RedMonk
+1.866.RED.MONK
www.redmonk.com



# Agile QA Automation

**A RedMonk Briefing Document**

## Overview

Agile software development thrives on rapid feedback loops. Few things are less rapid than manual testing. Thankfully, there are several key QA tools in the Agile world that automate and speed-up the development and QA process.

As with Agile in general, these tools eliminate the barriers between development and QA, realizing efficiencies and speed by eliminating the communication and process paths required between the two split efforts.

This document provides a brief overview of Agile QA tools.

## The Role of QA in Agile

In traditional software development the role of QA is to find bugs and missing or incorrectly implemented features in software before it's released to customers. The assumption is that developers will accidently write bugs into the software or incorrectly implement features, including omitting a feature entirely. In both cases, traditional development assumes that the developers cannot find the bugs before the code is handed off to QA[1], as the old QA saying goes, "developers are people who write bugs."

Assuring that the customer gets what they want is a necessary step in software development. But, Agile software development sees that as only *part* of QA's role. Agile QA's responsibilities also include helping eliminate bugs as quickly as possible, even before they're "found."

This new responsibility requires both process changes and new tools to support that process. The process change is QA's involvement in development from the beginning, working along-side development – both figuratively and literally. The most noticeable

---

[1] Episode 10, "Seeing QA's as allies - The Lean-Agile Way" of *Lean Agile Straight Talk podcast*.

practice is test-driven development (TDD) in which tests are written *before* code is written to specify what the code must do.

The remainder of this document cover several tools and related practices that support Agile QA.

## Unit Testing

A unit test is a small piece of code that can be easily and quickly run to verify that changes to a given piece of code have not created new bugs[2]. Unit tests are the first tool that Agile QA typically encounters. Developers maintain and write the bulk of unit tests, but QA shares the responsibility of making sure unit tests are used and maintained.

### JUnit

JUnit and it's equivalents in other languages are the undisputed leaders of unit testing tools. For ease of reference, JUnit and it's variants will simply be labeled "JUnit" for the remainder of this document. These libraries are actually small packages that developers can use to write a series of tests, each containing "asserts" that exercise different paths through code execution.

Aside from being a great tool on it's own, the real benefit of JUnit comes from the wide standardization on JUnit that IDEs and other tools include. For example, Eclipse and NetBeans both provide rich and seamless integration with JUnit.

When it comes to unit testing, any organization should be ready with a rock-solid excuse not to use JUnit.

JUnit is open source and, thus, freely available.

### Agitar

While JUnit is an open source, freely available unit testing tool, Agitar is a commercial unit testing suite. Agitar provides much more than a framework and library for writing tests, it also provides tools for generating unit tests. As with all code-generation tools, Agitar is not a panacea that can substitute for all human written unit tests. Instead, Agitar automates the writing of several common and

---

[2] "What Does Agile Look Like? Briefing Document," RedMonk

uncommon testing scenarios, taking the drudgery out of writing straight-forward tests.

Additionally, Agitar provides "code coverage" reporting that reports on sections of the code that are tested and not tested. For example, Agitar may tell you that only 40% of your code is actually executed during tests indicating that, more or less, 60% of your code is not being tested.

The price for this automation is high compared to the "free" of JUnit and commercial test coverage tools such as Clover. That said, as in build vs. buy decisions, the choice to use Agitar is one of trading money for time: the price of Agitar could buy you more time for your developers and QA to do other tasks instead of write and maintain the tests that Agitar generates.

## Acceptance Tests

Being close to the actual code, unit tests are primarily the responsibility of the development role and highly technical. A different type of automated testing, acceptance tests span both the development and QA role. While a unit test exercises code directly, acceptance tests exercise the software's functionality. A unit test can be thought of as testing methods and objects, while acceptance tests test scenarios and user stories[3].

As with all Agile QA, the goal with acceptance tests is to make them as quick as possible: both to write and run. The dominant tool suite, FIT and FITness, reflect this desire.

### FIT & FITness

Technically, FIT and FITness are two separate tools. FIT is the core framework, while FITness layers a wiki-based UI on-top of FIT. Typically, both are used. This document will refer to the combination of the two as FITness for brevity.

In it's simplest form, testers use FITness to specify a set of inputs, several pieces of functionality to call with those inputs, and the expected outputs. FITness then runs those test cases, reporting on the results. In a very real since, FITness is a tool for writing and then running test cases.

---

[3] User stories can be loosely thought of as use cases. See Mike Cohn's *User Stories Applied* for more.

Ideally, QA writes FITness tests before developers begin coding, providing developers with a clear-cut way to verify if and when they've written their code correctly. Developers will need to write small pieces of "glue code" called fixtures to link together more complex portions of the code base with the FITness tests, but FITness provides many such "fixtures" out of the box. While this may seem like more work than traditional models, the collaboration and test-driven development that it encourages pays off in the end.

FIT and FITness are both open source and, thus, freely available.

## Choosing Agile QA Tools

The primary context to establish when choosing QA tools is the nature of the code being tested: existing code or new code? If the code base is new, using a unit testing framework such as JUnit and/or Agitar along with acceptance testing tools like FITness are an easy and wise choice.

Existing, or "legacy," code is notoriously tedious to test with Agile QA tools. The method of applying test-driven development to legacy code is a slow and steady containment and then elimination or refactoring of the older code. That said, the alternative – not quickly, but manually testing the code – is less favorable than the tediousness of writing those tests and refactorings as needed.

In both cases, developers and the organization as a whole must understand and internalize the practice of test-driven development. Writing tests before code seems ludicrous to those who are comfortable with water-fall methods of software development, but become immediately comforting and effective once used.

Whether your project involves all new code or existing code, the benefits of establishing Agile QA will be immediate and clear: faster feedback loops between writing and verifying code, leading to more flexibility in delivering software that customers value and desire.

## More Resources

Links and books:

- http://fitnesse.org/ - home of the acceptance testing framework.
- http://www.junit.org - home the JUnit and other nUnit frameworks.
- *User Stories Applied: For Agile Software Development*, Mike Cohn.
- *Working Effectively with Legacy Code*, Michael Feathers - strategies for applying Agile QA to existing code bases.
- *Fit for Developing Software: Framework for Integrated Tests*, Rick Mugridge, Ward Cunningham – an introduction and overview of acceptance testing with FIT and FITness.
- Test Driven Development Book

# About the Creative Commons License



# About RedMonk

RedMonk is a research and advisory services firm that assists enterprises, vendors, systems integrators and corporate finance analysts in the decision making process around today's enterprise software stacks. We cover the industry by looking at integrated software stacks, focusing on business and operational context rather than speeds and feeds and feature tick-lists.

Founded by James Governor and Stephen O'Grady, and headquartered in Denver, Colorado, RedMonk is on the web at www.redmonk.com. If you would like to discuss this report email Michael Coté, cote@redmonk.com.