QA Role in Scrum

Leveraging Agile for Defect Prevention

Karen Ascheim Wysopal

karen@karenwysopal.com

Abstract

The key to a successful adoption of any software development methodology is a clear understanding of the roles and responsibilities of each team member within that framework. As Agile continues its rapid adoption in organizations across the globe, it's essential to define the role of QA in Scrum as concretely as we've defined the other Scrum Team roles.

During our team's recent Agile transition, I was approached from all corners of the organization, with questions like, "What does QA do every day in the sprint? What are their responsibilities beyond testing and reporting defects? What should they do in the early part of the sprint, when there's no software to test yet?" Despite being a recent "graduate" of several Agile methodology courses, I found myself ill-equipped to answer these questions. There isn't much information in the trainings or on the web about the role of QA in the Scrum team. What little I have found is primarily anecdotal, rarely specific, and never comprehensive. While Product Owner and Scrum Master roles are very well defined, QA Engineers fall under the generic label, "Team Member". We were on our own to define the role of QA in Scrum.

What we quickly learned delighted us: Agile methodology, with its fundamental shift away from waterfall software development, gives QA an opportunity for broader and deeper involvement in the overall software development lifecycle. This enables us better to ensure quality -- not by finding and reporting defects -- but by preventing the introduction of defects in the first place.

This paper provides a deep dive into the specific, day to day role of QA in Scrum, as our team developed it through the course of our yearlong transition to Agile. You will learn specific responsibilities and activities you can bring to your Scrum teams to transcend the traditional test-and-report, after-the-fact approach to quality assurance. And you will learn to leverage the unique components of Agile to significantly expand your scope of influence, and measurably improve the quality of your software. If you're not currently working in Agile, you will find these concepts relevant and applicable to other methodologies as well.

Biography

Karen Ascheim Wysopal is a Section Manager in Software Quality at Hewlett-Packard. She currently leads the Quality Assurance program for HPConnected.com, ePrintcenter.com, and related HP web-connected printer technologies. With over 20 years in the software QA industry, Karen's professional passions are building high functioning innovative QA teams from the ground up, defining new processes, and expanding QA focus on the user experience. She can't seem to stop breaking software.

1. Introduction

Traditional thinking about the role of QA has been *defect-oriented*: QA Engineers test, find, and report defects.

Agile methodology gives QA engineers an opportunity for broader and deeper involvement in the overall software development lifecycle. This enables us better to ensure quality -- not by finding and reporting defects -- but by *preventing the introduction of defects into the code in the first place*.

What is it about Agile that makes this possible? How does QA's role in the Agile Scrum Team enable us to release higher quality software?

2. Terminology

While this is paper does not set out to define Agile methodology, practitioners use its terms differently, so it's important to establish an understanding of the terminology used here.

Scrum Team refers to a group of people working together, with specific roles as defined by Scrum methodology. Scrum Team Roles:

- Product Owner (PO)
- Scrum Master (SM)
- Engineers (QA and Development)

Sprint refers to the time interval which the team has to complete the agreed upon – *committed* – set of work, after which the completed work is deployed to production. Generally sprints run at 2-4 week intervals. This discussion is based on a 2-week sprint cadence.

User Story is the Agile term for Requirements. And the *Product Backlog* is the set of User Stories yet to be implemented.

3. QA Responsibilities during the Sprint

The first days of the sprint emphasize user story analysis, sprint planning, and a lot of coding. New implementations may not be test-ready for several days. What does QA do during the first part of the sprint, when there's nothing yet to test? The answer is, a lot. Let's review the numerous responsibilities QA Engineers may have as part of overall quality assurance (regardless of methodology.)

Here are the responsibilities that our QA engineers are involved in throughout our sprints, starting from Day 1:

Daily Standup

Sprint Planning

Backlog Grooming

Test Plan Prep

Submit Infrastructure Requests

Cross-Team Meetings

Defect Triage

Backlog Grooming Prep

Test Plan Create/Execute

Exploratory Testing

Inform SM of Blockers

Story Acceptance Testing

Defect Disposition Exercise

Retrospective Prep

Backlog Grooming

Review Branch Merges

Sprint Demo

Sprint Retrospective

Some activities, such as Defect Triage, are tasks which must be attended to on a daily basis to keep the project on track. Others are one-time events, or occur periodically at key intervals along the sprint timeline, for example Backlog Grooming. In our team's 2-week sprint timeline, our QA tasks are scheduled something like this (each team makes adjustments to best suit their needs):

