ALM Best Practices Series

For ALM Practitioners

Agile Best Practices

Document release date: July 2020

Legal Notices

Disclaimer

Certain versions of software and/or documents (“Material”) accessible here may contain branding from Hewlett-Packard Company (now HP Inc.) and Hewlett Packard Enterprise Company. As of September 1, 2017, the Material is now offered by Micro Focus, a separately owned and operated company. Any reference to the HP and Hewlett Packard Enterprise/HPE marks is historical in nature, and the HP and Hewlett Packard Enterprise/HPE marks are the property of their respective owners.

Warranty

The only warranties for products and services of Micro Focus and its affiliates and licensors (“Micro Focus”) are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted Rights Legend

Contains Confidential Information. Except as specifically indicated otherwise, a valid license is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Contents

Welcome To This Guide

Welcome to the Agile Testing Best Practices guide.

This guide provides concepts, guidelines, and practical examples for the best implementation of agile testing principles in various organizations.

About Agile Testing

Have you ever come across a piece of software that just does not work and wonder, 'How did this code get released?' Or maybe you have actually worked on a project where you told the powers the software should not be released, but to your bewilderment, they went and did it anyway.

There are, of course, a myriad of reasons and circumstances that can lead to poor code being released. One major cause, however, can be the company culture.

If developers are highly valued and testers are seen to be of less value, then it is easy to see why the voice of the tester can go unheard and their view ignored. If testers are not sufficiently technical or are unable to write a coherent and convincing report, their message can get shot down in the general debate, or considered of little value due to poor presentation. If the release decision is taking place behind closed doors with no opportunity for the test report to be presented and discussed, then the information regarding the true state of the software might never be made available to those making the decisions.

There may be a number of reasons why, even when the test report is fully understood, the release is still sanctioned and bad code is released. However these tend to be few - the main reason is that the test department is excluded from the process.

Agile methodologies are the latest attempt by development teams to bridge the gulf between increasing business expectations and unsatisfactory IT delivery. While it is estimated that more than 70 percent of projects still follow the waterfall (classic) model, agile is quickly becoming mainstream in small and large organizations alike. With its promise to honor results over processes, agile methodologies are the focus of the efforts to deliver better software.

Micro Focus has adopted agile practices for its internal development processes and believes that sharing its experiences can help customers in adjusting their methodology and processes to deliver quality software.

These guidelines should help reduce initial creation time and achieve maximum value while operating ALM.

Audience

This guide is intended for:

- Quality Assurance Managers
- Testing Automation Engineers
- Development Managers

Prerequisites

To use this book, you should be well acquainted with the major phases of Software Development Life Cycle (SDLC). You should also be familiar with the business processes in actual IT organizations.

Structure

This guide is organized as follows:

- Introduction to Agile Testing
- Preparing for Agile Testing
- **Error! Reference source not found.**

- Conclusions

Feedback

If you have questions, comments, or valuable best practice information you want to share, send a message to the following email address:

docteam@microfocus.com

1 Introduction to Agile Testing

Importance of Agile

The need for software products of high quality has pressured those in the software profession to identify and quantify quality factors such as usability, testability, maintainability, and reliability, as well as identify engineering practices that support the production of quality products having these favorable attributes. The software development process, like most engineering artifacts, must be engineered. That is, it must be designed, implemented, evaluated, and maintained. As in other engineering disciplines, a software development process must evolve in a consistent and predictable manner, and the best technical and managerial practices must be integrated in a systematic way.

Businesses that are under pressure to deliver higher quality applications in response to competitive demands are turning to agile development practices. In fact, agile and other iterative methods are becoming the de facto standard for application development. Ideally, agile is about delivering the highest business value possible more quickly by focusing on people and continuous improvement. The agile methodology is usually perceived as mainly applicable to development teams, but actually requires the entire organization to adjust.

Agile development presents organizations with two important challenges: bringing quality and stability to applications much earlier in the development process in order to align with the business, and becoming flexible enough to keep pace with the iterative nature of the agile methodology.

Micro Focus offers an approach and solutions that help teams embrace the agile methodology and make a significant contribution to its successful outcome.

