Agile Software Testing
Ten Tips for Launching and Testing High Quality Apps in an Agile Environment
White Paper

Agile Software Testing

Ten Tips for Launching and Testing High Quality Apps in an Agile Environment

Table of Contents

• Agile Overview .......................................................... 3
  - The Agile Manifesto .............................................. 3
  - The Problem With Waterfall ................................ 3
  - The Emergence of Agile ...................................... 4
  - It's Okay To Use Waterfall ................................. 4

• Ten Tips of Agile Testing ........................................... 5
  1. Understand the True Role of Testing ................. 5
  2. Unify SRS and Test Plans ................................. 5
  3. Define Your Testing Matrix ............................. 6
  4. Tell a Story – Not a Use Case ......................... 7
  5. Capture Meaningful Data ............................... 8
  6. Fix Broken Windows ....................................... 9
  7. Make Testing Your Feedback Loop ................. 9
  8. Timing is Everything .................................... 10
  9. Run Frequent Load Tests .............................. 10
  10. Stick to Your Scrums ................................. 11

• About uTest ................................................................. 12

“Simplicity is the ultimate sophistication.”

- Leonardo da Vinci
Introduction

Agile Overview

A Google search for “agile development” returned 2.7 million results at the time this was written. It’s safe to say the word is getting out. But although the basic concepts have been actively discussed in books, blogs and everything in between, we’re going to first review them anyway. We promise to be quick. If you’ve heard this story before, feel free to skip ahead to the next section: Tips for Agile Testing.

The Agile Manifesto

Though ‘Agile’ is a relatively new term, the shift towards more iterative development methodologies began years ago. Eventually, in 2001, a small group of CTOs, academics and thought leaders published the well-known Agile Manifesto. Here is a summary of the document’s fundamental principles:

*Figure 1: The Agile Manifesto*

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

*That is, while there is value in the items on the right, we value the items on the left more.*

There have been multiple “spin-offs” of Agile since this document was first published, including Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, and Pragmatic Programming, among others. Though they differ in subtle ways, they hold the same foundational values described above.

The Problem with Waterfall (it’s not agile!)

Waterfall development is characterized by well-defined phases, each having a thorough review and authorization process that should be completed before proceeding. On the surface, this process is not inherently bad. In fact, it’s quite logical: First, figure out everything you need. Then design it. Then code it. Then test it. And repeat.
Problems arise when the project requires adjustments and modifications that were not anticipated in the early stages. Reality ruins the plan. So for projects that can be spec’d out with precision ahead of time, waterfall may be the proper choice. Unfortunately, experience has shown that a calm, unchanging set of requirements is a rarity.

Whose fault is that? The answer varies depending on who you ask. Upper management often blames the product team for not being thorough enough. Product blames sloppy coding by development. Testing blames management for not enforcing tighter deadlines and cutoffs. “You signed off on the spec two months ago; we can’t add this feature now!”

Looking back, it’s no wonder why these projects became the source of such frustration; basically, we delivered software that lacked the most important features! At the same time, we were up to our necks in fancy-looking documents – MRD, SRS, Architecture diagrams, APIs and Test Plans – that didn’t provide any real value to customers.

The Emergence of Agile
Out of necessity, development teams began shifting to frequent releases. At first, these iterative methods went with small cycles, but remained with a fixed schedule ahead of time (i.e. fixed budget, strict plan, and predetermined feature sets). So instead of the one major release, several minor releases would be scheduled.

This concept would be taken a step further: Iterative releases, with no fixed plan! Here, each release adds a bit of value, allowing for better decisions as to what features would be included next. Development now works in small cycles – planning and designing only far enough ahead to keep the flow working. With this increased flexibility, the burden of excessive formal documentation was greatly alleviated. Now testing is incorporated into each step of the process, not just at the end of each release.

“It’s Okay to Use Waterfall – It Isn’t One vs. the Other
Although Agile is gaining in popularity, this piece is not an endorsement of one over the other. Many of the tips we’ll discuss in this whitepaper are based on Agile concepts, but are relevant to both types of development processes. So, if your organization uses traditional waterfall methodologies, you can still benefit from Agile concepts to improve quality. This condition has been coined as “Agile-fall.”