Week 1				
Day 1	Day 2	Day 3	Day 4	Day 5
Daily Standup ***Sprint Planning*** ***Test Plan Prep*** ***Submit Infra Requests*** ***Set Cross-Team Meetings*** ***Defect Triage***	*Daily Standup* *Cross Team Meetings* *Test Plan Prep/Create* *Defect Triage*	*Daily Standup* *Inform SM of Test Blockers* *Test Plan Prep/Create/Execute* *Backlog Grooming Prep* *Defect Triage*	*Daily Standup* *Backlog Grooming* *Test Plan Create/Execute* *Inform SM of Acceptance Blockers* *Defect Triage*	*Daily Standup* *Test Plan Create/Execute + Exploratory Testing* *Inform SM of Test Blockers* *Cross-Team Check-ins* *Defect Triage*
Week 2				
Day 6	Day 7	Day 8	Day9	Day 10
Daily Standup *Test Plan Create/Execute + Exploratory Testing* ***Inform SM of Blockers*** ***Defect Triage*** ***Defect Disposition Exercise***	*Daily Standup* *Test Plan Execute* *Cross Team Meetings* *Inform SM of Acceptance Blockers* *Defect Triage*	*Daily Standup* *Test Plan Execute* *Final Defect Disposition* *Defect Triage*	*Daily Standup* *Retrospective Prep* *Backlog Grooming* *Review Branch Merges* *Defect Triage*	*Daily Standup* *Defect Disposition Cleanup* *Sprint Demo* *Sprint Retrospective* *Defect Triage*

Figure 1: QA responsibilities across a 2-week sprint timeline

Figure 1 illustrates QA responsibilities each day throughout the sprint. It's important for the entire team to be familiar with these tasks, because many of them involve other team members engaging with QA.

Many of these tasks and activities are familiar to QA engineers coming from traditional methodologies, such as reviewing requirements, writing test plans, testing and logging defects. Others may be less familiar. Earlier we asked the question, "How does QA's role in the Agile Scrum Team enable us to release higher quality software?" We're going to explore that in detail. But before we do, let's take a moment to review some well-known QA activities not unique to any methodology. I refer to these as "the usual stuff".

3.1 The Usual Stuff

- Test plan creation
- Test execution
- Logging Defects

These are familiar to QA engineers within any methodology. And of course they're cornerstones of the work in Scrum as well. They absolutely have their place throughout the sprint schedule.

But this is stuff we know well. So let's look at what's unique to the QA Role in Scrum that takes us beyond "the usual stuff."

3.2 Backlog Grooming: Quality Starts with the User Story

The process of reviewing and refining user stories to prepare them for commitment in a sprint is called Backlog Grooming. Two to four backlog grooming sessions should be planned for during the course of the sprint, so that stories have been fully defined ("groomed") and are ready to be committed in the next sprint.

Scrum training discusses story writing and backlog grooming in depth, defining this as a Product Owner task, perhaps with some input from developers. So what is QA's role in backlog grooming? What we discovered is that QA's input is essential.

Backlog grooming is our first opportunity to prevent defects from being introduced into the code in the first place. In other methodologies, QA rarely has input into requirements definition. And we often have limited interaction with developers during implementation. The approach of, "throwing code over the fence to QA" makes it impossible for us to prevent the introduction of defects. We can only find and report on them after the fact.

In Agile, we have the opportunity to ask key questions during requirements definition. And we can ask technical questions of developers before they start implementing. This results in more thoroughly defined user stories. QA's questions often prompt developers to consider issues they hadn't previously thought about. They can resolve these with the first check-in rather than waiting for the question to be asked in the form of a defect report. This is prevention vs. detection in action.

3.2.1 The Importance of Acceptance Criteria

Acceptance Criteria are used to determine if the completed user story meets the team's "Definition of Done". A story isn't complete without them. While this isn't a presentation on how to write good user stories, it's important to discuss the significance of well-defined acceptance criteria.

It's not enough to have criteria like, "Works as designed", or "Works like it did before the change". Acceptance criteria should be specific to what's being implemented, and speak to the customer experience and value proposition. They should include system integration expectations where applicable, and negative use case handling. Well defined acceptance criteria ensure quality both by driving good testing, and by enabling accurate story sizing.

