A Developer Centric Approach to Automated Functional Testing

Introduction

How many times have you worked on a project, where 'all' the tests were going to be automated? Sound familiar? I am going to guess that for many of these projects the automated test effort was dropped because it took up too much time, and the test harness was too brittle to withstand even the slightest change to the application.

In the following document I aim to provide a solution to the above problem which places the responsibility of automated testing with the developer. There are a number of reasons for this, and I will detail them in the following paragraphs.

Common Problems faced when Automating Functional Tests

1. Testers/Management underestimate the effort required to manage and maintain the test harness. The test harness should be treated as a software development project in its own right.
2. Most Testers have never developed software, or have only done so for small projects. Developing software as part of a large development team takes practice.
3. Testers usually aren't allowed to modify the application source code in order to add any hooks needed by the test harness.
4. Testers knowledge of related development processes like SCM, and Design patterns may be limited. This often increases the complexity of the test harness, and decreases the overall quality and maintainability of the test harness.

All of the problems above can be addressed by simply shifting the test harness development responsibility, from the Testers, to the developers. Many developers may be resistant to this, but at the end of the day its their job to produce high quality, thoroughly tested code. The functional tests just take the level of testing performed by developers one level higher, by verifying the various components the developers have unit tested actually interact as expected.

It is important to note that we are using the automated functional tests to perform acceptance and regression tests of the system under test. It is not necessary to write a test for every single relationship between the components of the system. A compromise is needed, for example you could only automate the positive tests – tests that ensure the system behaves as expected when provided with valid input data.

The criteria for what is automated and what isn't automated will differ from project to project and is dependent on a number of other factors like:

- available tools
- development resources available
- project time-frames

Responsibilities

- Management and Testers decide what functional tests need to be automated NOT developers.
- Developers need to take into account the time needed to write the automated tests when providing estimates for development tasks.
- Developers are responsible for providing Testers with a convenient way to run the tests.
- Developers are responsible for the structure and maintenance of the test harness.
- Testers are responsible for providing what ever information the developers need in order to complete a test and make it pass.
- Testers should provide a flow/state document for each automated test, detailing what the developer needs to check in order to validate a requirement.
- Testers are responsible for ensuring the tests written by developers are accurate, this could be in the form of a code review, or simply sitting with the developer as they run through the test having each

part explained to them.
- A Single Developer and Tester need to be responsible for test harness. The developer has control over the structure of the test harness code, whilst the tester guides what tests are included in the test harness and how successes or failures are determined. This pair should also provide mentoring and guidance to the rest of the development and test teams to help the understand the test harness and what is being tested.

Keys to Automated Testing Nirvana

- It should be a simple process to download/build and run the test harness.
- The test harness should lend itself to integration with automated build systems.
- The test harness should produce a report detailing which tests have passed, failed or raised an error. This report should provide enough information to determine the cause of failures and errors, or at the very least a link to the code for the test so further analysis can be done. This report should also provide access to environmental properties like OS, JVM etc.
- It should be possible to pick and choose what automated tests to run.
- If possible choose a test tool that allows you to write tests in the same language as the application under test. This makes it easier to extract data from the application under test.
- The test harness should make use of **functional decomposition**. This means you look at the functionality your application exhibits, breaking it down into distinct actions (be as granular as you can!) and then categorize these actions. You should then model your test harness around these categories and actions.[see appendix for more information on functional decomposition]
- Input and expected output from the application should be stored outside the test harness.
- Component Identifiers should be stored outside the test harness. for example. Widget names, servlet URL's etc.
- It should be possible to update input, expected output data and component identifiers without the need to rebuild the test harness
- When naming actions in the test harness ensure they are declarative, and unambiguous for example. ClickTotalButton. The advantage of using this naming approach is it makes the test scripts easier to read and understand by non-programmers.
- The test harness should provide and audit log of everything it does. The descriptions of these actions should be declarative for example. Clicked 'Total' in Window 'Sales'

The Process (for XP Projects)

1. At begginning of the iteration Testers prioritize each story's success criteria as follows:
    1. Must have automated test
    2. If time permits they would like an automated test for the success criteria
    3. No automated test will be written for the success criteria
2. Developers then estimate how long it will take to implement the functionality for the cards (making sure they add time for the automated functional test development)
3. Developer decide how many cards can be implemented in the iteration based on their estimates
4. Development starts.

Benefits of this Approach

- Hire cost effective Testers
- Leverage your existing development resources
- Code of Test Harness is of same quality as the application under test.
- Code of Test Harness changes in conjunction with the application under test, this means it is always well factored and functional.
- Developers will add hooks to application code to facilitate automated testing, since its their responsibility. Its no longer a request by a third party.
- Decreases test cycle, since the majority of acceptance tests are handled by the test harness freeing Testers to find 'real' bugs and the automated test harness highlighting regressions.

Caveats

- There is always going to be some 'pain' as developers familiarize themselves with the automated test tool of choice, and lay the foundations of the test harness.
- Available tools may need to be modified in order to interact with the application under test. This is often the case when using Open Source tools.

Author

Robert Keith
Dataplan I.T. Ltd
http://www.digitaltester.com
robert.keith@digitaltester.com
November 2004