Getting **QA and Developers** to Work Together

WHITEPAPER

ZEPHYR
Real-Time Test Management

For many development shops choosing to implement an Agile process, one thing becomes abundantly clear: QA analysts and developers have to learn to work together.

It's not that QA analysts and developers didn't ever have to work together before, but because the development and testing process is so different—and so intertwined—in an Agile environment, the need for cooperation between developers and QA becomes much more apparent.

Table of contents:

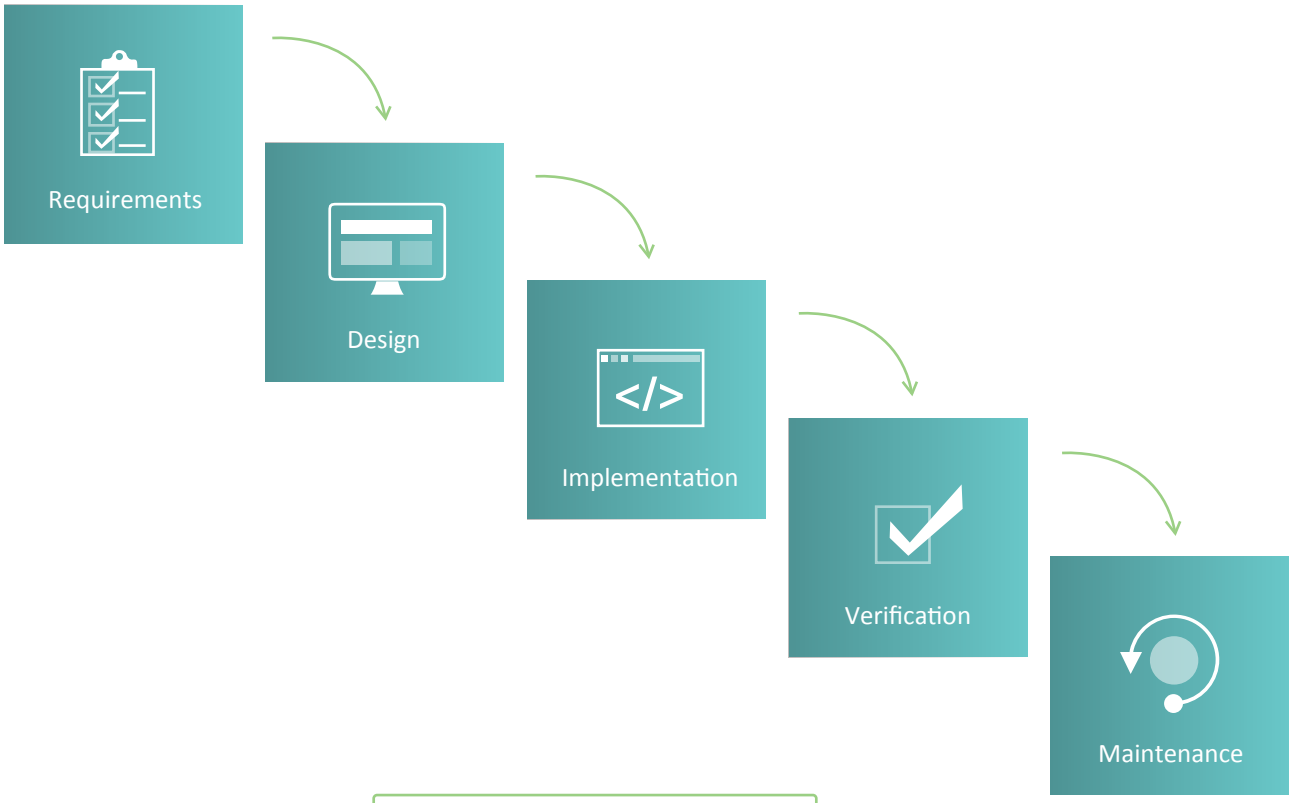
How things work now...

In many development environments it is commonplace for developers to work on some code and features independently of QA and when they are done pass that code over to QA to test, only ever hearing back from them when a bug is found that needs to be fixed. *(This is often called throwing the code over the wall.)*

Most software development shops run in this fashion, because it is the default way of doing things. It is pretty common for software developers to sit in a completely different corner of the office than their QA counterparts and it is pretty common for them to have minimal interactions.

