Lessons Learned in Agile Testing

by Rajneesh Namta

Recently, my colleague and I presented at Agile NCR (Gurgaon, India). In this presentation, we talked about our experiences of working as QA or testers on Agile projects (offshore) in India over a considerable period of time. While on these projects, we continue to learn and daily gain some new insights about our work, methodology and the people we work with.

I had a feeling that the slides in our presentation may not have been sufficient to get across the message we were trying to convey, and hence this article. The intent of this article is to reach a wider audience and share these lessons with the community.

As we uncover better ways of developing software, so are we finding better ways to test it. One of the best things about Agile is that everyone on the team (developers, testers, architects, analysts et al) starts right from project initiation, and if this is not the case (which may be possible with some organizations), it is still the preferred route. It helps greatly to be a part of the team from day one, since as a tester you get the lead time to get acquainted with the infrastructure and technology, understand team dynamics and initiate a customer dialog to get an insight into the business at a very early stage. This is something which will pay off in the longer run, as all this is very crucial for a tester to contribute effectively when the actual action takes place. A team with people of mixed skills right from the start will add a lot of value rather than a team of people with a very specific skill. This is something which rarely happens in a traditional framework, where people are generally added and taken off on need basis. Interestingly, slowly and steadily the realization has dawned on many that they can create more value for their customers while maintaining a consistent level of quality in each sprint by having people with mixed skills. It also brings in a powerful change of mindset which enables everyone to see the tester as an integral part of the core team rather than someone from a different planet talking in an alien language, as he is not in sync with the project reality.

Agile not only helps create better software, but probably better professionals as well. Here are some lessons that we have learned while transitioning from a traditional to an agile way of working.

One Team - One Goal

The concept of “One Team” is a very powerful one. It entails a complete change of mindset, which is very refreshing as well as rewarding. The team sitting in one place at one table is one such example, where the concept is actually realized, as physical barriers are removed. Conventionally, test teams or the independent verification and validation units (some fancy names in organizations for the test team) are separated from the development teams and usually sit in separate cubicles, on a different floor or in different buildings. This leads to obvious communication barriers, but more importantly it breeds a mentality of us versus them in the team. Fault finding and blame game ensue, as people look down upon each other and end up facing each other and blocking progress. A team of ‘comrade in arms’ rather than ‘adversaries’ is much more likely to succeed, and that’s what “One Team” means. This also means that the QA guy is no longer the “Quality Police” on the project, and the whole team owns quality resulting in better quality products.

The team should be like a well oiled machine, where all individual parts work in unison to achieve the goal. Every success is a team success, and each problem is a team problem in such a set-up. Team members take collective decisions and ownership for the work they do.

Have a Test Strategy

One of the key questions a tester often encounters when he/she transitions from a traditional to an Agile way of working is whether to create the heavy-weight test plan, strategy and other similar documents or not . These documents are given way too much importance in a process-oriented set-up and are pre-requisite to start any testing activity. In contrast, Agile focuses on ‘just enough’ documentation and is often misunderstood for no

documentation. Eisenhower once said '**plans are useless but planning is indispensable**' and that's the key while planning in an agile project.

As a tester you should possess a high level of clarity about the whole testing activity. A test strategy will help the whole team to see the testing activity clearly and enable them to contribute in fine- tuning it. You will get brilliant ideas from the team, as everyone would be interested in having a test process in place which is helping the final cause. It will give the tester a clear path to take when testing during a sprint or a release.

Test strategy should contain details on a high level, a plan for a release, for example, with things like testing techniques and tools to be used, automation of regression tests, testing any interaction with third-party tools and interfaces, database testing among others. There is no set template for a test strategy, and it can be anything: a piece of paper. a Wiki page, a text document, a diagram or an email detailing your approach. The only important thing is that you have a strategy and it's communicated to everyone in the team. Extensive documentation should only be created when you are able to keep it up-to-date, otherwise it will soon go stale.

Involve Customers in the Test Process

No one in the team has the kind of insight and domain knowledge that a customer or the end user of the product has. Involving the customer in the test process will increase the efficacy of the testing activity. To achieve this, you will have to create a transparent and trustworthy relationship and initiate the right kind of dialog. Nowadays, many BDD and ATDD tools are available and gaining popularity; these specify the tests in a domain language, which can be easily understood and learned by customers. By using these tools, the users can go one step ahead and design or write test cases for requirements themselves, which would serve as acceptance criteria.