The Real Agile

What Agile Is

The term agile is definitely suffering from buzzword syndrome. "Agile" sounds so good – it is nimble, flexible, and strong. Who wouldn't want to be "agile?" No one says, "Thanks, I'd rather be inflexible and slow to respond." Additionally, because it's a standard word in English, everyone thinks they know what "agile" means. The problem with buzzwords is that people invoke them without understanding what the terms really mean.

A Google search for "agile development" returns millions of responses and it is no wonder that many people interpret it differently. The main portion of Agile Manifesto is well known:

> *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.*

However, its true meaning is sometimes lost in the sea of multiple "spin-offs" of Agile, including Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, and Pragmatic Programming, among others. Even though they differ in subtle ways, these methodologies hold the same foundational values described above. "Agile" is a relentless focus on providing a continuous stream of business value, even in the face of constant change.

So what is agile and why do old methods not always work as efficiently?

Classic Is Not Forever

In our fast moving world, classic approaches do not always work. 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. Repeat.

Problems arise when the project requires adjustments and modifications that were not anticipated in the early stages. Reality ruins the plan. For projects that can be specified 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 it that? The answer varies depending on whom 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!"

It is no surprise that these projects become the source of such frustration - in the end, teams often deliver software that lacks the most important features! At the same time, there are piles of great looking documents – MRD, SRS, Architecture diagrams, APIs and Test Plans – that don't provide any real value to customers.

Agile is Born

Because of these failures, 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 was taken step further - iterative releases with no fixed plan. Essentially, each release would add a bit of value, allowing for better decisions as to what features should be included next. Development would work 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 is greatly alleviated. Now testing is incorporated into each step of the process, not just at the end of each release.

Waterfall and Agile Live in Peace

While Agile is gaining in popularity, there is still a place for the classic approach– after all, the success of any development methodology is measured by the quality of software outcomes. But even if your organization uses traditional waterfall methodologies, you can still benefit from agile concepts to improve quality.

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

Recently there has been a proliferation of *hybrid* approaches where some teams working on a new project or switching to a new technology adopt agile methodology in full, while some take pieces that suit their business goals and culture, and other teams, especially in big corporations, still adhere to the classic methodology simply because it works.

In larger organizations, we often see different teams working on different components of a larger initiative while using different methodologies to produce the software. This makes reporting on initiative status difficult as each team's goals and KPIs vary across the different processes. Being able to manage different flavors of agile and waterfall within the same project adds complexity to an already complex operation. ALM helps overcome the complexity by using standardized, template-based projects – agile and waterfall are both offered.

As with many things in the technology industry, there probably won't be a clear winner in the methodology battle and hybrid approaches will eventually prevail once the buzz begins to subside.

2 Preparing for Agile Testing

From Waterfall to Agile

The waterfall approach in software development is a sequential process in which development is seen as flowing steadily downwards (like a waterfall) through the phases of requirements analysis, design, implementation, testing, integration, and maintenance.

It treats testing as yet another phase in product life. Here are some of its qualities:

- First you code, then you test.
- Development and testing teams are separate, each working in its own silo.
- Each part of the process is heavy and requires tons of documents:
  - Requirements
  - Design documents
  - Test plans
  - Traceability matrices
  - Change review boards
- Testing becomes a "bottleneck" in the very end, thus making it virtually impossible to maintain a schedule.
- Automation tools get into the action after coding – this requires expensive efforts by testing teams to record/playback.

The waterfall approach suffers from a lack of involvement, broken communication between developers and testers, and minimal impact on product decision makers.

The introduction of agile methodology promises to alleviate some of the important problems of the waterfall methodology:

- Coding and testing become inseparable.
- Feedback and collaboration are the key ingredients of the cure.
- Exploratory testing reveals unintended consequences and uncovers implicit expectations.
- In most of the cases, the tester is expected to show critical thinking, not only adherence to requirement documents.
- As jobs definitions are blurred, diverse ideas begin flowing.
- All types of people start working as a single team.

