



Agile Test Plan

How to Construct an Agile Test Plan



Agile is changing not only the way we develop software but the way we work and do business.

Rather than a detailed plan that explains step by step what to do and who should do it, agile sets forth a direction with a compass and ways to work with the compass (users and product owners) so you get to where you need to go.

In traditional software development project life cycles, test plans play a very important role. In the initial phase of the project, the testing team sits together and puts together a test strategy while discussing the testing scope based on the requirements specification to ensure that all critical features mentioned in the specification will be tested. They then discuss who tests what and the timing of each test phase alongside development. The output of this process is the test plan.

A test plan document systematically describes the testing approach, what to do and by who. The team needs the test plan because they need to understand test coverage, test method and test responsibility. Who tests what and how are

results reported? This is not only about the role assignment but may also include additional responsibilities such as collecting and maintaining data, setting up test environments and use of tools. There are many references and documents that tell you the components of a software test plan. There is even an ISO standard for a software test plan. However, in agile, and in particular scrum, a formal test plan document is not always necessary.

When transferring from a traditional development workflow to agile, most teams will encounter numerous problems. One of the most common problems occurs when testing teams implement traditional test workflows in an agile project.



Agile isn't for everyone.

If it's not working, be agile and adapt to something that works for you.

Teams converting to agile have challenges not only in adapting processes and thinking differently, but also behaving differently.

After several sprints test coverage is not adequate and consequently the team often finds themselves struggling to catch up with the entire testing process. The problem is that with 'agile', people tend to interpret the principles of agile literally. Let's examine these in light of applying them to an agile test plan:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan
5. That is, while there is value in the items on the right, we value the items on the left more.

Now a step by step analysis:

1. Individuals and interactions are certainly important. We want people to talk to each other. We also want to recognize each person's skills and capabilities. I don't see anywhere in here that everyone should do everything, nor does it say processes and tools are not important. Just less valued than individuals. If an individual quits, and we have no processes and no documentation, then maybe we may want to rethink how we implement agile.
2. Working software could mean we want to fix defects quickly so the software works, but what does that mean? 1 hour, 1 day?

Most important, you need to be agile about being agile. Adapt to bumps in the road and the context of your project and organization.

That's up to us. Note that working software is not equal to 'perfect' software, and that working software leads to more of 3. Customer collaboration. So, the point is, get something working as soon as you can to the customer so you can interact and get feedback. Does it have to be perfect, or even working completely? Maybe, maybe not.

3. Customer collaboration is perhaps the biggest difference compared to waterfall. Working with the client on a frequent basis using working software, combined with 4. Responding to change, is critical if we want to eventually build something they want. Is

contract negotiation not important? NO. We still need a contract, but perhaps with changed terms which provide more flexibility yet require more trust on both sides.

4. Responding to change is key as well. If we are following 3, we'll discover a lot of changes because the customer will change their requirements. This will be facilitated by 'working software' so they can see what they like and don't like, often.

Traditional Waterfall Workflow

Let's examine a conventional waterfall workflow as shown in Figure 1.