When organizations were writing code using a waterfall approach, they could afford to have this disconnect, because they tested everything at the end of the development phase and they had detailed specifications that told software developers and QA analysts exactly how the software was supposed to behave. *(Or at least they were supposed to have those detailed specifications…)*

With the waterfall approach to software development, software developers and QA analysts could work independently of each other much more easily, because it was a downstream process. Developers would write code and they would pass that code downstream to QA who would test the code. Bugs would flow back upstream until the software had reached the level of quality that was expected.

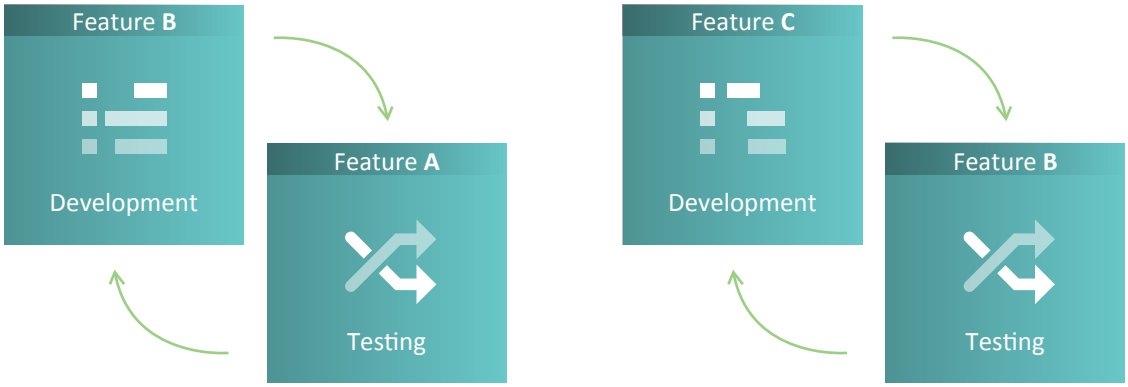


The traditional waterfall model

How **Agile changes** things

Many software development organizations practicing Agile methodologies still try to operate in this fashion. It is common to see software development teams staggering iterations so that the developers will work on writing code for one iteration while the QA team is testing the code from the previous iteration.

This kind of methodology is not really Agile, it is mini-waterfall. **When organizations are developing code in this manner, they are simply taking the waterfall process and chopping it up into smaller segments.** They are getting all the pain of the waterfall process and missing many of the benefits of Agile.



Staggered iterations (mini-waterfall)

To truly operate in an Agile environment, entire organizations need to be Agile. Sure, that is easy to say, but what does that actually mean? It means that organizations need to be able to respond to change quickly and to iterate their software in response to that change. Here is an example that will make things a bit more clear.

Suppose a team is developing a feature for call center software that allows a customer representative to provide detailed notes about a call to their supervisor for review each week. The software developer on that team's project writes the code for that feature during the current iteration, while the QA team is happily testing the features the development team had completed last week.

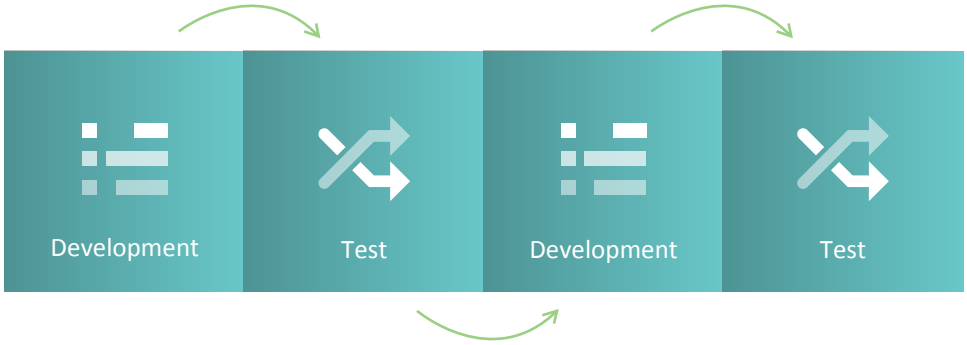
The development team finishes the feature and in the next iteration, they move on to another feature and hand the call notes feature over to QA to test. Well, it just so happens that when the QA team starts testing the feature, the customer who is going to use the software tries it out as well and discovers that they actually want the feature implemented in a completely different way.

Obviously, at this point it doesn't make sense for QA to continue testing the feature, so they stop. But now, the developers are already working on something else, so they either have to stop what they are doing and start working on the call notes feature again, or they have to put off the changes to the feature for the next iteration. Even if they manage to complete the requested changes to the feature in their current iteration, it will be a whole other iteration before the feature is tested. Not very agile.

