Automated Testing and Agile
Contents

Iterate with Confidence ........................................ 1
Is Automation Worth the Effort? ............................ 3
The Test Automation Manifesto ............................... 5
Continuous Integration ......................................... 7
Regression Testing ............................................... 8
Review at the End of Each Sprint ......................... 9
Plan for Automated Testing ................................. 10
Automated Testing in a Nutshell ............................ 11
Iterate with Confidence

You’re cruising along with Agile, working on daily tasks and developing the first iteration of your product. You’re now probably ready to get into your next sprint. But what happens if you break what you built in the last sprint?

That’s where automated testing comes in. Automated testing allows you to efficiently work incrementally with the confidence that each new sprint hasn’t broken previous sprints.

Agile practitioners often include Test-Driven Development (TDD) as part of their toolkit. TDD, which focuses on writing tests for code before writing the code, can be an incredibly useful way to raise the quality of software.

Automated testing goes hand-in-hand with TDD. While TDD tests the code, automated testing typically makes sure the application functions properly as accessed through the user interface. In addition, you can also automate the testing of backend services, such as databases, to ensure an application is functioning correctly behind the scenes. For example, when a user account is added on a new user screen, automated tests can test and verify both that the screen functioned properly and that the database tables were updated with the information entered.
Automated testing usually focuses on customer acceptance testing. One of the goals of acceptance testing is to ensure the sum of the code parts are actually (at least) equal to the individual pieces. Unlike unit tests developed under a TDD model, which are tightly tied to the code itself, automated tests are one step removed and can be challenging to create, maintain, and extend in an Agile environment.

As your team transitions to Agile, it can seem like the functionality to test is constantly under development, and creating test scripts can feel like driving a car by looking only in the rear view mirror. The trick is creating the right kind of automated tests, in the right areas, to ensure that every sprint doesn’t consist of one step forward and two steps back.
Is Automation Worth the Effort?

In the short term, automated testing can be challenging to implement in an Agile environment.

Some common challenges include:

• Manual testers don’t have the skills to write scripts for test automation

• Software development environments are not equipped to handle automation

• Resistance from within the organization

• Financial costs associated with investment in tools, training, coaching, and hiring knowledgeable staff
For example, it takes 10 minutes on average to run a test, each tester costs $40 per hour, and you have 500 tests to run. Your manual testing cost is $3,333 per test cycle. In addition, it takes 83.3 hours to run all tests using one tester. That’s over two man-weeks! An automated test tool can most likely run that same set of tests overnight across multiple computers, not only saving you money, but also saving you significant time. Plus, you can fully test after every nightly build, not just at the end of a sprint.

So, with these challenges, why not just manually test each sprint? The answer lies in the long-term return on investment, or ROI.

The simple fact is automated test scripts run faster, don’t get tired or bored, and don’t suddenly miss test steps like people do. With a relatively small investment in tools and test scripts, your testers can focus on testing the new and complex parts of the application, while the automation tool keeps retesting the old stuff. Think of the ROI as ‘time to run a test’ times ‘cost of a tester’ times ‘number of tests to run.’
The Test Automation Manifesto

Several years ago, Gerard Meszaros, Shaun Smith, and Jennitta Andrea created the Test Automation Manifesto. (Download it here: http://xunitpatterns.com/~gerard/xpau2003-test-automation-manifesto-paper.pdf.) It’s useful as a starting point when looking at automating tests in an Agile world.

The Test Automation Manifesto states that tests should have the following traits:

- **Concise**: Tests should be as simple as possible and no simpler.
- **Self-checking**: Tests report their own results.
- **Repeatable**: Tests can be run many times in a row without human intervention.
- **Robust**: Tests produce same result now and forever. They are not affected by changes in the external environment.

- **Sufficient**: Tests verify all the requirements of the software being tested.
- **Necessary**: Everything in each test contributes to the specification of desired behavior.
- **Clear**: Every statement is easy to understand.
- **Efficient**: Tests run in a reasonable amount of time.
- **Specific**: Each test failure points to a specific piece of broken functionality.
- **Independent**: Each test can be run by itself or in a suite with an arbitrary set of other tests in any order.
- **Maintainable**: Tests should be easy to understand, modify, and extend.
- **Traceable**: Tests should be traceable to and from the code they test, and to and from the requirements.
Several of these tenets can be quite challenging in the ever-changing Agile world. When examining what to automate (and what to apply these principles to), you might want to start with what not to automate. Edge cases and tests to explore functionality generally aren’t good candidates. Focusing on core functionality maximizes your automated testing ROI.