Recipes for Successful Agile

Micro Focus believes in a holistic view of application lifecycle management. When agile is added into the mix, it enables integrating QA and user feedback into development processes. Development and QA become part of the same team – as opposed to the uneasy truce that exists between those two roles today. It also allows greater adaptation to customer needs and market changes, ensuring that the products meet the high standards customers expect.

Let's see what recipes Micro Focus has for making a testing process an agile. These recommendations should help build successful agile teams that deliver better software.

Assess Your Processes

Many people think the goal of testing should be to simply find *all* the bugs. In fact, this task is not entirely realistic – it is actually impossible. Despite their 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. 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.

For agile 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. It is a methodology that has programmers build unit tests before writing code, thus instilling quality as the main success factor of the software product. It has a different kind of order from the classic methodology – one that focuses on code rather than paper, and on value as opposed to blame.

Tell a Story

Forget the words *use case* and instead say *storytelling*. In theory, it is the same thing, but in reality it is much different. To illustrate this difference, try searching for each of these terms on Google Images: "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 diverse ideas start to flow. This way, you encourage them to be more creative and not settle for simple screen reviewing. You empower the testers without actually using the cliché word "empowering." This change in terminology helps them go the extra mile to feel the pain of the user working with the piece of software.

Testers should be involved throughout the design phase. They should help with the specifications development and requirements gathering.

In an agile environment – where testing is being done continuously – it is especially important that you keep things fresh.

Define Testing Strategy

Make sure to define your testing strategy early on. Start from understanding that a software requirement specification (SRS) is quite similar to a test plan. Therefore there is no reason to maintain both types of documents in the fast agile world. 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.

Testing is not a phase in 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 getting done. One benefit of involving QA early is that the testers ensure application requirements will be testable. Many times with the agile approach, the testers are able to begin building test use cases during the initial design phase.

What to Test

The multi-dimensional matrix of what needs to be tested should be defined early. The importance of this has grown exponentially during the past few years, as the matrix has become increasingly complex. 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 and what should be offshored or outsourced for greater coverage.

Testing Metrics Should Speak Business

Testing managers often focus on information that is of little use to anyone - keep this in mind when collecting data. They are 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 by type of discovery. This type of information is like gold to decision-makers. If you can collect this data, then do so immediately. Just use data properly – sometimes metrics collection becomes a task of itself, while the message does not go up to the management.

Play Well with Others

A true collaboration must occur between development and QA. Without collaboration, agile becomes a set of forced procedures and will not help create quality software. All members of the team must be exposed to the same data in real time so they can have their fingers on the pulse of the sprint. This means a common version of the truth must be made visible to all members. This includes sponsors outside the SDLC process in the form of reporting that relates to the business needs as well as to other development teams who may or may not be using agile or at least a different flavor of agile.

Alert Only When Needed

If every defect discovered by the tester gets the highest priority, developers will inevitably come to overlook some of the ones that are really important.

On the other hand, if you continually say "Ignore this bug, it's a known issue" or "Ignore that test, it's not yet relevant," then your testing team will get accustomed to ignoring bugs, and they will miss the ones that you do not want ignored.

Be careful in assigning the severity and priority of the bug so that the developers will able to comprehend the significance of the problem.

Timing Is Everything

The short release cycles of agile development can often place QA teams under tight time constraints. One way to overcome this is to be in constant contact with the developers during the iteration. The earlier a bug gets discovered or feature shortcomings show up, the easier the fix. The QA team's involvement allows testing as the engineers develop. There is no need to wait until the end of an iteration to produce a list of defects. As defects are discovered, that information is fed back to the development team, which addresses them as part of the development process. The cost of solving defects early in development is much lower than solving them later. The risk that a defect might generate issues with other, related pieces of code is also reduced. Overall, the team will have a higher level of confidence in the quality of its development output, because it knows defects are being continually identified and addressed.

Not Only Functional Testing

As testing becomes integrated with development – and as testing activities become more and more compressed – there is a tendency to forget about other aspects of quality assurance, namely application performance validation and security assessment.