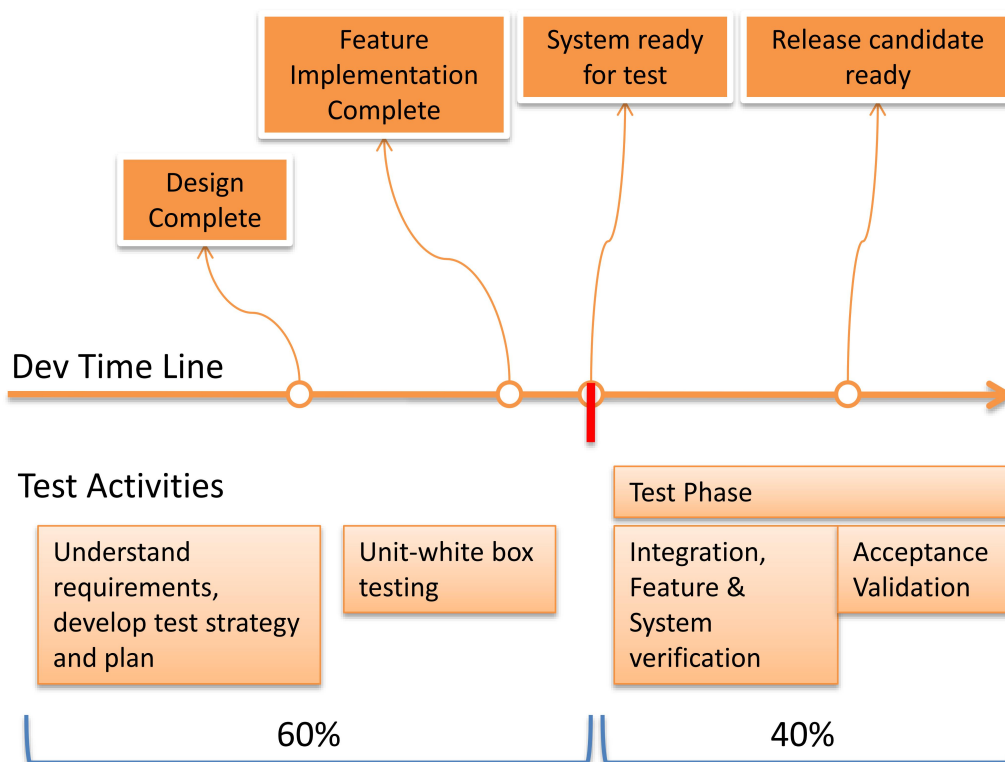
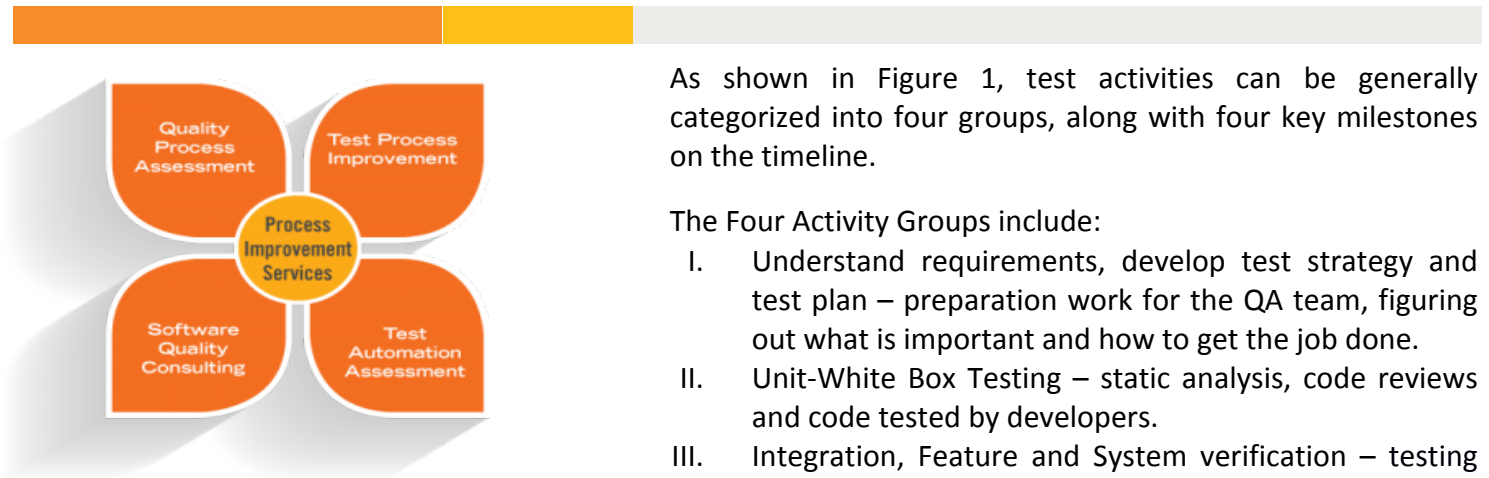


Figure 1. Waterfall Workflow Timeline – Test Activities

In agile, some waterfall activities are reduced while some get larger, expand, change, and move to the left.



Agile doesn't mean
no documentation.
It means
documentation for a
purpose.

As shown in Figure 1, test activities can be generally categorized into four groups, along with four key milestones on the timeline.

The Four Activity Groups include:

- I. Understand requirements, develop test strategy and test plan – preparation work for the QA team, figuring out what is important and how to get the job done.
- II. Unit-White Box Testing – static analysis, code reviews and code tested by developers.
- III. Integration, Feature and System verification – testing all features, and interactions within and outside the software.
- IV. Acceptance validation – end user scenarios and acceptance criteria.