How **working together** makes you more agile

So, following an Agile process alone does not make one agile. In order to really be agile, in the true sense of the word, meaning an organizations can respond to change rapidly, they need to have their QA and development teams working together **each iteration**.

Instead of developers completing code and throwing it over the wall to be tested in the next iteration, **a truly Agile team will have developer and QA team members work together during the current sprint to both develop and test a feature.** By doing this, a team is able to respond to any changes immediately and is truly able to iterate the development of their software.



Development and testing in the same iteration

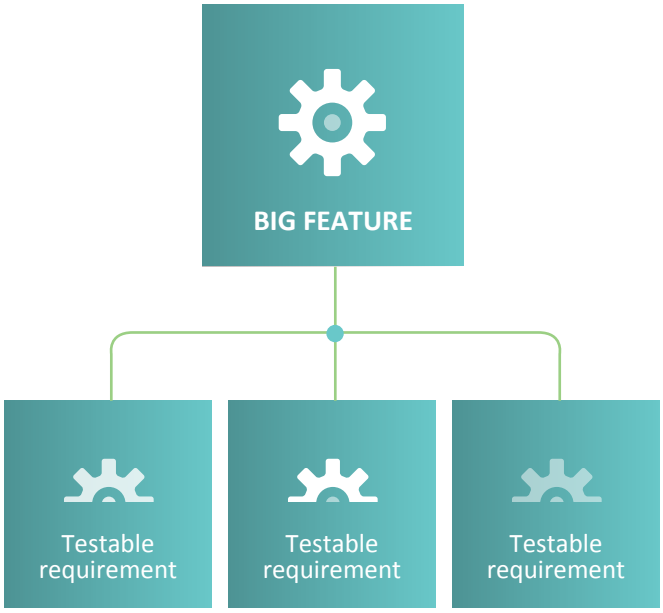
Going back to the example with the call center software. It can be seen that if the QA team was testing the call note feature in the same iteration that the developers were developing it, making changes midstream would be a much easier task. The developers could simply make modifications to the feature and give those modifications directly to QA in the same iteration. No time would be lost waiting for the feature to make it completely through the pipeline.

How to **create a unified team**

Of course this is all easier said than done. How can a team develop and test a feature in the same iteration? Doesn't the feature need to be developed before it can be tested?

If a team looks at a feature as an atomic element that can't be broken up into smaller parts, then yes, it does. But, most features are not unbreakable stones. Most features can be broken up into smaller pebbles which can be developed and tested independently of the whole.

To look at features and develop them this way requires coordination and communication between the developers and QA analysts. When a new feature is going to be worked on in an iteration, the development team needs to meet with the QA team together to talk about exactly how the feature is going to be broken up and what exactly is going to be tested.



The term test driven development, or TDD is used to describe the practice of writing failing unit tests before writing code in a software project. This same idea can be applied at a higher level to the development and testing of an actual feature to allow the tests to drive the development of that feature.

In this scenario the development and QA teams meet together and the first thing that is decided upon is the high level tests that will be used to verify the correctness of the feature. Developers and QA analysts work together to define, at a high level, what test will be run and created to test the feature. Developers then start writing the code that will be necessary to make those tests pass, one test at a time. Each time the development team has enough code created to make a test pass, that code is handed over to QA to execute that test against the code for the feature. One by one, the code required to make each test pass is written and tested and little by little over the course of the iteration the feature is both developed and tested at the same time.

This process of developing features one test at a time requires the entire team to work together as a single unit to complete the work for a sprint. It requires developers to understand more about the testing process, since they will need to know how the features they are developing will be tested. It requires the QA analysts to know more about the development process, since they will need to be aware of when certain parts of a feature are ready for testing.

Inevitably, when bugs are found using this process, they are handled immediately. When QA member finds a bug, it indicates that a particular test that a developer thought should pass, does not pass. Work doesn't move forward until the failing test passes, so bugs are always fixed as the software is developed, not after. (At least not bugs that are found by normal testing.)

Metrics and Dashboards

Most Agile teams use either a burndown chart or a wallboard with different lanes to show the progress of items being worked on and to let the team know what the priority of work is. These tools can be an important part of getting development and QA teams to work together.

Having a central location where team members are able to see priorities and the progress of work being done during the iteration, helps the team to have a unified goal and to visualize how work will progress during the iteration.

