How Scrum has taken the tester’s role to new heights

Scrum has been key to strengthening Agile methods in systems development organizations around the world over the past decade, while Agile has gone from being quirky and controversial to essentially standard practice. But how have methods like Scrum altered the tester's role in development organizations? In this article, you’ll learn more about some of the most significant changes.

Testing has crept ever closer to development

According to Scrum, the team should be as self-sufficient as possible. This means that collectively, the team members should have every skill needed to complete a task; and since few functions can be considered complete without testing, most teams need testers. No other facet of Scrum methodology has had a bigger impact on the testing role than this. In contrast to traditional development methodologies, Scrum has pushed testing into development teams, physically as well as procedurally.

Historically, development and testing have been separated by an organizational chasm, one which persists even in some companies that have adopted Scrum. However, by planning and conducting testing in close conjunction with development work, these organizational boundaries are slowly fading away. Testers now enter the team earlier, often at the same time as developers. Seeing it from the development side, traditional testing tasks used to be an external activity of planning tests, executing them on an ongoing basis, and reporting results. Nowadays, most testers sit scattered across development teams, with each team sharing responsibility for software that is coded, tested, and ready for delivery.

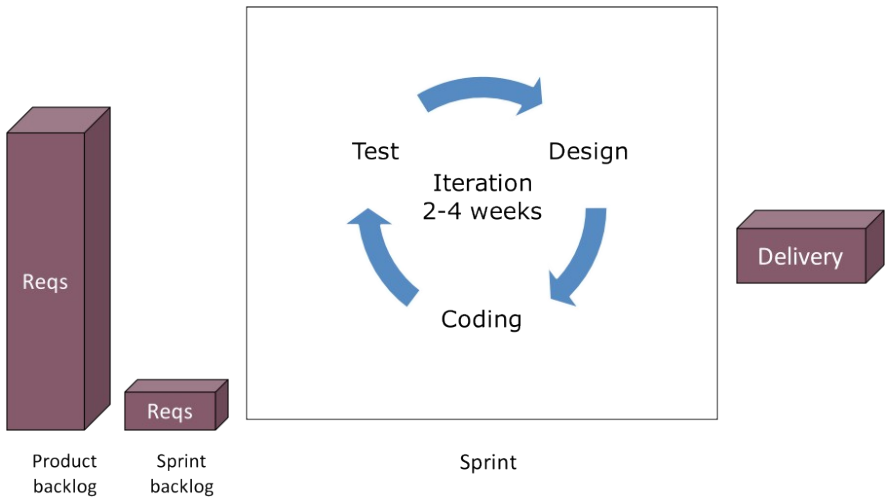

Figure 1 – The Scrum development cycle.

Naturally, some testing activities should stay separate from development; partially to maintain objectivity, and also because these activities come later in the sequence and hence can’t be planned at the same time.

Testing work starts earlier

Scrum automatically assumes for testing in the development process as early as possible – something which is undeniably positive and which most organizations strive for. Testing can review the business requirements early on and, based on the testers' skills, influence the system design. This gets the testing discussion started early – even before the team starts writing code – and keeps testability from falling into the background where it frequently gets forgotten. Without this focus, teams often forget about testing during the requirements and design stages.

The earlier you detect defects, the cheaper they'll be for you to fix. Defects are also referred to as bugs, design errors, or other terms that all signify something that's different from what is expected. Correcting these deviations costs the team dearly, in the form of extra work like additional documentation, regression testing, and potentially expensive re-delivery to the customer. Cost, stress, and embarrassment are all excellent reasons to introduce testing right out of the starting blocks and to keep trying to discover errors as early and as often as possible. There are countless cases where, during an early requirements discussion, a tester's simple question made a client or coder rethink a critical point. Discovering defects at the design stage is considerably less costly than fixing failures found later.

Work proceeds in iterations

When testing and development teams plan work together, the development team can deliver work more frequently for testing (and of course, working together in a team boosts cooperation and camaraderie). If team members can hand over internal deliveries informally, they can spend less time on the detailed documentation normally required. Assuming that the team has flexible build and deployment tools, it can deliver work to testing frequently, with very small iterations inside the sprint (see Figure 2).