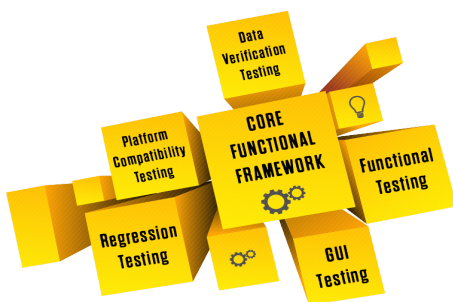
Four key milestones include:

- I. Design complete – architects and designers are done.
- II. Features implementation complete – programmers are finished.
- III. System ready for test – testers begin actual testing.
- IV. Release candidate is ready - most critical bugs are fixed.

Using a traditional testing workflow, verification starts at Milestone III (System Ready for Test). Even if unit tests are excluded, the test execution time is usually about 40% of the entire development process. Because the timeline is much longer, let's say a 6-month project, there may be 10 weeks dedicated to testing at the end of the project (40%).

Agile Workflow

But with agile, the time line is shortened to the length of the iteration (depending on the particular agile implementation, some companies use 1 week, while others use 2 months) and milestones change as shown in Figure 2.



In agile, step by step processes can get blurry. Be flexible. Go where you are wanted and needed, all with the goal of quality.

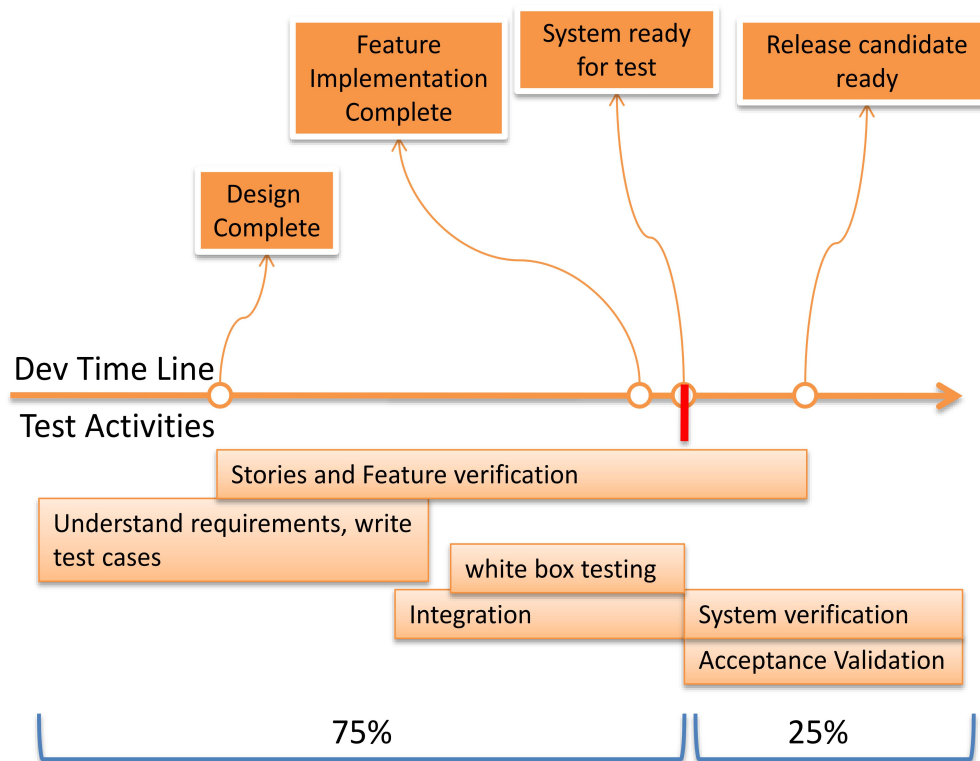


Figure 2. Agile Workflow

One of the agile principles is that shorter is better. The reason for this is to have small pieces of working software to show the customer, get feedback, and change requirements if necessary earlier rather than later. But because of this, the system candidate is ready relatively late at about 75% (of a short iteration) as developers are supposedly doing more checking of their own code and due to using many CI tools that integrate and make sure the code is working as it is checked in. Therefore generating the feeling that we can use up more time in the cycle for development. Because of this, software testers must be involved a lot earlier in the process. Even if they write test cases and conduct small tests (involving an incomplete user story) before feature implementation ends, the actual test

execution usually starts about 75% of the sprint time spent (rather than 60% as in a traditional workflow). Consequently there is insufficient time to execute complete regression, even if there are very well designed test cases.