When deciding what to automate, there are two parts of any development process to look at: continuous integration and regression testing.

**Continuous integration** means bringing your team’s code together as often as possible, at least once per day, to ensure the software as a whole keeps working as changes are made. In its simplest form, continuous integration is used to make sure that all your code still compiles and links. When combined with automated testing, though, the value of continuous integration can dramatically increase.

**Regression testing** uncovers software errors by partially retesting a modified program to ensure that errors were not introduced in the process of fixing other problems. This area of testing often receives the least attention.
Continuous Integration

Most experienced Agile practitioners use continuous integration as part of their team plan. TDD can help here, and your compiler and linker will catch the basic “broken build” problems. However, including a good subset of your automated tests will help you find issues that have been unintentionally affected by the current sprint’s updates.

When deciding which automated tests to include, your focus should be on broad automated tests, such as smoke tests, to ensure the base functionality of the application is intact. You want to look for tests that touch all the key application areas, so the testing you’ll be doing in the sprint isn’t delayed.

**EXAMPLE:** The application we’re building requires users to log in before they can perform key actions. We include automated tests to make sure an administrative user can log in and view each main screen. However, we won’t be adding the security functionality until a later sprint, so we don’t include detailed security tests in the current sprint.

To learn more about automated smoke tests, check out this [Automated Smoke Testing blog post](http://blogs.qawizard.com/automated-smoke-testing/).
Regression testing is where traditional automated testing often comes into play. The challenge is how to integrate regression testing in a continually evolving environment.

Depending on the project, hardening sprints may need to be used as a way to compensate for a slow destabilizing of your application. In each sprint, you drift a bit further from your quality ideal as different parts of your system start diverging from each other.

Use hardening sprints to get everything back in line through refactoring. Fixing issues as they occur—rather than paying this technical debt toward the end of a release—is not only more efficient, but also builds in better quality.

The key is to focus your automated testing on stable or mature functionality. In those areas, going deep with your functional tests helps exercise the most code and minimize those late bugs in unchanged parts of the application.
All good Agile practitioners understand the need for constant review and feedback, and it’s no different with automated testing. At the end of each sprint, you should review three areas.

First, what new functionality should be automated and added to either your continuous integration or regression testing suite? If the new functionality is going to be extended in the next few sprints, add a set of shallow tests (probably in the continuous integration area). When you’re ready, you can extend those scripts into a deeper set that can be used in your regression suite.

Second, which scripts should be rewritten or removed from your automation suite? You might want to rewrite or remove scripts that either test areas of functionality that are about to undergo large changes or consistently fail because of changes from the current sprint.

Finally, review areas that were not automated, but had significant defects associated with them. For example, you had no smoke or regression tests around failed passwords, but ad hoc testing discovered that area of the code was sensitive to change. Adding new automated tests to ensure that future changes don’t break this sensitive area might be in order.
This is one area where an Agile approach can pay real dividends for automation. Because the whole team is involved with sprint planning, team members who focus on automation have a chance to think about what needs to be automated.

In addition, while you’re defining acceptance criteria, you should also be thinking about what you can (and cannot) verify in an automation test.

**DANGER!** If all your tests need to be rewritten after every sprint, you may need to examine your test automation approach. Are you automating the wrong areas of the application? Are your scripts using “fragile” methods, such as screen location, to find controls?

Engaging the Product Owners, developers, and other stakeholders on what can and will be automated also helps them understand how to define good acceptance plans. An acceptance plan of “screen should look clean and well-balanced” is difficult to verify, and even harder to automate.
Automated Testing in a Nutshell

Automated testing helps Agile developers iterate with confidence. Here’s what we learned:

- TDD is a great way to improve software quality.
- Automation brings efficiency to testing and has a high long-term return on investment.
- Automated testing usually focuses on customer acceptance testing, but can also verify data behind the scenes.
- The Test Automation Manifesto is a good starting point for automated testing in an Agile world.
- Continuous integration ensures software keeps working as changes are made to the code.
- Regression testing helps you fix issues as they occur.
- Automated testing also requires constant review and feedback.
Intelligent Tools for Testing

QA Wizard, a business unit of Seapine Software, provides testing tools that help developers and testers work more efficiently and deliver working software to their customers more frequently. We believe that the right set of intelligent tools can improve the productivity of both developers and testers, while boosting product quality and shortening the development cycle to ultimately improve customer satisfaction. Our growing suite of tools includes functional testing, load testing, and stress testing.