At what point will your application's performance begin to degrade? How many concurrent users can it support? How do the various components of your application behave with 50, 100, or 1000 active users? Where are the bottlenecks between your code base, database, and load balancers? Without answers to these questions, your testing and development efforts could become meaningless. If your software can't hold up under stress, it does not matter how agile your methods are. To be prepared, start from performance risk analysis early in the iteration so that you can map all of the potentially weak areas in advance – even before any actual code is written.

Similarly, the only software that matters is secure software. Attacks on software by hackers, criminals, and insiders can result in business interruption, brand damage, tremendous financial loss, and harm to innocent people. The targets of these attacks are hidden vulnerabilities within software applications. These vulnerabilities accumulate within software, waiting to be exploited. To make matters worse, new vulnerabilities are continually introduced into organizations from their own internal software development groups, as well as through procurements from vendors, outsourcing firms and open source projects.

Addressing potential security threats in a fast manner during short interaction cycles of agile development increases the chance of early discovery and fix. If developers are notified about the security threat of a feature they just checked-in for the build, they will be far more capable to address this problem in a timely fashion and may even be able to commit the fix for the next build. For example, you might add a malicious user story in the planning stage so that it gets addressed during iteration validation phase.

Meet Often

Although Scrum is not the only method to conduct agile testing, 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.

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. To avoid this, set a time that works for everyone and do your best to stick to the schedule.

3 Making Agile Testing Work

The reality of the current software development scene causes many team leaders to ask about how to transform into a more effective agile entity. If there was a Wizard of Agile, the following conversation between a team leader and the Wizard might occur:

Team Leader: "How can I make my team agile?"

Wizard of Agile: "You can't change people; you can only help them change themselves."

Team Leader: "How can I help them?"

Wizard of Agile: "You must first become agile yourself."

Team Leader: "How do I become agile?"

Wizard of Agile: "It's simple… but not easy."

To take the mystery out of the task of launching agile methodology, let's consider the main aspects of the journey to becoming agile.

Roles Shifting

The introduction of agile methodologies shifts traditional organization roles. The triumvirate of business analyst, developer, and QA person transforms into the new agile roles of domain expert, programmer, and tester, with each role having some overlapping responsibilities. Here are some of the roles:

— Scrum master

This person is responsible for deadlines and sticky notes.

— Product owner

Sometimes called domain expert, this person defines in general what needs to be done but without too much details.

— Programmers

This person implements stories with high business value using good techniques and practices.

— Testers

This person provides information on the state of the project.

— Development managers/team leaders

These people participate in planning discussions and bear overall responsibility over the group of programmers in terms of delivery.

Depending on the number of people and complexity of the project, there are additional roles of:

— Project Manager

This person does whatever it takes to ensure that valuable projects are delivered now and in the future.

— (Agile) Customer

This person ensures the team works on stories that deliver the highest business value possible.

— User Experience Designer (UX)

This person helps the Customer discover usability needs, helps the team meet them, and verifies they have been met.

— Architect

This person guarantees the team implements good techniques and practices.

Remember that the above definitions are really roles and responsibilities, not people. One person could fulfill multiple roles within the agile organization structure.

Since one of the main appeals of agile methodology is a promise to work as one team and not as the silos of the classic approach, this structure has some clear advantages:

— Team, not individuals

The team should work as a unit, not as a set of individuals. The team leader should find ways to renew the team members' commitment and their purpose to each other. Remember: Not everybody can be motivated to the same level. One very motivated person can do a lot of work, but the

opposite is true as well. Annual review of each team member should be focused on an individual member's personal achievement, not his/her team's achievement.

— The whole team is committed to quality

Quality is maintained continually with the agile development approach. Both the testers and programmers have more confidence in the code: they know there is a reduced chance that undiscovered defects will create problems downstream.

— The whole team is responsible for testing

The testers also do a risk assessment: prioritizing defects to ensure that those which have the most potential impact on the software receive the most attention and development resources.

— Testers get support and constant training

Because the testers are more integrated with development, their skill sets are also different than they were before the company moved to the agile approach. There is a need for testers with a technical profile because they have to dig deeper into the code. They are not only working on a GUI or user interface level.

