Accelerate the Move from Waterfall to Agile Development

Introduction

In a traditional waterfall development process, code quality is often controlled primarily with system tests run by a quality assurance (QA) team, that commonly drive the application through its user interface. Today, software development processes for many organizations are transitioning to an Agile development approach, which uses developer-run continuous testing as a key to controlling code quality. This raises the challenge of creating a large volume of unit tests, that can be run automatically and frequently by programmers to help ensure that the code actually does what it is supposed to do. A new generation of JUnit generators delivers unit-level regression tests, also called fast tests, that can provide near-complete unit-level code coverage at the push of a button, providing a substantial boost to development teams that have moved to or are planning to move to an Agile process.

This paper will discuss:

- **Continuous feedback is a key difference between Waterfall and Agile development**
- **Unit testing with CI can provide continuous feedback**
- **Unit testing adds continuous design element**
- **Unit testing techniques and tools**
- **Maintaining continuous visibility**
- **Essential steps for moving from a Waterfall process to an Agile process**

Continuous Feedback is a Key Difference between Waterfall and Agile Development

With a traditional waterfall development process, developers write code (and may write some unit tests that may or may not be maintained with the code and may or may not provide a high level of code coverage), send it to quality assurance (QA) for testing, and wait for the results. QA hopefully finds the bugs; developers then fix the bugs (and re-run unit tests if they exist), send the new code to QA, and wait while QA verifies that the bugs were fixed. This approach wastes time because development is often locked or limited while QA is testing the code, and because of the back-and-forth activity required between development and QA.

Over the past decade, there has been a strong movement towards an Agile approach that breaks tasks into small increments performed by cross-functional teams that are responsible for requirements, design, coding, and testing. A key difference in the Agile approach is that testing starts in early iterations and is done concurrently with coding. To succeed, Agile software development requires that developers receive continuous feedback on the code they have produced.

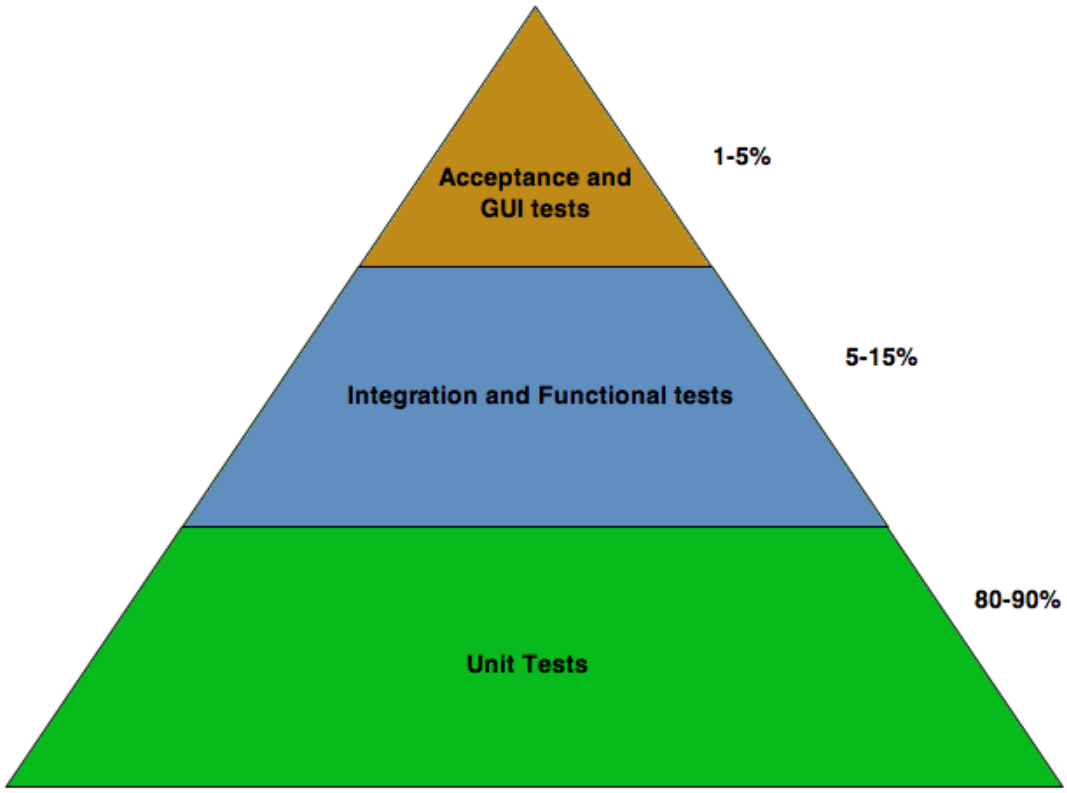
Unit Testing with CI Can Provide Continuous Feedback

The vast majority of testing efforts are typically devoted to system tests that are designed to exercise the system as it is seen by the different users that interact with the software. System tests are usually written by QA, sometimes in a high level specification language. System tests typically drive the application from or close to its input and verify its performance from or close to its output. These tests typically require external resources or a running application and nearly always run too slowly to be run repeatedly as a part of the development process. Another limitation of system testing is that it is difficult to determine which component is responsible for a problem, and drilling into components to find the problem can take considerable time.

On the other hand, unit tests, which are short and specific tests that exercise specific building blocks of the code, are usually the primary testing tool in Agile software development. Unit tests are designed to be used by programmers and are written in a programming language to ensure that the code does what it is supposed to do. Unit tests typically take only seconds to run which makes it possible for the same people who are writing the code to perform them many times a day. When a unit test identifies a bug, it's usually obvious what caused it and how to fix it.