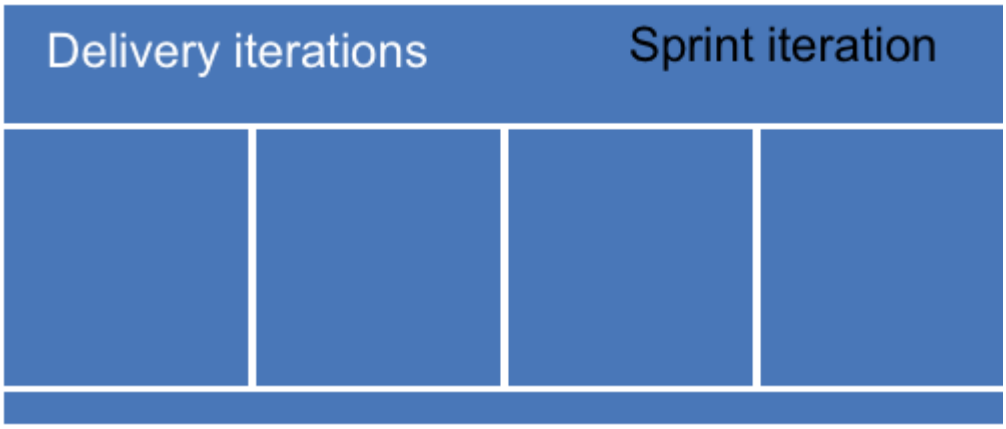

Figure 2 – The sprint iteration contains delivery iterations.

Continuous delivery iterations are worth striving for, for several reasons: for example, testers see software that's similar to previous iterations and isn't as "jumpy". They give developers feedback earlier, when they're still developing the software and can quickly

correct bugs while they're actively working on the code. The disadvantage, however, is the risk that developers drop their focus on quality before they throw things over the fence, for a mere "unofficial," internal delivery. Teams need to work together to find the best balance, where developers self-test software to a sufficient level before they deliver it to their tester colleagues.

With repeated sprint iterations, the product owner also naturally gets insight into how the work is progressing. By adding, removing and prioritizing tasks from the product backlog, the product owner can continuously control how the product evolves.

Close collaboration between development and testing also creates opportunities for continuous learning within the team. Testers deepen their technical knowledge about the product being developed, making them more effective at planning the testing strategy, and resolving bugs that nevertheless get through. Meanwhile, the developers gain a better grasp of how testing works and can take better advantage of testing environments, testing data and other assets that the testing team possesses.

The tester's role on the team is evolving

The "purest" version of Scrum recommends that anyone on the team should be able to take on any role. In reality, this is very rare in medium- to large-sized organizations because employees are usually either developers or testers, so teams typically have dedicated testers.

On the bright side, dedicated testers are usually allowed and even expected to ask tough questions, which would have been overstepping boundaries even a few years ago, before testing was well-rooted in the development process.

Organizations traditionally expected testers to objectively examine the deliverables they received and report back on the quality (or lack thereof) they actually found. In the worst cases, this led to repressive "quality police" but in better organizations, testing remained constructively independent. In these organizations, testing teams could report defects via various bug reporting tools without having to worry about who would be on the receiving end. In Scrum, testers are an integral part of the team, and provide feedback on delivered work in a close relationship with developers. Consequently, Scrum introduced a novel challenge to the testers' role: the risk that testers get too "chummy" with developers and don't fully expose the defects they uncover.

On the other hand, it's certainly easier to explain defects face-to-face, to someone you know well, than to a complete stranger. Testers have to stick to their stern stance regarding what the Scrum team delivers, while individual members never cease to be players on a productive team. Given this, it's crucial that they know how to (and remember to) provide critical feedback constructively.

The test manager's role has changed, too

Among these broader changes, the test manager's job has shifted into more of a coordinating role, responsible for strategically planning testing across various teams. Now,

testers can take on tasks like planning, reporting, and documentation in the team, while the test manager compiles and synthesizes their work. The test manager also has to coordinate different types of testing activities to keep major differences between teams from arising; as before, the test manager serves as the nexus of testing activities for the project, and collects and compiles test results on the team level. In large projects with parallel Scrum teams, it's easy for teams to focus solely on the one part of the system that they themselves are building, so the test manager has to ensure the integrity of the system, long before integration formally starts.