— Developers do testing activities

Unit testing is the de facto standard in agile teams. The best scenario, adopted by quite a few teams, is to have unit tests be written during the coding, or even before that.

In addition, Application Lifecycle Intelligence (ALI) equips ALM with the ability to monitor source code changes, flag untested changes, report if a defect is linked to a code change, and warn if a code change is not related to a business requirement. It enhances ALM by aggregating data from a variety of sources, allowing decision makers gain access to meaningful information and visibility into the potential risks of change in application projects. All this happens in the familiar development environment of choice without changing the way developers work, while giving them insight into the requirements and quality assurance world that usually gets forgotten in the waterfall development approach.

— Testers and developers pair to tackle certain problems

Each member of the team brings his/her own perspective on the problem, thus bringing resolution closer. Since all members work for the same team, there is no problem of broken communication and vested interests.

Collaboration Equals Strong Team

In many ways the tester and developer roles are in conflict. A developer is committed to building something successful. A tester tries to minimize the risk of failure and tries to improve the software by detecting defects.
Developers focus on technology, which takes a lot of time and energy when producing software. A good tester, on the other hand, is motivated to provide the user with the best software to solve a problem.

Agile strives to break misperceptions and create a team unified by the goal of delivering quality software on time. It is about delivering the highest business value possible more quickly by focusing on people and continuous improvement. While collaboration remains a key theme, something more elemental needs to be addressed at the very beginning - the importance of mutual understanding. This is the basis of building a strong team, where individuals communicate in a mutually agreed language to deliver quality software. Without this cooperation, the testing function would have a difficult task defining the test criteria and making sure the necessary quality levels are achieved. The only possible solution is for testing and development to work together. Testers can be powerful allies to development and along the way they can be transformed from adversaries into partners. This is possible because most testers want to be helpful; they just need a little consideration and support. To achieve this, however, an environment needs to be created to bring out the best of a tester's abilities. The tester and development manager must set the stage for cooperation early in the development cycle and communicate throughout the cycle. Many times it results in physical colocation of development and testing sides of the project, encouraging face to face interactions.

Much better communication between developer and QA greatly simplifies the process as it means less bureaucracy and more coding – hence producing faster delivery. Testers are represented at every stage of SDLC and have early influence over quality related decisions. As test design and execution are done earlier in the cycle, it is still possible to impact and change a piece of troubled code.

To summarize the above, there are clear benefits to the agile approach:

— One team.

QA has never been closer to development and becomes part of the cycle.

— Quality is now a concern of the entire team and not only the tester's.

— The skillset level of a QA person rises due to constant interaction with developers.

— Everything happens very quickly.

The quicker the testing, the better the influence.

— Quality level improves due to early testing and defect detection.

Approaches like unit testing (to validate that the right thing is developed correctly) and acceptance tests (to validate that the right thing is developed) are not new. In the past, however, these approaches have often been handled in an isolated way. With agile, a comprehensive, pragmatic solution is preferred, which focuses on the requirement itself, while also having the stakeholders in mind. The solution can be found in ALM, which helps to overcome various barriers:

— Barriers between project phases and project activities. Because coding and testing

move together more closely.

— Barriers between artifact types.

Because code and executable specifications are written on the same unified infrastructure.

— Barriers between project roles.

Because tests are written collaboratively, with mechanisms to use terms close to the problem domain.

Fast Cycles

Traditional waterfall methodology of systems development is essentially a sequential solution development approach. A common problem with the waterfall model is that the elapsed time for delivering the product can be excessive.

By contrast, agile development accelerates product delivery. A small but functioning initial system is built and quickly delivered, and then enhanced in a series of iterations. One advantage is that the clients receive at least some functionality quickly. Another is that the product can be shaped by iterative feedback; for example, users do not have to define every feature correctly and in full detail at the beginning of the development cycle, but can react to each iteration's results. With the agile approach, the product evolves continually over time; it is not static and may never be "completed" in the traditional sense.