Therefore in an agile test plan, feature verifications should start as early as possible, immediately from the end of design. About 60% of sprint time is spent from the end of Milestone I to Milestone II. During this period the stories/features verification and integration tests should be completed. According to our project experience, after Milestone III, only 20% to 25% of sprint time can be allocated to system regression tests and acceptance tests for release candidates.

Rather than thinking “oh, they’ve changed their minds again”, “Being” agile means welcoming changing requirements.



Agile Test Plan Components and Considerations

In contrast to a traditional waterfall test plan, an agile test plan is more time box oriented and related to the timing of the development iteration. Basically, in ‘agile’ fashion, we do what is needed within a framework and divide our plan into these areas of consideration:

1. Requirements and design phase:

This phase is similar to the traditional test workflow. Major tasks consist of understanding the features and user stories but for only that iteration, defining the test strategy, defining the scope of tests and broadly estimating how much time should be spent. But there are some differences:

- I. Defining test drivers or test interfaces becomes more important in this phase. This is because in most cases, UI is not ready and the tests have to be executed through these interfaces.
- II. Since additional time is spent on test driver design, test case design will require less time. It is common to find that test cases in agile are more of a loose guide.

This phase will spend 10% - 15% of sprint time.

When requirements change, Agile workflows gives you the ability to adapt to these changes – so you can't have a huge rigid test plan. Agile testing needs to be as agile as the requirements themselves. Testing needs to follow features implemented in that iteration regardless of the plan.



2. Stories/Features verification phase:

In this phase, test execution should start for stories or features even if not completely functional. The tests activities in this phase may include:

- I. Investigation on the features
- II. Static review of the code
- III. Gray box tests using drivers or interfaces to test the features
- IV. Black box test of features with the existing UI
- V. Cross-story tests among features – as nightly builds are done, features and stories will begin to work together so scenarios which involve more than one, sometimes 2 and 3 user stories need to be tested.
- VI. Integration tests

The main purpose of this phase is to make sure all the new features work well within themselves and with other components. Some minor verification can be



Be flexible, cooperative and collaborative. Those are big words that sound good, but old habits are sometimes hard to break. You may not even be aware of your own ‘non-agile’ behaviors and habits. Be open.

postponed to Milestones 3 or 4, but the level of confidence in the new features should be built within this phase.

A lot of defects should be found during this period. Some will be fixed immediately and need to be validated in the next daily build. Others will be postponed into next milestone or next release.

This phase will take about 50% - 60% of sprint time.

When implementing Agile Testing, keep your feedback loop short. If you find a defect, talk about it immediately with developers. If you need clarification regarding requirements, talk to the Product Owner. Then adapt your test plan and testing accordingly right away.

3. System verification phase:

When most features are implemented, the system will be integrated for testing. As mentioned above, you may only have 20% left before you have to deliver ‘working’ software at this point. Additionally you need to reserve some time for phase 4 when a release candidate is ready to perform smoke tests.

The main purpose of this phase is to ensure the new feature set and all the latest changes do not destroy or disrupt the system in any way. This also includes ensuring that the previous set of features function as they should with the system.

In addition to this, you need to plan time for defect validation. This means there will be no spare resources to conduct any more feature tests. All the rest of the resources and time available should be placed into regression testing.

“Just because in Agile, everyone takes ownership at driving towards a high quality product doesn't mean everyone does everything! Everyone is tasked at doing what they're best at, but they're always cognizant of what needs to be done outside their own role.”

4. Acceptance phase:

Once regression (usually targeted and not complete) is finished, the product is nearly ready to be released. Most bugs are fixed during the regression tests, the system becomes stable and a release candidate is built. Running a smoke test before the release is necessary to ensure all the changes made will not break the system and confirm that all critical bugs are fixed.

If possible, some resources can be allocated to exploratory testing for the newly developed features within this sprint.

5. Feedback, retrospective and change:

Some teams just breeze through the retrospective and aren't honest about what they think or feel. Its not supposed to be a bashing session but if you don't do it 100% in, then you're losing one of the key aspects of agile. With respect to the test plan, at each retrospective, the following iteration's plan should be adapted in terms of resource and time allocation. For instance, if developers are saying that unit testing is falling behind, then QA may need to work directly side by side and match requirements to unit tests. Or, if a certain feature, function or user story has more problems than it should, QA needs to shift focus with the goal of ‘working software’.

