Last year our team at HP transitioned to Agile.
A lot of people asked me what QA does in Scrum.
It was a great question! And I quickly discovered my we weren’t the only one asking it!
It turns out there wasn’t much information out there to help us define QA role in scrum, so we spent the early part of last year developing the role to best suit our needs.
What IS the QA role in Scrum?
And how can we leverage Agile to prevent defects?
Let's start by talking about Traditional QA role and activities. We test software, and we report defects. In this model, our ability to ensure high quality software is limited. Why? because we're scrambling to fix problems instead of preventing them in the first place.
Let’s think for a minute about how we keep our *selves* healthy. We’re told to eat right, exercise, wash our hands a lot, and get enough sleep. This is to PREVENT disease, because preventing illness keeps us much healthier and more productive than if we wait to fight illness until after we’re already sick.
This is more expensive, and painful.

So why don’t we take the same approach to software quality?
Let's PREVENT defects from entering the system in the first place
Rather than chasing them down after the fact

How can we leverage Agile to do that?
First, let’s start with the fundamental Agile premise that delivery, and therefore Quality delivery, is the responsibility of the entire scrum team. As part of the team, QA is involved from the beginning. No more “throwing the code over the fence.” For example, we’ve all seen defects resulting from unclear requirements. In a Scrum team, QA can have requirements clarified before implementation begins. This is a great first step toward defect prevention. But this is just one example...
Let’s get into the day to day of QA role in scrum.
What does QA do??
Our role does include the traditional activities:
• Test planning
• Running tests
• Logging defects
But there’s a whole lot more!
We want QA engaged in all kinds of activities throughout the sprint, that will help reduce the introduction of defects. Prevention, vs. Detection.

These are some of the responsibilities that our QA engineers are involved with throughout our sprints. The heat map represents frequency:
Some are daily activities.
Others are one-time events, or periodic at key intervals along the sprint timeline.

We talked about the traditional stuff QA does, and those play a big part as we can see in this heat map. We’ll spend the rest of the discussion talking about the other stuff: the key things we can leverage from Scrum to really help us prevent software defects.

We’re going to use the example of a 2-week sprint cycle. Laid out across the sprint timeline, these activities might be distributed something like this:
Now, this is just an overview of one example sprint timeline. I don’t want you to try to read through this and interpret every item here. What I want you to see from this view is how the many activities we saw in the heat map might be distributed throughout a sprint.
The first thing we’re going to dive in to is Sprint Planning.
The purpose of sprint planning is to review the Story Backlog to decide which stories the team can commit to completing in the next sprint. The amount of work is represented in “Story Points”.

A common mistake we see is that teams equate Story Points only to Dev Effort. Did you know that some teams don’t even include their QA in Sprint Planning? This is a mistake…

How can we accurately estimate the amount of total effort without factoring in test effort?

Don’t forget the defects…
Remember in Sprint Planning, as with User Stories, Defect Fixes are important backlog items which must be considered for commitment in the Sprint. Make sure your team gives as much attention in sprint planning to the defect backlog as they do to stories.

Finally, Plan for the unknown…
In addition to the defects we know about when planning begins, there will be defects introduced during the sprint, which we refer to as “unknown defects” at the time of sprint planning. Story points should be added to account for Unknown Defects.
While this isn’t a presentation on writing good user stories, I do want to take a moment to emphasize the importance of Acceptance Criteria in defect prevention.

Acceptance Criteria are the criteria by which we determine if the story meets the team’s Definition of Done. We’ll base our test cases on these, so we need to be very thorough about what those criteria are. It’s not enough to say, “works as designed”, or “works like it did before the change”. We need criteria that are
- very specific to what is being implemented,
- speak to the customer experience and value add,
- and we also need criteria that include integration verification, and negative scenarios.

QA must have a strong voice in advocating proper, precise, and comprehensive Acceptance Criteria for every story.

Ask Questions about expected behavior.
Don’t commit to any story unless you understand how you will test it.
Let’s talk about User Stories as Units of Work.

When we think of the lifecycle of a User Story, we tend to think Develop/Test/Accept, using a simplistic model that looks something like this.

When we view several stories, then, in the context of the Sprint Timeline, it would look something like this. Nice, tidy, and predictable.

But this is an “ideal world” model. Now I’m going to ask you to help me out here: WHAT DO YOU SEE IS MISSING HERE?

- Dev and test times which are equal and predictable
- One task doesn’t start until the next one is finished
- !! Failure is not an option (notice all the tests are passing and stories accepted)

Thinking of our stories and defects as units of work in this simplistic model isn’t realistic, and sets us up for failure.
All User Stories are not Created Equal

We’ve learned that all user stories are not created equal! <click>

Instead, let’s consider some real-world examples of story lifecycles.