Micro Focus's internal agile teams' experiences, as well as those of its many customers teach that there are some common periods in the cyclic nature:

— Iterations, usually called "sprints" in SCRUM lingua, last between 2 and 6 weeks, with 4 week cycles being the most popular. The outcome of the sprint is the testable version of the code.

— Milestones, when the working version of the product is built and delivered to the business customer, occur every 3-4 sprints. At this point, the planning for the next milestone is performed. Customers get a chance to see an incomplete but working version of the final product, and supply feedback which is incorporated into the plan.

— Daily meetings to discuss current state of the team. These meetings, usually conducted near the drawing board and filled with sticky notes, bring team members together and allow sharing of problems and ideas in an open manner. Team members choose the next tasks to perform as they want, keeping skill sets in mind.

— Simple statuses are in place to track the progress. They usually include:

- To-Do: an initial status after the task has been taken from the queue.
- In Progress: when the feature is actually being developed.
- Done: when the feature passes an acceptance test.
- Blocking: what is blocking me from accomplishing the task? The blockage may have a technical or nontechnical cause.

These, of course, are just examples of statuses – different companies may enhance the list with additional values that best suit their business nature.

These short periods encourage quick response from the tester and good collaboration with the developer. Without them, it may become too late to affect the product decisions that evolve over time.

Agile teams use smaller increments and just-in-time collaborative planning to ensure critical changes can be released sooner. If something is important, it can be moved into an earlier release or iteration. It may displace another feature that isn't as important, but it won't disrupt an entire release. The result: faster delivery.

Planning is Key

Software development is all about implementing requirements. The requirement is the central unit and the driver of a software release.

Traditional requirements-based testing expects that the product definition will be finalized and even frozen prior to detailed test planning. With agile development, the product definition and specifications continue to evolve indefinitely; that is, there is no such thing as a frozen specification. A comprehensive requirements definition and system design will probably never be documented.

With all this said, the planning phase is probably the most important part of the process. During milestone discussions, the whole content of the project, sometimes called the *backlog*, is taken as the basis for the next milestone effort. The backlog is then broken into smaller pieces called *user stories* which are essentially the requirements, but without too many details and described in a human way.

From user stories, team leaders derive *tasks* – activities that should ideally take no more than 3-4 days to accomplish. If a task is expected to take more time, then there is a need to divide the user story into multiple smaller user stories so they can be combined into the task at the end.

The task becomes an ultimate unit of work, and developers take them from the general pool as they please, based on skillset.

There have been attempts by the fresh agile teams to oversimplify or skip the planning process altogether as being too formal. Do not let it happen; planning should never be skipped or become irrelevant. On the contrary – investing the required amount of time, bringing all relevant parties to the table, and cutting the big pie of backlog items into user stories of appropriate size will ensure a healthy flow of events during the milestone period. As you work in agile teams, use your agile skills to calibrate planning process – make it longer or shorter as per business needs.

Communication – Agile's Main Vehicle

As the whole team, including the programmers, testers, customers, is responsible for the outcome, only strong commitment from each and every team member makes it possible. To facilitate this outcome, everyone on the team should be accessible and actively communicating throughout the project.

QA engineers in waterfall tend to rely on documents. Agile testers don't get a big requirements document as a basis for test cases and don't get three months to write test cases before they see a working piece of code. They jump into the communication stream, which may be verbal, written, or virtual – and assimilate the information they need. They learn to ask the right questions at the right time and provide the right level of feedback at the right moment.

Unfortunately, testers are typically:

- Bad at understanding what customers want to know.
- Bad at editing out information that is not necessary or relevant (more is *not* better).
- Bad at translating technical problems into their likely business impacts.
- Bad at presenting data in a way that is easy to digest.

To improve the widespread attitude towards testers and effectively communicate their value, there is a need to invest a serious effort in crafting information for numerous stakeholders in the team and the company. Use modern tools like wiki to produce test documentation and provide test statuses and outcomes. These two can reside on the same wiki page and serve as both test design and test results tracking. Make information visible!