“There’s no shame in that if that’s what works,” said testing expert John Bach. “When you’re going through a transition from Waterfall to Agile, that may be the best thing as opposed to a sudden lever-pull one day where you show up and your desk is next to someone else with no walls and there’s a stack of sticky notes and markers on your chair with an email to report to your first standup in 30-minutes.”

"Agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans.”

- Barry Boehm
Software Engineering Guru
If your company is developing quality software; if the development is sustainable and the business is being adequately served by that software, there’s really no need to press the issue. Agile is no panacea for problems related to development, process, and management. However, if you’re willing to make some of the necessary changes, it can be extremely beneficial in the long run.

**Agile Testing Tips**

If you’re operating in an Agile development environment, or just want to adopt more real-time, real-world testing processes, here are some tips that should enable you to improve efficiency and quality:

1. **Understand the True Role of Testing**
   People often think the goal of testing should be to simply find all the bugs. Well, that’s not entirely realistic. Actually, it’s impossible! Despite the title of “quality assurance” testers can never truly assure quality, nor can they be expected to find every bug. The real goal of testing should be to improve the software. This is a widely held view of testing in the Agile world.

   Despite this, upper management still tends to believe that testing can uncover all the bugs or prove whether or not the software will function as intended. They wrongly equate testing with validation. To understand the true role of testing, one needs to understand that it is a scientific method – a continuous search for information. So instead of thinking of tests in terms of ‘pass’ or ‘fail’, try to think of whether or not they provide valuable information. In an Agile environment, testers should be called upon to provide this valuable information through the development cycle, as opposed to issuing a ‘pass’ or ‘fail grade at the end.

   If Agile is to succeed, testing must become a central pillar of the development process. It can seem chaotic at first, but when executed properly, Agile is not chaotic at all. Any methodology that has programmers build unit tests before writing code is orderly by definition! It’s just a different kind of order – one that focuses on code rather than paper, and on value as opposed to blame.

   “Testing is the infinite process of comparing the invisible to the ambiguous in order to avoid the unthinkable happening to the anonymous.”

   - James Bach
   Testing Expert

2. **Unify SRS and Test Plans**
   This tip requires that you define your testing strategy early on. This does not have to be a burden. Understand that an SRS (software requirement specification) is quite similar to a test plan. From an optimization perspective, it eliminates the wasted process sometimes referred to as “Test Plan Alchemy”, in which separate test plans
are created by merely copy-pasting from the SRS, and then search-and-replacing “The software should do …” with “Make sure that the software does…”

It’s important to note that optimization improves quality. Testers know how to find errors. They also know that errors don’t just appear at the end, they appear throughout the process. Finding errors in logic or in usability is a must for Product Managers during the spec phases, yet their skill sets are often geared towards other areas. Looking at the unified SRS/Test Plan with a tester’s eye, therefore, adds tremendous value in the early stages.

3. Carefully Define Your Testing Coverage Matrix

The multi-dimensional matrix of what needs to be tested should be defined early. The importance of this has grown exponentially the past few years, as the matrix has become increasingly complex:

*Figure 3 – Growth in Complexity of the Compliance Matrix*

Defining the coverage requirements is an essential part of the specifications process, and thus can be dealt with at an early stage. The size of the coverage
matrix has a significant impact on staffing needs and QA costs. Because of this fact, defining it clearly and early helps management allocate resources and determine what can be done in-house versus what should be outsourced for greater coverage. This applies to:

- Operating systems
- Browsers
- Plug-ins, anti-virus programs and firewalls
- Geographic locations
- Languages
- And in the case of mobile applications, handset makers and wireless carriers

Unless your software is designed for a simple and homogenous audience that’s identical to your in-house QA team, it’s likely that you have gaps in your testing coverage. Yet to fill these voids - whether they be browser, OS, location, language or device – would be impractical, expensive and a huge waste of time. With the emergence of crowdsourcing, this is no longer the case. Having already assembled a global testing community, agile teams of all sizes are now able to target very specific users for specific assignments.