In the real-world, we have stories that look more like this… In this example we have a story for which

Dev times may be long,

test times may be short
  For Example: long time to develop, but easily tested by existing automation.

Notice in this example also that we’ve allowed for the possibility of something very important: Acceptance test failure, and the need to rework and retest the code.
In this example, the opposite is true.

Dev time may be short

but test time is long

For Example:

- Long test setup procedure;
- long manual test workflow;
- shorter workflow, but multiple browser testing required
Putting all the stories together in to the sprint plan, the reality looks more like this:

- Some stories will take more time and resources than others
- One resource will be needed for multiple tasks
- Acceptance Tests will fail, requiring more dev and test time
- New defects will be found which must be fixed

This represents a sprint plan which takes into account that “not all stories are created equal”.

But let me ask you again: what are some problems that still remain in this plan?

- ! Don’t deliver all testable code at once at the end
- ! No time allocated for cleanup

- ! When will the code be committed and frozen? <click>
- ! What about sprint planning and other sprint wrap-up activities?
With some careful Sprint Planning, we can create a plan which looks more like this. It looks complicated, but for a number of reasons, it’s better.

Notice, in this plan,
- there are 7 Stories instead of 6 (smaller, more granular)
- Dev and test times vary but are shorter (clearer, more granular stories)
- Dev tasks carefully staggered
- Acceptance testing planned to be complete by Code Freeze
- <click> Leaves time for the unexpected
- <click> And it leaves us time for sprint-wrap up work.

And accounts for Sprint Planning on day 1.
We’ve been talking about sprint planning exclusively in the context of User Stories for one Scrum Team. But what about system integration testing? If your group is like ours, you have multiple scrum teams delivering work into an integrated, enterprise solution. Many of our teams’ user stories have integration points with stories being developed by another team. Within the Scrum, How do we prevent defect introduction, and ensure quality, across the full system integration?
We’ve addressed this in our team by having a weekly cross-team test planning meeting.
This is a meeting of All QA from every scrum team
People bring user stories which include integration points with other scrums to discuss with the entire QA group.
It’s amazing how much we’ve learned about what we didn’t know:
- What other teams are working on
- How their work ties in with our own
- What the end user experience is supposed to be
By learning these, everyone is more informed in what they bring to their own test planning, and to the holistic approach to quality and defect prevention.
Going back to our heat map of responsibilities…
We’re going to talk about a few more areas where QA plays a key role in holistic quality and defect prevention, and *when* in the sprint cycle these occur.
While we’re striving for defect prevention, there will be defects 😊 And we need to constantly manage them.

Again in the context of the 10-day sprint cycle, There are three main aspects of Defect Management that the team is responsible for.

- Triage
- Fix Verification
- And Final Disposition
When do we do Story Acceptance?
Pretty much need to drop everything and do story acceptance testing as soon as
the story is complete.
Why? Because developers need immediate feedback as to whether or not their
story meets the team’s Definition of Done
Earlier we talked about the importance of Acceptance Criteria in our User Stories.

Backlog grooming is the process of preparing stories to be committed in a sprint. QA has a critical role in Backlog Grooming. We need to:
- Ask questions to help better define the story, and
- Advocate for comprehensive Acceptance Criteria.

In this example we show weekly backlog grooming every Thursday. This is probably the minimum amount of time you need to ensure stories are ready to be committed in the next sprint.

Backlog Grooming is driven by the Product Owner. If you’re not currently participating in your team’s backlog grooming sessions, tell your Product Owner you need to be included.
And finally, Sprint Wrap Up Ceremonies: Sprint Demo and Retrospective.

What’s QA’s Role in the Sprint Demo?
• Answer questions like, “how did we test this?”, “what was the end to end test coverage?”
• Some teams will have test automation achievements to highlight as part of demo
  • Some automation is even interesting to demo

What’s QA’s Role in Sprint Retrospective?
• Like the entire team, discuss what went well, what didn’t, and how the team can do better next sprint
By implementing the 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,
- reduced overall defect count and deploy time,
- and had a 90% reduction in production hotfixes.
Everyone finally stopped working weekends!
If you take away one thing from this presentation, please make it this:
Story Points = Dev Effort + Test Effort
Keep this as the cornerstone of your sprint planning, and the rest will fall into place.
Hopefully, the material here 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 all of these things. Be the voice of Quality!

### QA Role in Scrum

**Be the Voice of Quality!**

- *Remind the team: Story Points = Dev Effort + Test Effort*
- *Groom backlog stories*
- *Define acceptance criteria*
- *Manage the defect lifecycle*
- *Communicate integration dependencies across teams*
- *Contribute in all Scrum ceremonies*
Thank You