If you want to achieve a high level of quality within your systems then you need to embed quality professionals within the development teams to iteratively review and advise on the ongoing work (thereby reducing communication time and decreasing opportunity for information loss via documentation). Note that you may still have an independent test team which externally validates the project team's work, but the majority of quality professionals would be embedded within the team.

Automation Is the Name Of The Game

Since development cycles have been shortened and testers do not have the luxury of having months to design and execute test cases, the agile approach demands quick turnaround of testing activities.

Shorter cycles increase agility. Fortunately, with agile projects the software is ready to test almost from the beginning. Typically, agile teams employ several levels of testing to uncover different types of information:

- Automated unit tests

  These tests check the behavior of individual functions/methods and object interactions. They answer the question: "Does the code do what the programmer intended?" The unit tests run often, and provide feedback in minutes.

- Automated acceptance tests

  When you need to check the behavior of the system end-to-end, use acceptance tests. These tests are the most important ones as they provide answers to questions like, "Does the system do what the Customer wants?" They typically run on checked-in code on an ongoing basis, providing feedback in an hour or so.

- Automated regression tests

  Because of the cyclic nature of agile projects, there is a need to verify the state of the product to see if what used to work is still working. Manual regression tests take longer to execute and, because a human must be available, may not begin immediately. Feedback time can increase to days or weeks.

  It is no wonder agile projects favor automated tests. Actually, without automation it is nearly impossible to keep up with the constant change that emanates from each cycle. Hence there are some simple ideas that would allow an agile tester to stay ahead of the curve:

— Plan for the current iteration.

  Design tests for the features or stories to be done in the near term. Many times proactive planning means rework.

— Use catalogs of reusable tests.

  Do not redesign the same tests over and over. Instead, keep reusable checklists.

— Keep an up-to-date, prioritized risks list.

  What kind of information are the testers looking for? The risks list should cover it.

  Combined with source code, build automation, automatic deployment to test environment, and continuous integration, testing automation is one of the main facilitators of fast and quality delivery of software products in an agile world.

  However, manual testing, particularly manual exploratory testing, is still important, even on agile projects. Traditional test wisdom says it is impossible to start testing a feature until it is accessible from an external interface like a GUI. But there is no need to wait. Test automation can facilitate manual exploration which provides additional feedback and covers gaps in automation and is necessary to augment the automated tests.

  There are, of course, more types of testing that need to be applied to the software – for instance, cross environment testing, testing in customer- specific environments, internationalization and localization testing, and many more. Just like the functional testing, they can and should be automated as much as possible and addressed during the agile period.

Start Small

One of the important steps in spreading agile methodology within the company is conducting a pilot project. Its purpose is to:

- Verify the concepts formulated in books or by agile trainers.
- Demonstrate the initial value of establishing an agile team.
- Market the idea of an agile development inside the organization.

Usually the process starts with one team or project, and depending on the results, it is rolled out to the entire organization.

To conduct the pilot, an organization should determine the initial focus of the newly formed agile team and define the exact contents of a specific development project. When preparing for a pilot stage, it is necessary to address as many aspects of agile as possible, but on the smaller scale:

— Planning the pilot

- Conducting generic introductory sessions on agile principles
- Possibly bringing in an agile coach
- Identifying project content and its business representatives
- Collocating team members in one physical place
- Defining members of the pilot
- Making a timeline of the pilot project and recognizing delivery points
- Documenting the progress of each step

As the experiences of Micro Focus customers shows, the pilot team gets transformed while working on an existing product module or its enhancement. Companies create small teams with predefined small-to-medium project life spans and put them into one location. The team is given the liberty to define its structure and working methods but takes full responsibility for the outcome. Even though a pilot is conducted on a smaller scale, it is imperative to enlist executive management support. Regularly providing the management sponsor with reports on the progress and problems will help ensure that support is forthcoming.

At the end of the process, it is a good idea to conduct an analysis session, sometimes called "retrospective," for both the members of the pilot team and their customers. Extensive feedback allows you to prevent unnecessary mistakes in the future, and facilitates streamlining expansion of the agile methodology. If possible, the executive sponsor should participate in this analysis session. In any case, there should be a document describing the process and its findings which can be used later as a master plan for rollout.