4. Tell a Story – Not a Use Case
Forget the words “use case” and instead say “storytelling.” In theory, it’s the same thing, but in reality it’s much different. To illustrate this difference, try searching for each of these terms on Google image. “Use case” brings images of dry and intimidating flowcharts and diagrams, while “story” brings friendly and colorful pictures.

Testers will have a similar subconscious reaction: Ask a tester to build a use case and they will think in dry and uninspired terms. But ask them for a story, and you will see the creative juices start to flow. This way, you encourage them to be more aggressive and not settle for simple screen touching. It empowers the testers, without actually using the cliché word “empowering.”

In an Agile environment – where testing is being done continuously – it is especially important that you keep things fresh. Testing expert James Whittaker has been an advocate of “test tours.” Here’s an example from his latest book, *Exploratory Testing*: 

---

*Exploratory Testing*
The Supermodel Tour:
For this tour, I want you to think superficially. Whatever you do, don’t go beyond skin deep. This tour is not about function or substance; it’s about looks and first impressions. Think of this as the cool tour that all the beautiful people take; not because the tour is meaningful or a great learning experience. On the contrary, you take this tour just to be seen.

Get the idea? During the Supermodel tour, the focus is not on functionality or real interaction. It’s only on the interface. Take the tour and watch the interface elements. Do they look good? Do they render properly, and is the performance good? As you make changes, does the GUI refresh properly?

Although he is essentially discussing test cases, the same idea applies to use cases: encourage creativity whenever possible.

5. Capture Meaningful Data
Testing managers often focus on information that is of little use to anyone but themselves - keep this in mind when collecting data. We’re all used to compiling detailed reports that show the number of test cases written (i.e. how many have been run, failure rate, severity levels, etc.) and this is certainly valid data to the testing manager, but who else in your organization will care? Would the business development team, for instance, have any practical use for this?

One type of data that would be of value is direct business impact statistics: Sales missed due to bugs, customer renewal rate and relative cost of defect (see chart below). This type of information is like gold to decision-makers. If you can collect this data, then do so immediately.

![Figure 4 – Relative cost of defect, by time of discovery](image)

When collecting data, however, be sure not to go overboard. As testing expert Michael Bolton said in a recent interview:
“When you enslave numbers, they eventually rise up, revolt, and enslave you. These organizations spend so much time collecting the data and talking about it and justifying it and trying to duck blame that they don’t seem to have time to do anything about the actual problems, which generally fall into two categories. One: the organizations are trying to do more work than they can handle with the approaches they’re using. Two: they’re not listening to people that matter—neither to their customers, nor to their own front-line staff, many of whom are closest to the customers.

When your car is about to go off a cliff, it’s a weird time to be thinking about gas mileage and drag coefficients; better to take the right control action—look out the window and steer or use the brake until you’re back on course.”

Some statistics, specifically the containment rate, require a certain degree of self-measurement. Using a technique such as Evidence-Based Scheduling, therefore, can save you a great deal of time, effort and money. You can find an excellent overview of this approach here.

6. Fix Broken Windows
“Fixing broken windows’ is a perfect metaphor for this issue. In the 1980’s, an experiment was run to measure the impact that broken windows have on crime in urban neighborhoods. Now, we wouldn’t normally think of broken windows as a cause of crime. But indeed, it was shown that just fixing broken windows in a neighborhood helped decrease the crime rate. The reason for this is that if you give the appearance that you don’t care, then others won’t care either.

“Testing is NOT a phase on Agile teams, testing is a way of life. Agile teams test continuously. It’s the only way to ensure that the features implemented during a given iteration or sprint are actually done.”

- Elisabeth Hendrickson
Quality Tree Software, Inc.

It’s the same with your testing efforts. If you continually say “Ignore this bug, it’s a known issue” or “Ignore that test, it’s not yet relevant”, your testing team will begin to listen to you: They’ll be more willing to ignore bugs, and they’ll miss the ones that you don’t want ignored.

