alodar systems™

Integration Tip

WHITE PAPER

Hallmarks of Good Test Procedures

All of us have probably experienced the project that would not end. Even if the design and development phases came in on schedule and within budget, sometimes the final tweaking drags on and on. New requirements are suggested at the last minute. Bug fixes have unexpected side effects. Exactly what the software can and cannot do gets lost in a haze.

Savvy managers realize that there needs to be a mechanism that allows them to declare unambiguously that a project is completed.

One popular way is to generate test procedures -- step-by-step instructions that exercise all of the promised functionality of the software and record associated performance measures. Get past all of the procedures and the job is done; if there are deficiencies make the fixes and try again.

But what constitutes a good test procedure? Poorly designed ones allow embarrassing flaws to show up after the software is released to production. The time to set up and execute the tests can become prohibitive so that ultimately they are abandoned and the effort to develop them in the first place is wasted.

Test design and execution has been discussed at length in the literature, of course, and the marketplace provides a wide choice of tools such as open source JUnit (http://www.junit.org) that help automate the process. Most of the help available, however, focuses on unit testing --- increasing the quality of the components that are turned over to the integration effort. This tip discusses instead developing and executing test procedures at the system level - test procedures that have two hallmarks - completeness, and easy execution.

Completeness

The primary challenge in developing a test procedure is figuring out what exactly is to be tested. A common mistake is to focus on the software design and develop a procedure that tickles each of the components. A test document generated this way is certainly better than having none at all, but runs the risk of not addressing everything that needs to be.

The correct starting point for test procedures is the requirements document. If the requirements have been properly stated (see How to Analyze Requirements white paper) each one is a statement of a testable capability that the software must have.

A complete test procedure is one that methodically marches through these requirements and proves their satisfaction one by one. A procedure that tests all of the requirements is complete. If the software makes it all the way through the procedure without any discrepancies, then by definition the software does everything it is supposed to do. The project is done.

Simple bookkeeping tools help generate a complete test procedure. First give each requirement in the requirements document a unique identifier. Resist the temptation to go overboard. A simple numbering scheme with increments of 10 to allow for later insertions (i.e. 10, 20, 30) will work fine. Complicated hierarchical identifiers such as 3.2.3.1.7.2 add little insight and create visual clutter.

Second, in the test procedure document, at the conclusion of a step that proves satisfaction of a particular requirement or requirements, denote those requirements in parenthesis or with some other delimiter. For example, a particular step might read "Run diff on the output file and the reference file. Any differences reported constitute a discrepancy (20, 290)".

Third, build a simple tool that extracts and sorts the requirement numbers in a draft test procedure document. Run the same tool on the requirements document and compare the two outputs. Add additional test steps to the test procedure until the two outputs agree. When they do, you have a complete test procedure.

Easy Execution

Good test procedures are easy to execute -- they are exercised by computer based scripts with little manual intervention. For example, if a series of programs are run as part of the procedure, the script starts each program, saves the output and compares it with anticipated results. The test conductor can look at a final output file to see how things fared. Automation can extend to the user interactive portions of a test procedure as well by using tools such as open source Canoo WebTest (http://www.webtest.canoo.com)

But true reproducibility extends much farther than this simple example. For system integrations, ones that perhaps involve several processors interacting in a network, environments on each processor must properly initialized for the tests. Special database tables might need to be created and then deleted at test completion. Temporary user accounts and passwords might need to be established.

Often these ancillary steps are performed on an ad hoc basis during development, or incrementally as a test procedure is run and the additional settings are discovered on the fly. Then, later when the test is run again, these additional steps are rediscovered each and every time. If there are enough of them, retesting becomes too much of an effort and the procedure is abandoned.

Try to anticipate all of the things that need to be done to establish the test environment and include these steps in the procedure and test scripts that automate them.

Often this will require steps that recreate test datasets used by developers. Start the discovery process of these needs early so that test scripts will be complete when testing is to be started.

Of course, most likely, some additional steps will still be discovered the first time the test is run. Diligently record what is done to get through the test and improve the procedure and script for the next iteration. The ideal is a test procedure that is started with a single command and then runs to completion without any additional human intervention.

Why is this important? Because any complete and fully automated test procedure is worth its weight in gold after the project is complete. Invariably there will be bug fixes, vendor package upgrades, and additional requirements placed on the system. All of these changes call into question the continued validity of the original testing.

Does the system still satisfy all of its requirements? Having a complete and easy-to-execute test procedure on the shelf answers this question cost-effectively and provides the confidence to roll the improvements into production.