3.3 Sprint Planning for Success

3.3.1 Don't Forget the Defects

During sprint planning, the scrum team reviews the product backlog and decides which work can be committed to the sprint. But there's more work to plan for than just implementing user stories. We must also plan time for defect fixes. Our defect database contains defects found (but not fixed) during previous sprints. Like the product backlog, the defect database is another "backlog" which must be reviewed during sprint planning. The team must decide which known defects it can commit to fixing during the sprint.

3.3.2 Planning for the Unknown

In addition to known defects, we must also allocate time for fixing defects which are *not yet known* -- that is, the defects which will be found during the upcoming sprint. These defects may be related to the stories under development in the current sprint. Or they may be found during system integration or other testing which could only take place after an earlier sprint had been completed. In any case, each new defect must be evaluated during our daily triage. Some will likely need to be fixed as part of the current sprint. Neglecting to allocate time to fix unknown defects makes the sprint plan inaccurate.

3.3.3 Sizing: Story Points = Dev Time + Test Time

"Story Points" are units of measure used in Agile to reflect the amount of work needed to complete a user story. Agile refers to the estimation exercise as "story sizing". When teams do story sizing, they often forget to include *test time* in their overall estimation. (This is not unique to Agile.) More often overlooked is the *test-fix-retest cycle* that's necessary when a critical defect is found that prevents the story from being accepted. For our sprints to be successful, it's critical to include these in our sizing. Here's a guiding principle to ensure the best accuracy in story sizing:

Story Points = Dev Time + Test Time

Some engineering teams resist including test time in their story sizing. In our teams, once we added test time to our story sizing we saw a dramatic increase in the percentage of stories completed in a sprint.

3.3.4 Timing

When planning a sprint, avoid sprint-end bottleneck! Remember that the last step in accepting a finished story or defect fix is QA verification. If stories aren't complete until the end of the sprint, QA won't have time to do all the verification testing. When committing stories to the sprint, be sure to select stories which will be ready for QA verification throughout the sprint timeline, rather than all at the end.

3.3.5 All User Stories are Not Created Equal

Part of story sizing is recognizing that all stories are not created equal. When considering the lifecycle of a User Story, we tend to think Develop->Test->Accept -- a simplistic model that looks something like this:

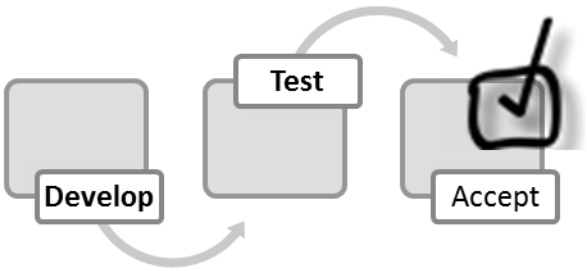

Figure 2: Simplistic model of user story lifecycle

How does this simplistic vision of story lifecycle impact our ability to plan our sprint realistically? Applying this model to multiple user stories in the context of the Sprint Timeline, our sprint plan might look something like this:

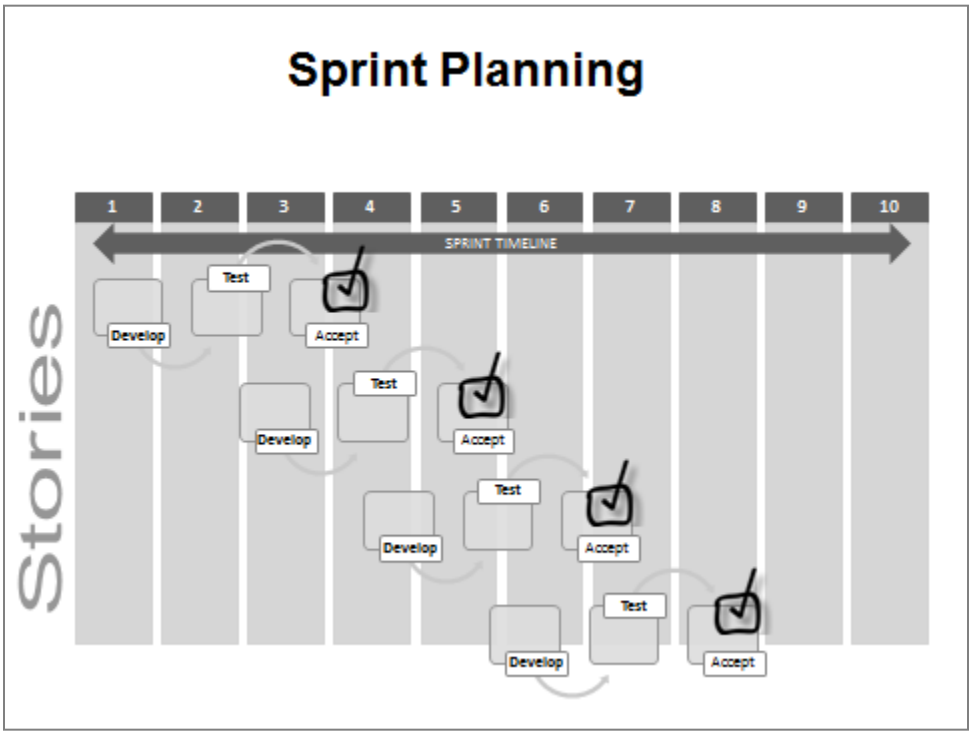

Figure 3: Simplistic model of user story lifecycle on a 2-week sprint timeline

Unfortunately, this straightforward model isn't a realistic representation of a sprint plan. What are some problems with this sprint plan?

- Dev and test times are represented as equal and predictable
- One task doesn't start until the next one is finished
- Possible test failure is not accounted for – no time is allocated for the unexpected.

Expecting our units of work to fall into this simplistic model isn't realistic, and sets us up for failure.

Instead, let's consider some more realistic examples of user story lifecycles:

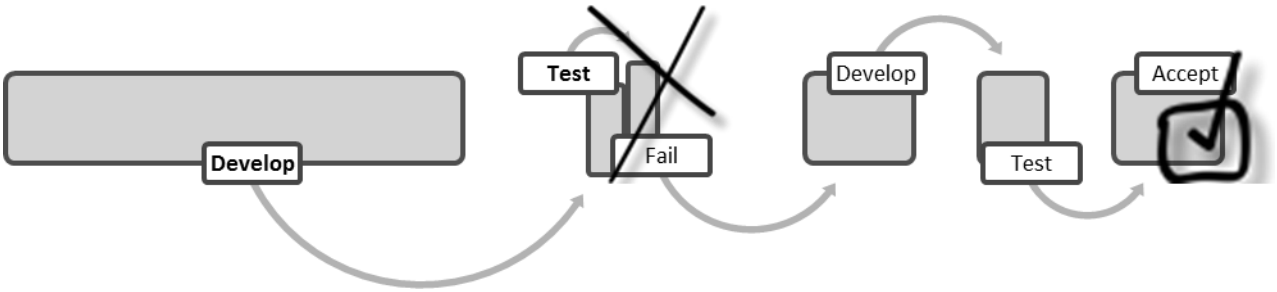

Figure 4: Example story lifecycle: long development time, short test time.

In this example, we have a story in which development times are relatively long, and test times are relatively short. This might be the case when a large refactoring is called for, which takes significant development time. Test time would be relatively short, however, because no additional work is required on the existing test automation needed to test it.

Notice also in this example, we've allowed for the possibility of acceptance test failure, and the need to rework and retest the code.

Another example illustrates the opposite condition: a story in which development times are relatively short, but test times are relatively long.

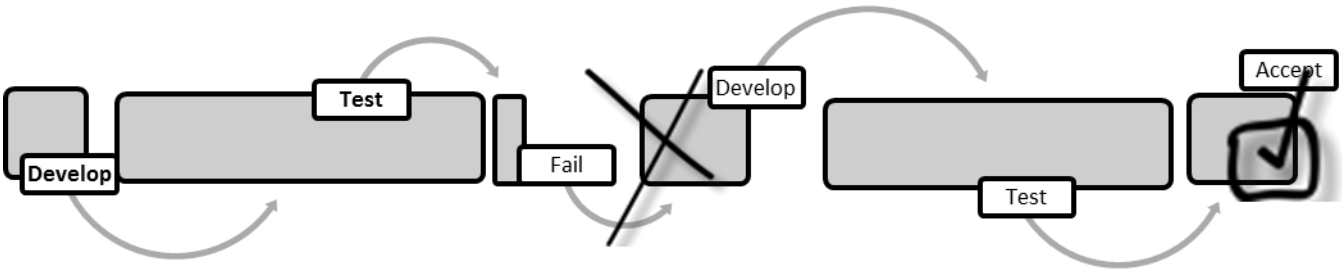

Figure 5: Example story lifecycle: short development time, long test time.