For agile, many think velocity is the goal. Think quality first, and you'll get velocity. Think velocity first, and you'll get velocity.

6. Technical Debt:

Many don't plan for technical debt. The fact is, if you drive faster, you open yourself to more risk and crashing. Agile, with a focus on velocity, can also crash if you don't plan for quality. Addressing technical debt should be part of the plan. What often works best is periodically dedicating an iteration to removing debt. This comes in the form of documenting code, correcting requirements that were coded differently, but the documentation is lagging, fixing automation scripts, refactoring, optimizing database tables, etc...

7. Sliding and staggering:

Shifting regression to $\frac{1}{2}$ a cycle later can also relieve the regression burden without much loss in velocity. Most teams operate a hybrid form of agile while still holding to agile principles centered around velocity, quality and working software delivered incrementally so that rapid feedback can be given and change direction-add-subtract requirements and features if necessary. As mentioned, the rapid pace of iterations and continuous integration often leads to limited time for regression towards the end of the iteration. To handle this, many teams stagger the regression testing and just deliver one week late. You are still getting working software at the pre-determined interval, just not right after it is coded, but $\frac{1}{2}$ an iteration later. Since iterations are so short, it's certainly nothing to be ashamed of when compared with a waterfall process whereby working software would wait until the very end.

8. Time Estimate:


When everyone sits down and does the plan together, you estimate the amount of work needed to do each task and sub-task and assign tasks to individuals. This is a critical component of agile. On one hand, you don't want a plan, on the other hand, here you have a task and a timeline, and a person assigned! The difference is that this is an iteration plan rather than a project plan. These estimates are critical to the success of the project and the iteration. Its important for people estimate real hours and log real hours rather than pipe dreams.

From a QA point of view, testers need to speak up and include testing effort into the plan. Too often its overlooked and just added on with a rough percentage or unit allocation without real thought into the complexity of the features that are being tested.

Remember that one of the principles of agile is sustainability. If you are consistently underestimating the work and then working overtime, this is not sustainable. Thus have measurements for estimation and schedule adherence accuracy.

Conclusion

As we've discussed, traditional/waterfall projects usually dictate the need for a detailed test plan document because they want to set up a very clear view of the testing scope, method and responsibilities at the beginning of a project. In an Agile workflow the process is incremental with test planning scheduled with each development iteration. With each iteration (usually called Sprint in Scrum), planning helps



teams decide the test scope and methodology while adapting to the current development situation. Some iterations may be dedicated to fixing defects and doing rework – refactoring, just because there is a need. On the other hand, there may be a change in a major feature or requirement, so everyone must focus on that and get it working correctly, so perhaps there is little time for any regression and the development ends 90% into the cycle. So, there is no sense in planning several months of testing when all you need to do is test 2 weeks at a time. On the other hand, we need to be agile about agile. Just because we start out with 2 week iterations, or 1 week iterations, doesn't mean we have to stick to that or we're not 'agile'. Agile means 'short' iterations and there is no hard rule.

Therefore, the necessity to create formal test plan documents at the beginning of the project is minimized, and the testing adapts to the given situation for the current iteration, with growing importance in that quality is the responsibility of everyone in the team. Developers pay attention to quality via writing good code with unit tests, while

product managers focus on writing clear user stories that people can understand, and QA ties it all together by testing each user story and developing cross stories for larger systems. With everyone focused on quality, that doesn't mean that everyone does everything. You don't see any sports teams where everyone does everything? They each have specialties that they are experts at, and that benefits the whole team.

Also, remember, plans need to be documented to the extent that its useful. Some think that agile means zero documentation but you still need a record of assets and progress. Otherwise, when you do maintenance or have changes in team players, the knowledge walks out the door.

Lastly, remember that quality is the goal, not speed. If you go for quality, you'll get speed by reducing tons of rework. But it doesn't work the other way around. That being said, you need to remove all obstacles to quality.



XBOSoft

Software Quality Improvement

XBOSoft Inc.

3333 Bowers Ave. #130
Santa Clara, CA 95054
www.xbosoft.com
services@xbosoft.com

+1 408-980-7120