There are numerous ways in which the customer can contribute to the testing activity within or outside a sprint. For data-centric applications or in fact for any application, only the customer can provide the actual production data, which is a critical requirement for testing. Testing the application with the right kind of data is key to uncovering the defects, which may otherwise only be found once the software goes into production.

Additionally, the customer can provide actual usage scenarios and other requirements, such as performance benchmarks, early in the release cycle, which ultimately will translate into a more usable, fast and stable product.

Have a Definition of Done in Place

Something which bothers testers frequently is how to make a decision to stop testing and ensure that the user story or feature under test has been tested adequately. Theoretically, testers are never done, but they need to make a decision at some point to stop testing what they are testing and take up new tasks. One possible approach would be to do risk-based testing within a sprint focusing more on critical items and ensuring that they have not introduced any regressions. Additionally, it would make sense to have a checklist in place, which testers can refer to when moving a task from 'Test' to 'Done,' so as to reduce the risk of missing anything due to plain oversight or any resource crunch.

The definition of done, or the DoD, is negotiated between the team and the stakeholders as a generic set of guidelines of when to consider the user stories done for sprints in a release. A tester should also create a DoD based on the sprint/release goals and his testing objective for the project based on discussions with the team and the stakeholders. Once the criteria for done is decided, it can be added to the DoD for the project. This will make it visible to everyone involved and act as a guiding principle and reference to the tester. A DoD for the tester can be something like the following:

- Required functionality as described in the user story is implemented.
- Test data and cases are documented (automated in BDD tool) or on a Wiki (confluence).
- The implementation has passed the functional tests.
- All automated regression tests are green.

Clarify Specifications Using Examples:

It is good practice to give concrete examples when asking something from the Product Owner, developer or the user. A query asked in plain language or a lengthy email might not get the response you need, but seeing a real example will definitely evoke positive reaction.

This helps tremendously if you work on offshore projects in a distributed mode and need to clarify or disambiguate requirements with the customer. Since the customer will be geographically located elsewhere, it's very important to keep the feedback cycle fast and short, as sprint cycles are usually short as well. Additionally, the user might himself come up with alternative scenarios using examples, which can help to disambiguate requirements.

An example of such a case could be a project, where you are testing a finance application in which complex calculations are done. It would be good idea to create a small spreadsheet with calculations for different scenarios and ask for any clarification citing specific examples on the sheet.

Test as per the Context of the Sprint

Testing inherently remains the same in Agile, only the way it's applied in an agile project is different. Alternatively, we can say that the rules of the game remain the same, but it's altogether a different playing field. A tester in an agile framework would be flexible enough to allow changes to a set plan (responding to change over following a plan). Hence, to test as per the context of a sprint would be a logical and wise choice. It's very important that the tester remains aware of the context and constantly makes adjustments in his/her strategy to accommodate change.

At the start of a sprint, the tester can bring in his unique perspective and do requirements testing and story exploration along

with automating left-overs from previous sprints to add to the regression test suite. As a sprint progresses, the tester can test the features being coded (ideally, it would be faster to do it manually the first time) using exploratory and other methods, which would result in the most effective utilization of time and effort. As the sprint nears completion, the focus should shift toward testing end-to-end workflows and regression testing to ensure everything that worked before still works. Generally, workflow and regression tests are automated and should not take much time to execute. Hence it's very important to first figure out the context of a sprint and then test accordingly.

Test Automation is a Team Effort

Test automation is not only about an automatic execution of the test cases. It has a much wider application like integrating automatic tests with the build process, integrating toolsets/frameworks developed to test different components of the application, automatic reporting and notification mechanisms etc.

A tester might be involved in creating and maintaining most of the test artefacts, but he needs the help of the entire team to keep it going. Some situations might demand an in-depth technical know-how (like mocking some external interfaces, or making disparate tools talk to each other), which a tester might generally lack, and hence it's all the more important that the team is there to support you. It's again an offshoot of the "One Team" concept, where every problem is a team problem.

It also gives the team (especially developers) a lot of confidence before making any change if a robust framework is in place that they can rely on. So it's for the benefit of the whole team and the whole team owns it. A tester might take the ownership of maintaining and keeping it relevant in the long run, but not without the commitment of the team.