3.3.6 Putting it all Together

Now that we understand that Story Points = Dev Time + Test Time, and that not all stories are created equal, putting all the units of work (user stories and defect fixes) together into the sprint plan, the reality looks more like this:

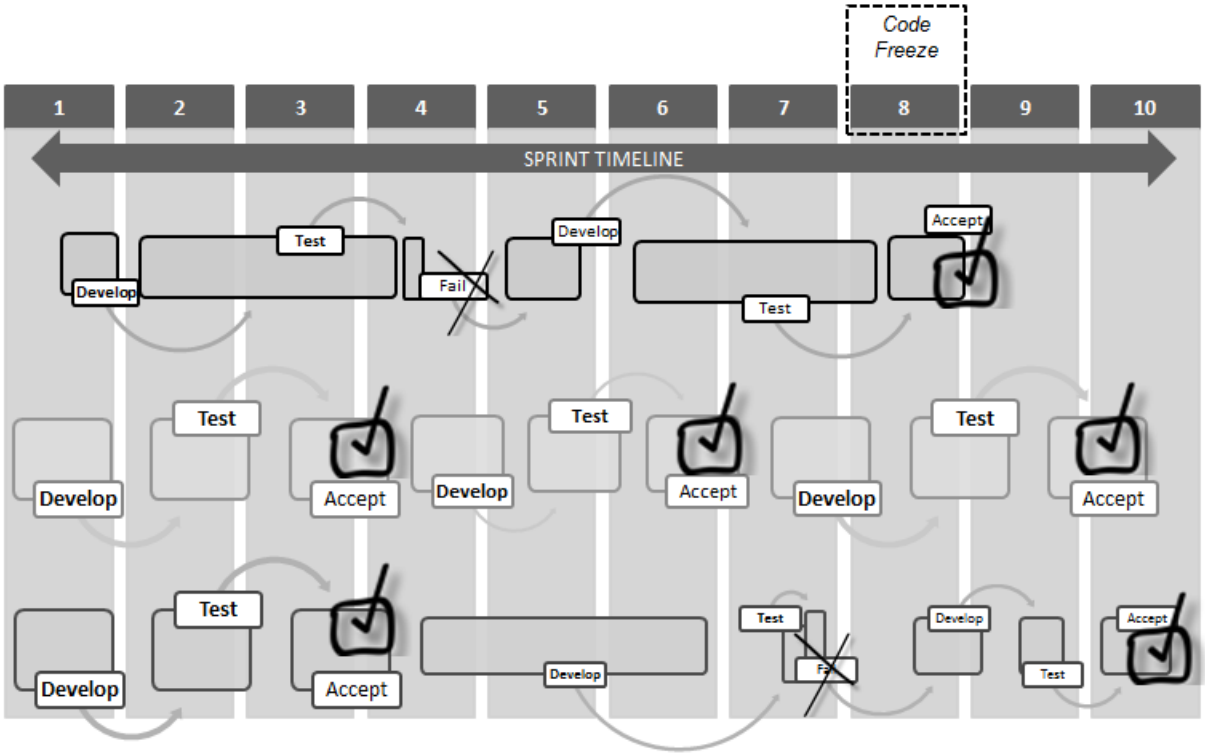

Figure 6: Planning stories in the sprint

In this example we see some effective planning, and some unknowns accounted for:

- Stories are represented as having varying lifecycles, some demanding more or less dev and test times than others
- Time is allocated for potential test failure, requiring more dev and test time

But what are some of the planning considerations we've discussed which are not accounted for in this sprint plan?

- Expectation for first testable code comes too early in the sprint
- Too much code delivery at the end of the sprint – creates a bottleneck in QA
- No time allocated for necessary sprint wrap-up activities including code freeze, release branch check-in, release notes creation, final defect disposition, sprint retrospective and demo.

