inspected plan, prepped test cells, a schedule, and assignments, it is time to start. Keep the list of priorities and the test plan close at hand to keep the test on track and focused. If new requirements come in during the test, check the priorities. The priorities shouldn't change during the test, so they can be used to ensure that any changes to the plan don't derail the

goals. Resource changes, time changes, and new test requirements should all be measured against the earlier priorities.

In-process improvements can be made as the test goes on, as long as they are fairly small and don't disrupt the test. Larger suggestions for improvements should be recorded for later evaluation and use.

Communicate

Monthly status is fine during the planning stages, but after the test starts, things are happening (and changing) at a rapid-fire pace. Weekly status reports are required to keep up with the changes and make sure all of the other groups (and your own group) understands the work and the time table.

Lack of expected progress is caught early, and gives everyone time to set priorities and restructure resources to help meet the original goals of the plan. More progress then expected may allow time for a few process improvements or additional function, depending on the amount of progress made.

Lessons learned

After the test is over, it is important to do a complete analysis of the entire process, goals, and results. Many lessons can be learned by examining data collected through the release and comparing it to the plan.

Large differences between the actual and planned progress can indicate problems with the process, team, measurement and estimation, support from other groups, and a variety of other things. It is important to involve all the key people in this analysis, including those from other groups. The goal here isn't to assign blame, but to find the rough spots in the process and improve them.

Bad example

We didn't make enough progress on the interface testing because Bob installed the wrong operating system prior to the test. Bob should work harder next time to make sure this doesn't happen to us again.

Good example

Some of the machines had the wrong operating systems loaded because the list of requirements was created too late in the cycle. For the next release, we should make sure this is done at least three weeks prior to the test to give Bob the time required to do the work.

Improve the process

This is part of the “lessons learned” step, but it is often forgotten. If you spend the pain required to learn from your mistakes, and the time required to analyze them and write them down, it is a waste to file the list and not implement it. At the beginning of every project development cycle, make reviewing the list of suggested improvements from the last release the first part of your planning. If you do this, each release will be smoother than the one before it.