Continuous feedback also requires reliable continuous integration (CI) to trigger test runs on schedules and code check-ins. Without it, continuous feedback falls apart because developers can just 'forget' to run their tests. A CI system keeps everyone honest. Additionally it reduces the time it takes for QA to access a viable build and/or deployment. Unit tests built with automated test generators such as AgitarOne can be scheduled with CI software and run with build automation tools.

The following diagram illustrates a suggested focus on the use of unit tests for an Agile development process.



Unit Testing Adds Continuous Design Element

When they are run on a regular basis as part of the development process, unit tests determine at any moment in time the behavior of the system. The tests serve in effect as a form of detailed design documentation that is maintained in sync with the code base. Developer testing can also add the element of continuous design by steering the code and providing developers with freedom to improve methods, classes and objects while ensuring that the design intent is protected by the suite of automated unit tests. Unit testing also reduces the amount of debugging required by breaking testing down into manageable chunks.

Unit Testing Techniques and Tools

Considerable manpower is required to write unit tests, which, as a rule of thumb, require about three times as many lines as the code they are designed to test. Furthermore, writing unit tests is typically seen as uncreative, low-status and undesirable work on the part of software developers. In addition, conventional hand-authored tests rely upon developers to identify all of the potential scenarios and tend to miss exceptions, unexpected values, and border cases that have the potential to create future regression bugs.

Automated test generation tools such as AgitarOne JUnit Generator help address this challenge by automatically analyzing a Java project and generating tests that capture and preserve the code's behavior. JUnit Generator routinely achieves 80% or better code coverage. These tests leverage mocking technology that, by resolving dependencies on components such as databases, generates tests for code that would otherwise be untestable within unit tests. In cases where the JUnit Generator does not have enough information to provide full coverage, developers can write test helper methods consisting of a few lines of Java code to produce additional unit tests. If developers have manually created unit tests, they can be run by and have results and code coverage reported in conjunction with tests generated by the JUnit Generator.

The following diagram illustrates how an automated tool can summarize unit test results following the generation and execution of tests. The results here include percentage of classes and methods with tests; coverage of the code by those tests; and various other metrics to assist users with analysis of the tests. There is also drill down to detailed results for each class/method.

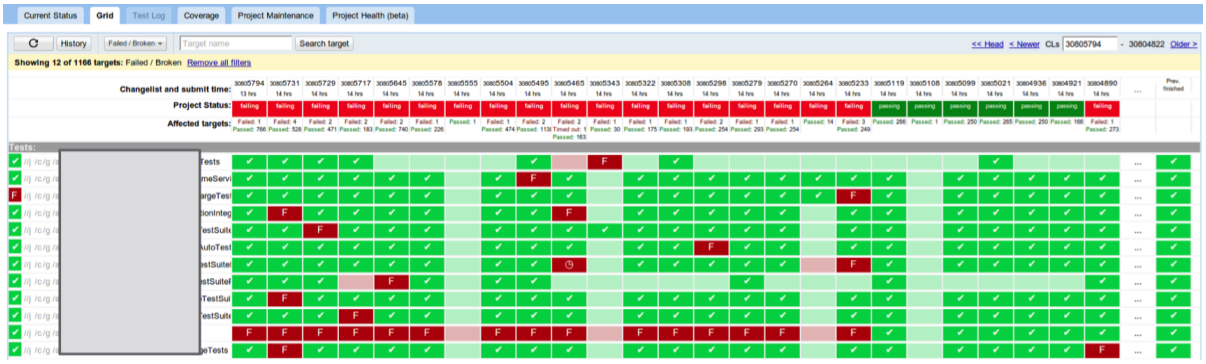
Maintaining Continuous Visibility

Continuous visibility is a physical concept, not just a reporting concept. You should have an actual display of the project status visible to the whole development, QA, and management team for a project. This improves team coherency, accountability and pride. AgitarOne's Dashboard can report on the health of each project and module by assigning a green or red status in addition to reporting more detailed information such as how much coverage is offered by current tests, how many tests are passing and failing, etc.

Essential Steps for Moving from a Waterfall Process to an Agile Process

The following are a suggested set of steps, in addition to the commonly considered development process step changes, to move from waterfall to Agile processes.

1. Set up Continuous Integration to trigger build, test, metrics reporting on code check-ins and a scheduled build.
2. Create unit tests for legacy code. Focus on unit (fast) tests only; not integration, system or functional tests. Delay new development during this process.
3. Reduce sole dependency on functional (slow) tests. If step #2 is complete, you should be able to reduce the frequency and possibly number of slow tests you run in automation or manually from QA, as code coverage is no longer a primary concern of those tests. Consider likely faults and produce a minimum set of slow tests to run with end-to-end usage stories as the goal. This might require that you change the focus of your QA department to focus more on exploratory testing and regression bug follow-up, and less frequently on performing time consuming rote testing plans.
4. Consider integrating personnel roles, of some test team members being involved with development and vice versa.
5. Set up physical continuous visibility. Use a display, or a red and green light, to indicate success/failure of continuous testing using fast tests. If those are not an option, consider using an application or email list to make these results physically visible.



Source: [EclipseCon 2013 - Continuous Integration]

6. Introduce new developer testing tools and techniques through team education. Set up mandates and testing expectations.

Conclusion

Many Agile development teams have ramped up their use of JUnit generators with the goal of expediting and accelerating the move to Agile software development. For example, one of the world's largest financial services institutions has expanded its use of the AgitarOne JUnit Generator from one team to at least twelve teams using the tool over the course of the past four years. This expansion was based on the experience of the original team trying out the JUnit Generator, finding that frequent unit testing increases the speed and reduces risk in the development process by identifying and preventing errors from being built into the code.

For more information about AgitarOne, please visit us at www.agitar.com.