How do we fit everything in? A sprint plan which is set up for success will look more like this:

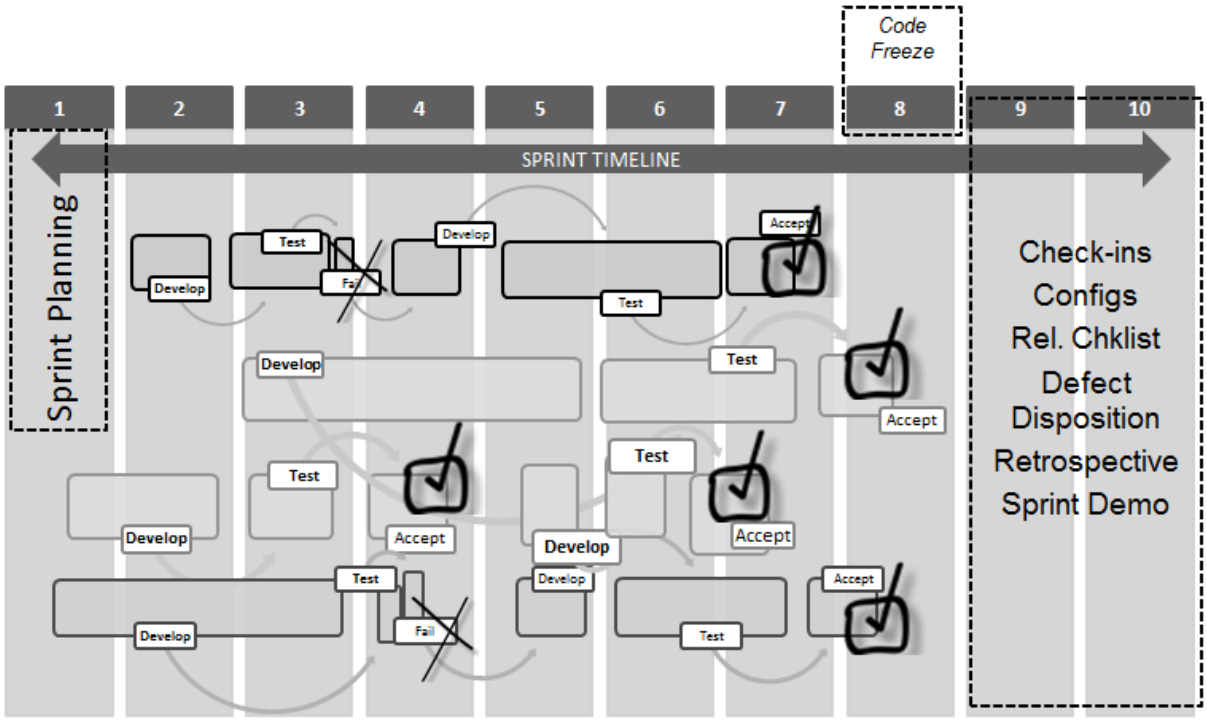

Figure 7: Sprint planning for success

This example allocates time on Day 1 for Sprint Planning. It's key that the entire team take time together to create the sprint plan. This plan looks complicated, but it works, because it accounts for all work needed to be successful, not just development and test.

This plan has one less story than the last. But it's well positioned to deliver 100% of the work committed. The previous example was not. The more granular your stories are, generally the more you can commit to in a sprint. Here we're less concerned with quantity; the emphasis is delivering quality, and completing 100% of the work committed. Here's why this plan is positioned for success:

- Time is allocated on Day 1 for sprint planning
- Test-ready code is not expected until Day 3, some later
- Story sizing is carefully planned, and includes both dev and test time
- Time is allocated for acceptance testing before Code Freeze
- Time is allocated for sprint-wrap up work

3.4 What about System Integration Testing?

Thus far, we've been talking about sprint planning in the context of only one Scrum Team. But what about scaled Agile, where multiple Scrum Teams deliver code into the same code base? Many of our user stories have dependencies and integration points with stories being developed by another team.

Within the scrum we focus our testing on the specific stories and defect fixes being worked on by our team. This work has a tendency to become "silo'd": our intense focus on the work of our own team results in loss of visibility to what other teams are working on. How do we prevent testing gaps across the full system integration? We've addressed this by setting aside time for **Cross-Team Test Planning**.

