Test-Driven Development and Agile: A Path to Improved Software Quality

Since its creation, the Agile movement has only continued to grow in popularity. The reason for this is simply that, when implemented correctly, Agile works, and its benefits are undeniable. For example, most Agile implementations foster increased communication among stakeholders and improved time-to-market. However, while testing is certainly just as important within the Agile framework as in traditional development, it is not strictly defined, leaving individual teams to determine how best to approach testing-related tasks. This is problematic, because while testing is essential for mitigating risk, it is often undervalued and not considered a priority. As a result, there is a reduction in quality, ultimately risking the reputation of the team or the company.

A solution we support at DCG Software Value (DCG) is to combine Agile with Test-Driven Development (TDD). The synergy between the two frameworks facilitates higher quality software in less time. More importantly, it allows IT to comprehensively support the needs of the business and the end users of the software.

Agile Software Development

Agile software development runs counter to the traditional waterfall methodology that many organizations still have in place. Agile is defined by continuous delivery, focusing on what can be defined, designed, coded, delivered, and tested in stages. These stages, called “sprints,” are short, time-boxed increments, often as little as a week or two. This leads to faster feedback from the stakeholders, increasing communication and reducing the impact if a change is requested. To learn more about Agile, read, “What Are Agile Best Practices and What Does the Future Hold?”

Overall, the end result from implementing Agile is reduced costs, improved quality, and better time-to-market for the features deemed to be the most important.

Test-Driven Development

A traditional development cycle utilizes the “Test Last” method, meaning that most testing takes place after all other stages of development, often right before delivery. Test Last also involves a separation between developers and testers, whereby the developers complete their work and “throw it over the wall” to a tester, separating development from the user experience and ultimately hampering communication. This frequently leads to bottlenecks near the end of a cycle, as issues are discovered that require rolling the product back several stages.

Even within the Agile framework teams can choose to leave testing as the last task in a sprint. Of course, the delays are considerably less than with Waterfall (where testing is completed at the very end of the development process), but bottlenecks are still an issue that impede quality and delivery.

“Test First,” in which unit tests are written before the code, can mitigate bottlenecks. In this case, the test helps to define what the code is meant to do, providing guidance for the developer in terms of user functions. This concept is a natural fit with Agile in two ways:

1. By developing the tests from the requirements, rather than the code, communication increases. The creator of the requirements, the developer, and the tester must collaborate on the tests and the subsequent code, thereby increasing everyone’s understanding of the work at hand.

2. By having the test or test suite written first, there is no need to wait for the testing to be done. The code can be written and tested immediately, especially when automated testing is included in the process (considered a best practice). If the code fails, it can be pushed back onto the backlog, and if it succeeds, the next item can be started.

Test-Driven Development (TDD) is a specific type of “Test First” process. The primary difference is in adding refactoring (step five, below). The steps in TDD are as follows:

This process is sometimes referred to as “red-green-refactor,” where “red” represents writing the test and not passing and “green” is creating the code and passing the test.

Advantages of Test-Driven Development

There are a number of inherent benefits in TDD, including:

- Fewer delivered defects. This is largely due to early testing, which prevents defects.
- Improved communication, a central theme in Agile.
- Higher quality tests, developed by the collaboration of all stakeholders.
- Improved code quality, as the code is generally kept simple as a result of the TDD process.
- Less dead code, primarily due to the refactoring step, which simplifies and cleans up the code. Since each code section is as simple and clean as possible, there is usually less dead code – even late in the application lifecycle.

Disadvantages of Test-Driven Development

Of course TDD does have its share of drawbacks, which should always be a consideration:

- It is a change, and change requires effort. As with any business transformation, there will be resistance, and the added effort needs to be shown to be worthwhile for internal buy-in.
- TDD often begins with no application to run the tests, so it is often necessary to develop stubs (a testing segment for the code to send results to), drivers (something to send results to the code under test), and other extra blocks of code. However, these items are often reusable as the product proceeds, so they may only need to be created near the beginning of the project and occasionally thereafter.
- The testing is not complete. There will always need to be security testing and acceptance testing on most products.
- It requires strong communication between team members, but this is generally true of all Agile teams.
- It usually requires the developers to also do some testing. Then again, developers almost always do some degree of testing, at least a simple check to make sure the code they create works. In fact, it can be argued that testing should be done by developers even with a separate testing team, as it ties the two groups more closely together and can lead to better code.

Other Forms of Test-Driven Development

In the same way that TDD is a refinement of “Test First” development, there are techniques that take TDD further still.

While TDD creates unit tests, Acceptance Test-Driven Development (ATDD) creates acceptance tests before coding begins, based on the team’s understanding of the requirements. This requires even more discussion with the requirements’ authors, creating deeper collaboration and furthering the understanding of the requirements, by the developers and the authors.

Behavior Driven Development (BDD) combines unit tests and acceptance tests within specific contexts. The test often follows this formula:

- Given a context
- When an event happens
- Then an outcome is generated

While TDD can be implemented on its own, without the use of ATDD or BDD, it’s important to consider which framework most appropriately supports a given team or project.

Improved Software Quality and Value

The true value of IT is how well it can support the needs of the business. One of the core strengths of Agile is the increase in communication between testing and development groups, as well as between technical and business teams. Test-Driven Development supports this as well, increasing the lines of communication. It furthers the collaborative environment encouraged in Agile and enables improved understanding of requirements by all parties.

With a better understanding of the requirements from the business, and by collaborating earlier with testers, developers have a more accurate picture of the end result; as a result, they can write cleaner code with fewer attempts. This impacts and even multiplies its effect throughout the lifecycle, reducing defects and dead code down the line. Less time and money needs to be spent in defect fixes and design revisions due to misunderstood or poorly defined requirements.

There are numerous ways in which these two methodologies combine for value-added outcomes. The greatest and truest gains come by way of facilitating a well-interpreted picture of the business objectives and what meeting them will require as early as possible.

Resources

Websites

SPaMCAST 31 – Ambler, Test Driven Development, Words and Change: http://www.spamcast.libsyn.com/s_pa_mcast_31_ambler_test_driven_development_words_and_change

SPaMCAST 295 – TDD, Software Sensei, Cognitive Load: https://tcagley.wordpress.com/2014/06/22/spamcast-295-tdd-software-sensei-cognitive-load/

SPaMCAST 401 – Listening, Quality, Testing and Contract Closure, Developers and Testing: https://tcagley.wordpress.com/2016/07/03/spamcast-401-listening-quality-testing-and-contract-closure-developers-and-testing/

Books

Agile Java: Crafting Code with Test-Driven Development, by Jeff Langr

The Cucumber Book: Behaviour-Driven Development for Testers and Developers, by Matt Wyne and Aslak Hellesoy

Clean Code: A Handbook of Agile Software Craftsmanship, by Robert C. Martin