Central Is Agile Too

While most agile teams resemble the structure described above, there are usually some specialized cross project teams with unique skills in infrastructure, test lab admin, load testing, security assessment, etc. This comes as no surprise considering the unique skills required from the members of such teams. It takes a lot of time to master the art of performance engineering or security analysis. However, the specific nature of these teams should not prevent them from participating in agile processes. In large organizations, it is not unusual for centralized teams to have some advantages in supporting enterprise initiatives because they can provide a different prospective and more holistic view on all non-functional problems.

These are some core principles that should be followed in involving a specialist:

- This person works exclusively with one point of contact in each team.
- In some cases, one specialist is dedicated to the specific team. In other cases, one engineer serves multiple teams. It really depends on the size and complexity of the project.
- Each engagement should start with the formal kick-off meeting, where each party has visibility into its goals and timeline.
- Each engagement is concluded with a sign off document using a predefined template.
- The involvement requires commitment from the development team manager to assure that non-functional aspects are considered and taken care of.

The iterative nature of agile projects perfectly fits non-functional testing objectives – getting involved and applying influence at earlier stages of the project allows addressing inherent problems and leads to a successful fix rate.

Problems With Agile

Another aspect of the agile adoption has to do with multiple variants of the process. No company is the same and this is reflected in the way application development and testing is performed. Micro Focus, through its wide customer base, has the privilege of watching different forms of agile implementations and agile testing techniques.

Agile is much more about company culture, human relationships and business procedures than about specific tools, slogans and tricks. It is much faster than waterfall, but it can still have many of the same problems as some traditional project lifecycles: bad requirements, communications issues, lack of documentation.

While agile has huge potential and a great adoption rate, it has some problems and limitations. Management should take the following into consideration prior to global roll out of the methodology:

- The agile approach seems to work best with smaller groups.
- It is a lot easier to manage when people are physically located together. However, there still examples of successful agile teams located at multiple sites.
- Another benefit of physical colocation is that offshore development is problematic and puts enormous pressure on day to day operation.
- Sometimes it is really difficult to test something prior to the end of the whole project even if each iteration testing was marked as "done". Many times this status is given without full satisfaction on the QA side.
- Many times GUI testing is problematic in agile as everything may break out in the next iteration.
- The planning phase still takes considerable time – at least at each milestone.
- Wrong planning at the beginning of the milestone gives way to problems, i.e., if backlog items were cut improperly, the user story is too big or too small – this can affect the entire development cycle.

It should be clear from the very beginning that agile is no panacea for problems related to development, process, and management. However, if you follow the Micro Focus advice and constantly examine the situation, it can be extremely beneficial to the quality of the software output.

4 Conclusions

The agile approach has been around for quite some time now. It has proved invaluable in many of the complex projects being carried out by small and big companies alike. The most innovative companies of today and tomorrow will continue to push the agile methodology envelope. For them, being able to effectively define, plan, and execute projects in a dynamic and high-speed environment will be the difference between merely surviving and flourishing. Agility is about making the right decisions during project *execution* as well as *planning*.

From a testing perspective, the biggest challenge of being on an agile project team is that the software is *always* changing. With new code being pushed into the test environment daily or even hourly, the system you are testing becomes a moving target. Dealing with that kind of change can require different approaches to testing, with different skills, tactics and tools. There are certain aspects of testing software that do not change just because the project team is using an agile approach to implement software, particularly the mechanics of test execution, but some testers may need to make significant changes to their testing approach if they are going to add value on an agile software project. Agile testers have to make decisions about what work to do next, how to do it, how to make it meaningful to the client, and how to exercise the application in different ways to improve the understanding about how things work and where risk might be.

We are all interested in improving the quality of the applications we produce on behalf of our respective departments, lines of business, and enterprises.
What is required to make this goal achievable is a lifecycle approach to quality, proper planning and design of the environment, consistent use of best practices, state-of-the-art tools such as ALM, and intimate knowledge of company business.

We hope that the best practices listed in this document will help in adoption of agile methodology and utilization of ALM in your organization.