To close integration testing gaps, QA leads from different scrum teams need regular communication. They need to understand stories being worked on in other teams, so that together they can plan testing which ensures coverage of all integration points. This also ensures dependencies are accounted for in acceptance criteria and test plans. These meetings should occur weekly. Tuesday of each week works well for our team. By the first Tuesday of the sprint, sprint planning will have been completed. Testers will have reviewed all the stories and are prepared to share integration and dependency information with other teams. By the second Tuesday, having completed some work, testers now should have increased insights to share with each other.

3.5 Defect Management

There are three critical steps in defect lifecycle which need to be managed throughout the sprint: Triage, Fix Verification, and Final Disposition.

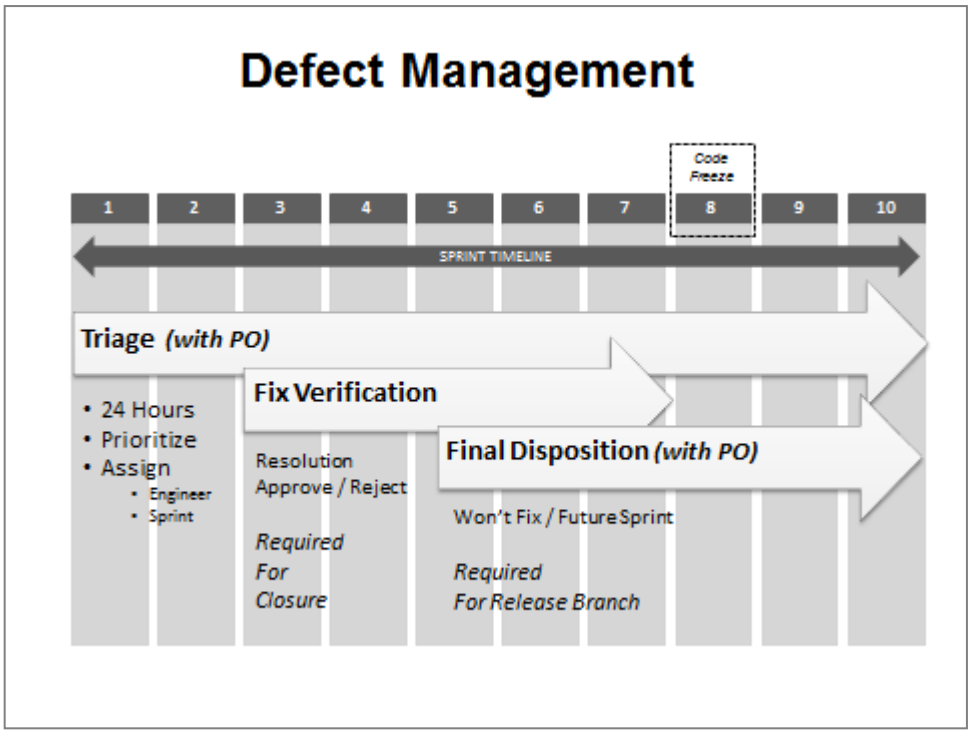

Figure 8: Defect management tasks along the sprint timeline

3.5.1 Triage

Defect triage involves reviewing newly submitting defects, verifying the defect is reproducible, determining priority, target sprint, and development ownership. In our team, Product Owners are responsible for prioritizing units of work. This includes prioritization of user stories as well as defect fixes. Product Owners and QA perform triage together. In a two-week sprint cycle, defect triage must happen daily. Many teams I've worked with balk at the idea of a daily triage. They feel it's too time consuming. But consider the alternative. If at sprint's end we find that there's an outstanding defect that must not go to production, we have to make a choice: either disrupt the release cycle by fixing urgently, rebuilding, and

redeploying the release branch, or don't merge this code into the release branch at all. We hope never to be faced with this choice at the end of the sprint, but it does happen. The way to minimize this is by holding daily defect triage sessions so that everyone is keenly aware of all the existing defects throughout the sprint cycle.

3.5.2 Fix Verification

As soon as a defect is resolved by a developer, QA is responsible for verifying the resolution, whether it's a fix, or another resolution such as "can't reproduce", "not a defect", "will never fix", etc. As with story acceptance testing, it's critical that QA provide quick turnaround on defect fix verification. Developers need to learn as soon as possible that a defect fix didn't really work, or that the issue they thought was invalid really is a defect. Receiving the information when the issue is still fresh in the developer's mind ensures a faster turnaround in the event that more work is needed. QA's approval of the final resolution is required before the defect can be closed.