BONUS – IS THE TEST LEADER STILL NEEDED IN THE AGILE WORLD?

Efficiency through automation

Agile methodology only provides the preconditions for shortening the time it takes to transform an idea into software. In Scrum, you expect that what the team produces in a sprint (1-4 weeks of development time) will naturally be of high-enough-quality to deliver to a customer, production, or end-users.

However, this requires the testers to examine components newly developed in the sprint, as well as preexisting code that may be affected – which may be impossible without automated regression testing if the existing body of code is large. Testers should focus manual efforts on newly developed components, where it's most useful and is most likely to detect defects. You can save significant resources if the team of testers knows how to build automated test suites that can be continuously executed – even if it requires help from the developers.

BONUS – WITHOUT AUTOMATED TESTING, YOU CAN'T WORK AGILE

Value-adding activities move to center stage

According to Lean and Agile methodologies, organizations should continuously seek to eliminate waste, so no one does any work that doesn't add value. This is especially important from a tester's perspective, since testers traditionally prefer to have the project deliver smaller chunks of functionality with few defects rather than big, buggy releases.

Project planners usually prioritize bug fixes over other tasks that the team needs to perform. However, testers may be forced to tolerate defects going uncorrected, since the trouble to repair them simply does not create enough net value for the business.

On the other hand, it can be less costly to the organization to make even small bug fixes with very little beneficial impact, if the developers simply perform them on their own initiative. If the team has to document, discuss, and then prioritize defects, they can take hours to administer before they can even begin a fix. This approach obviously requires that the project has come to a stage that allows "open season" on bugs.

Scrum in the future?

Scrum is just a tool like many other methodologies and it doesn't suit every situation. It

will serve you well in a variety of problems that arise in software development, but there are others it cannot solve and it can even introduce new ones. You're expected, for example, to know which tasks need to be performed in the near future and to be able to estimate the time they require fairly well. That said, if you're in a maintenance cycle that often is incident-driven, reliable information is seldom available, and reliable planning flies out the window. In such situations, agile methods like Kanban offer a more effective approach.

Productivity and quality in software development are notoriously difficult to measure. The widespread adoption and impact of agile methods is equally difficult to measure and evaluate objectively. Regardless of the results, there is no obvious way to get back to traditional methods – even if we wanted to. Many organizations have abandoned the waterfall method, since it simply can't keep up with rapidly shifting business needs. Instead, they've adopted agile methods that allow room for natural shortcomings in requirements and a constantly-changing external environment.

With Scrum at the forefront, agile methodologies have changed the way testing relates to development. Testers enter the process earlier, and now deliver results together with developers. This makes it even more important that they are able to communicate effectively and maintain productive relationships within and outside the testing team. Testers need to increasingly question which tests will add the most value, and what can be automated. While the new approach that agile methodologies have introduced will still be around a decade from now, specific methods such as Scrum will certainly not survive, but be replaced by even more optimized approaches – their names won't be important, as long as the agile principles remain. Hopefully, working "agilely" will be so instinctual by then, that we will wonder why we ever questioned its principles at all.

Next steps

- Scrum borrowed its basic principles from Lean. Read more in the article "What does Lean mean for requirements management and testing?"
- Read more about testing automation in the article "Test Automation in Agile projects."

About the author

Ulf Eriksson is one of the founders of ReQtest, an online bug tracking software hand-built and developed in Sweden. ReQtest is the culmination of Ulf's decades of work in development and testing. Ulf is a huge fan of Agile and counts himself as an early adopter of the philosophy, which he has followed for a number of years in his professional life as well as in private.

Ulf's goal is to make life easier for everyone involved in testing and requirements management and he works towards this goal in his role of Product Owner at ReQtest, where he strives to make ReQtest easy and logical for anyone to use, regardless of their technical knowledge or lack thereof.

The author of a number of white papers and articles, mostly on the world of software testing, Ulf is also slaving over a book, which will be compendium of his experiences in the industry. Ulf lives in Stockholm, Sweden.