When adapting a truly Agile process, as suggested in this paper, many teams struggle with a huge backlog of QA work being pushed to the end of the iteration. Breaking down features into smaller components and testing them as they are finished can reduce this problem, but it can also be very helpful to have a burndown chart or wallboard that clearly shows the work backing up before it becomes an issue.

Communication is key

This whole process hinges on one very important thing: communication. For many organizations this is the single hardest barrier to overcome. In many development shops QA teams are not used to communicating with development teams with the frequency that is required to be successful in an Agile environment.

There is no time for indirect methods of communication like complex bug tracking systems and long emails. The development and QA teams need to learn to communicate most things immediately instead. Co-location can help make this communication easier, but when that isn't possible, instant messaging systems can be another fast way to communicate. The key is to drop many of the formalized—and slow—communication systems QA and development teams are used to using in favor of fast, immediate communication.

It is also common for QA and development teams to develop a sort of rivalry in many organizations where they act as opposing forces. This kind of ingrained mentality can be difficult to overcome, because developers often look at QA analysts as enemies who seek to find problems with their work, and QA

members often have a similar view of developers or equate their job performance to finding a large number of bugs. **Both QA and developers have to learn to work as single team with the purpose of shipping quality software.**

While QA and development teams continue to hold an antagonistic view, it is impossible to have worthwhile communication.

Active steps must be taken to break down these barriers and join the two teams together. Group activities and team building exercises that incorporate the whole team together can be helpful, but the most success is often achieved by mixing up roles to some degree. It is helpful for QA members to learn a little bit of development and even pair program with developers. And, it can be equally helpful for developers to participate in some QA activities like helping to run or create tests.

Automation

Another critical area for developers and QA to work together is test automation. Successful test automation efforts require cooperation from both developers and QA. A common mistake is to hand all of the test automation efforts over to the QA team instead of making it a joint effort.

Maintainable and cost effective test automation efforts require the creation of a test automation framework. This framework has to be developed, just like any other piece of software. But, most of the time actual automated tests are created and maintained by QA analysts. Coordination is required between development and QA team members to design and create an automation framework that will support the creation of automated tests by non-developers.

Some of the most successes come when test automation strategies involve developers creating and maintaining the test automation framework, while QA analysts develop and maintain the bulk of the automated tests.

Putting it into action

Many organizations agree with what has been said in this paper, but don't know where to start to get their development and QA teams working together.

Here is a simple list of steps that an organization can incorporate to get things moving in the right direction:

1. **Start by co-locating teams.** If this isn't possible, set up communication channels that make it much easier for developers and QA analysts to interact. Try to break down the barriers between the two teams and treat them as a single team.

2. For organizations which are currently staggering iterations so that QA is working on testing the iteration development has just completed, **change it so that QA and development work on the same iteration.** Those organizations can start out by having developers hand over each feature to QA when they are finished with it in the current iteration and later graduate to breaking features into smaller parts and doing a test driven approach.

3. **Get rid of any performance metrics related to bug tracking.** Having developers measured on how many bugs they create or fix and QA measured on how many they find, puts both parties at odds with each other and breaks down effective communication.

4. **Try swapping roles for an iteration.** Move one QA person to the development team and vice-versa.

5. **Put an extra emphasis on fixing bugs, not reporting them.** The goal of the team should be to ship quality software. Bugs need to be tracked, but only for the purpose of getting fixed. Changing the emphasis to fixing bugs and shipping software will force both QA and development to work together to achieve the task.

By getting QA analysts and developers to work together many organizations are able to deliver a higher quality product is less time, because they avoid the staggered approach of split iterations that force both developers and QA to revisit old work and constantly switch context.

Many organizations can benefit directly applying the principles in this paper to get their teams working together instead of operating as two different teams with different goals. It is a challenge, but an effort worth undertaking for any organization that truly wants to benefit from an Agile process.

About Zephyr

Zephyr is a leading provider of quality management solutions, powering intelligent DevTestOps for more than 11,000 global customers across 100 countries. Project teams and enterprises of all sizes use Zephyr's products to enable continuous testing throughout their entire software delivery pipeline to release higher quality software, faster. Zephyr is headquartered in San Jose, CA with offices in King of Prussia, PA, Europe and India. For more information, please visit www.getzephyr.com.

ZEPHYR
Real-Time Test Management

Contact Zephyr Today!
sales@getzephyr.com
www.getzephyr.com
+1-510-400-8656