3.5.3 Final Disposition

In our organization, the term "disposition" is a verb. To "disposition" a defect is to decide when it will be fixed, or that it will not be fixed at all. One of our criteria for code merging into the release branch is that there are no outstanding defects assigned to the current sprint. Earlier we talked about the two kinds of defects to consider during sprint planning: known, and unknown. As the sprint comes to a close, there will be a number of defects falling into that second category: they were unknown – didn't exist – at the start of the sprint. These may have been reported near the end of the sprint, in which case they are likely still open and assigned to the current sprint. However, there's no time to fix them in the current sprint. Final disposition involves reviewing all defects which are still open at the end of the sprint, and making sure each is a) acceptable to be released, and b) targeted to be fixed in a future release. Without final disposition, open defects remain assigned to the completed sprint, and get overlooked as we move to the next sprint.

3.6 Story Acceptance

Story Acceptance is the process of determining whether a user story meets the Definition of Done. Our team's Definition of Done is this:

1. Meets Acceptance Criteria
2. Passes Acceptance Tests
    - This could be viewed as part of #1. We found the need to explicitly call it out so that QA would be an active participant in the story sign off process.
3. No outstanding Priority 1 or Priority 2 defects against the story.
    - Why allow for *any* outstanding defects against the story? Our testers find all kinds of defects, many of which, while relevant, don't warrant rejecting the story. Examples include minor cosmetic defects unique to non-priority browser versions, or minor issues with code which won't be visible to the user yet but is needed by a dependent team for further work in the next sprint.

QA should be prepared to perform story acceptance testing as soon as a story is complete. Just as with defect resolution verification, developers need immediate feedback as to whether or not their story is accepted. This enables them to either address any outstanding issues immediately, or turn their full attention to the next backlog item.

3.7 Sprint Wrap-Up

We talked earlier about the need for QA to have a strong voice in “non-traditional” aspects of their role, such as backlog grooming. Our voice is also needed in sprint wrap-up activities.

3.7.1 Sprint Retrospective

The sprint retrospective is an opportunity for each team member to voice ideas for improvement. QA should prepare feedback for the team about what went well, what didn’t go well, and what can be done better in the next sprint to further the goal of preventing, rather than detecting defects.

3.7.2 Sprint Demo

The sprint demo is the time for the team to show off the great work they’ve accomplished during the sprint. Generally this takes the form of a software demo, or maybe even a brief code review highlighting a sophisticated solution to a complex implementation. Development accomplishments are easy to show because they’re ready for production release. If QA has done its job well, which we assume it has, the software performs brilliantly and the demo goes off without a hitch. QA tends to remain the quiet observer. While the audience is wowed by the software being shown, they don’t think to say, “Hey, no bugs! Great work, QA!” Rather, they just expect it to work. So what does QA have to offer in a sprint demo? We think it’s important to use the sprint demo to showcase QA’s contribution. We do this by allotting time for QA to either demonstrate or talk through a testing accomplishment. Some team members have demonstrated test automation in action. Others have presented a brief automation code review, demonstrating innovation which makes the automation faster, or more extensible. Still others have talked through a strategy for test coverage of a critical new feature. This raises everyone’s awareness of the important role QA plays in the scrum team. And it increases confidence in the quality of what’s about to be released.

4. This Really Works!

By implementing the core concepts and processes described here, our team has delivered measurable quality improvements consistently over multiple releases. Having implemented just some these processes, in less than six months we successfully transitioned from quarterly to 2-week release cycles. We reduced overall defect count and deploy time, and achieved a 95% reduction in production hotfixes. Everyone finally stopped working weekends!

5. Conclusion

As the transition to Agile continues its rapid growth across the globe, it’s essential to fully define and understand the role of QA in Scrum. Software organizations depend on us for success.

I hope that this paper has provided you with insights, information, and ideas that you can take back to your own scrum teams to add to your continuous improvement process. Share with your teammates the important voice QA must have in sizing units of work during sprint planning, defining acceptance criteria, managing the defect lifecycle, communicating dependencies across teams, and actively contributing to all scrum ceremonies.

Although Agile has been in full swing for over a decade, there still remains an exciting opportunity for us to further the success or our organizations by bringing clarity to the role of QA within the Scrum Team.