Provide Fast and Quality Feedback

Faster feedback is the very essence of agile development. Having automatic checks in place (like CI, automated unit testing and regression testing) ensure that feedback is instantaneous. Taking a cue from such practices, design your functional tests such that they can be integrated into the build. If some tests slow down the build, abstract them out#put them into inside a separate suite and schedule them to run overnight with some notification mechanism. There is no point in creating suites or tests that keep on running for days, as delayed feedback would slow down the entire chain and hamper the speed and productivity of the team.

Additionally you should not wait for a bug to be logged and go through the complete lifecycle in the bug tracking system before it's fixed and re-verified. As soon as an issue is found, it's good to announce it (write it on the whiteboard, instantly message the developer concerned, or just shout!).Get it fixed there and then.

Quality feedback means that everything (bug report, incident log or a test report) that a tester provides for the consumption of the other team members should be so precise and refined that only a cursory glance through it should be enough to get an idea about what problem was found where in the system. It should be kept simple (apply the KISS principle here), yet should include every possible indicator that allows the developer to debug the problem quickly and efficiently. Test/defect reports (automatic or otherwise) should be smart enough that only a glance through them is enough to know about the failures that have happened and their probable cause.

Regularly Reassess your Test Process

It's very important to reassess the test process, since only then it will be of relevance. The software which is being created grows in size and complexity in every sprint. Similarly, the test assets and artefacts also grow, both in number and complexity. The maintenance nightmare sets in as the release cycle progresses and relevance of automated regression tests and other test artefacts becomes a major issue. A strategy that worked yesterday may not be relevant anymore, as there have been continuous changes. Hence, the testers need to re-visit and rethink their test process to keep it relevant and up-to-date.

A tester must be smart enough to try and see beyond a sprint and strategize accordingly. The release backlog is always accessible, and he/she should learn to make good use of it. While automating and creating a test strategy, the tester should always factor in the type and complexity of stories which are further downstream, so as to minimize the rework when it's time to actually implement them. A tester should be evaluating and revisiting the test design every sprint to ensure that he/she is on the right track. Evolutionary test design should happen in parallel to the evolutionary design of the system.

Explore, Learn, Innovate and Improve Constantly

Agile is not only about working on any one project. It's a continuous learning process, where people constantly enhance existing skills and add new ones. As a tester, you add more value to yourself, your organization and your customer by constantly learning and exploring new things. You will learn new tools and techniques which will not only help you to work more efficiently and better in the current project, but will stay with you long after the project is over.

Try something new within a project and introduce new toolsets or a working methodology to improve the current state of affairs (say a quick POC). This will probably help you to make things better, and even if do not and fail quickly, you know something which didn't work.

Maybe you would also like to share your knowledge and expertise with others and give something back to the community and contribute for example to open source or other initiatives. Blogging, writing papers, participating in conferences and engaging in discussions with the community at various forums will not only add to your knowledge and skill, but will make you visible to the community as well.

To conclude, it's great fun and highly rewarding to be part of an agile team, since as a tester you not only contribute significantly to the entire lifecycle, but you also improve as a person and professional. A tester feels more valued and empowered, when he/

she is heard and consulted for each activity.

Testers that are part of the core team no longer have to take over the role of 'sentinels of quality 'or the 'last line of defence', as everyone on the team is committed to build the right product. Testers help in creating and maintaining the safety net, which enables the developers to accommodate and make changes with confidence. Testers collect the information emitted by various signals put in place, and present that to the stakeholders so that they can make informed decisions about the software being built. Above all, testers question the software to know more about it, and good questions can only originate in an agile mind. ■

I would like thank Harsh Saini (my colleague and friend) for the thought-provoking and stimulating discussions I had with him, which translated into the original presentation and ultimately led to this article.

> About the author

Rajneesh Namta
is a Senior Test Consultant at Xebia India. He is a passionate tester and has now worked for almost 7 years in various roles in QA. He is certified Scrum Master and has been practicing Agile for the last 2 years now.
He has worked across different stages of software development, including requirements specification, user acceptance testing and post-production training. Rajneesh is passionate about software quality and related tools and techniques and has made consistent efforts to improve the existing way of working, not only for himself but for the teams and organizations he worked for. He frequently blogs about software testing on his company blog and recently presented at Agile NCR (Gurgaon, India) a talk titled ‚Lessons learned in Agile Testing', which was very well received by the audience. This talk is the idea behind his article.