7. Make Testing Your Feedback Loop
The testing process not only eliminates bugs, it can also serve as source of customer feedback. If you build a community of testers surrounding your product that truly represent your user base (in terms of demographics, geography, level of knowledge, etc.), you can add feedback into the system before you even launch your products. This is especially important for public web and mobile applications, as opposed to turnkey custom projects.
Feedback generated by a community of testers - whether through beta testing or crowdsourcing - can affirm (or refute) any preconceptions you might have regarding your software. You will get to see your product used under real-world conditions, something that often does not occur in many QA labs. Better yet, this information can be shared with other key decision makers in your organization, notably the sales and marketing departments, who’ll often spend great amounts of money on surveys and research.

“My job as a tester includes understanding and advocating for a great customer experience,” said noted testing blogger Lanette Creamer. “This means feedback beyond ‘does this meet requirements’ and evaluating ‘does this meet the customer needs overall based on all that I know and continue to learn about the customer?’”

8. Timing is Everything
The short release cycles of Agile development can often place QA teams under tight time constraints. Naturally, there are gaps in your staff’s schedules. The two day weekend is almost always a period of low development and testing activity, and this is a good thing – people need their rest to maintain productivity. However, this two day gap need not go to waste.

By adding a crowdsourced community to your testing process, you can deliver software releases for testing each Friday and receive the results by Monday morning, thus compressing the development cycle without adding pressure to in-house staff. It’s one thing to be able to get a release out on schedule, but if you and your staff are working 70 hours a week to get it done, what’s the point? With crowdsourcing, there’s finally an easy, cost-efficient way to augment your in-house QA team.

9. Run Frequent Load Tests
As testing becomes integrated with development – and as testing activities become more and more compressed – it’s easy to forget about the routine responsibilities. Namely, load testing.

At what point will your application’s performance begin to degrade? How many concurrent users can it support? How do the JavaScript-based components of your application behave with 50, 100 or 1000 active users? Where are the bottlenecks between your code base, database, CDN and load balancers? Without answers to these questions, your testing data (and efforts) could be all for naught. If your software can’t hold up under stress, it doesn’t matter how agile your methods are.

With recent innovations in crowdsourced testing and open-source tools – innovations that have greatly reduced cost and time commitment – Agile companies no longer have an excuse for failing to run frequent load tests. Load tests can now be run on extremely short notice, while targeting specific functionality as opposed to the entire application. They can be run with real users or the latest in automated technology. For more on affordable load testing, read Load Testing For Less.
10. Stick To Your Scrums

Far from an Agile testing prerequisite, daily Scrum meetings have proven to be extremely beneficial for teams in the long run, as they help departments to remain focused, energized and coordinated. Writes the Scrum Alliance:

In most companies, development is slowed down by issues identified as impediments during the daily meetings or planning and review meetings. With Scrum, these impediments are prioritized and systematically removed, further increasing productivity and quality. Well-run Scrums achieve the Toyota effect: four times industry average productivity and twelve times better quality.

Scrum removes management pressure from teams. Teams are allowed to select their own work, and then self-organize through close communication and mutual agreement within the team on how best to accomplish the work. In a successful Scrum, this autonomy can significantly improve the quality of life for developers and enhance employee retention for managers.

Too often, however, daily Scrum meetings become weekly meetings, which in turn become monthly meetings. Pretty soon, you no longer recognize Bill from development. So set a time that works for everyone, re-introduce yourself to Bill, and do your best to stick to the schedule. Good luck!

For more on testing in an Agile environment, check out this [uTest webinar](#).
About uTest

uTest provides in-the-wild testing services that span the entire software development lifecycle – including functional, security, load, localization and usability testing. The company’s community of 60,000+ professional testers from 190 countries put web, mobile and desktop applications through their paces by testing on real devices under real-world conditions.

Thousands of companies -- from startups to industry-leading brands – rely on uTest as a critical component of their testing processes for fast, reliable, and cost-effective testing results.

More info is available at www.utest.com or blog.utest.com, or you can watch a brief online demo at www.utest.com/demo.

uTest, Inc.
153 Cordaville Road
Southborough, MA 01772
p: 1.800.445.3914
e: info@utest.com
w: www.utest.com