The quality assurance journey - HP case study

Roy Nuriel - HP Software R&D

December 11, 2013

Abstract

In the past several years we have seen more and more organizations taking the decision to move their development divisions towards adopting agile methodologies. In most cases the change starts with a Proof of Concept (POC) of a small new project that can validate the ability of the organization to make the shift to Agile. In many cases the development team takes the lead: changing the process, moving to unified teams, selecting which Agile practice to adopt, tool selection, etc.

As an R&D group that develops an agile project management solution – HP Agile Manager – we wanted to do it right. We changed the way in which we develop software from Waterfall to Agile, and built a process to support the teams in a large and complex enterprise. While previously we were accustomed to delivering releases in yearly or two-yearly cycles, we now operate within a Software-as-a-Service (SaaS) model where we update our production environment every two weeks.

We understood that we needed to review all our processes, tools, organizational structure, and – above all – our culture. Following I will describe how we made the shift, focusing on the change in our quality process.

Having experimented with the same shift from Waterfall to Agile that our customers are going through, we changed the way in which our QA engineers work. In accordance with their new role, we gave them a new title – DevTesters.

Our mission, as an R&D organization, was to deliver new functionality at high quality every four weeks, divided into two sprints. On the one hand, we had to understand how to increase the quality of our product,

while on the other hand dramatically reducing the time available for our quality processes from a few months' stabilization period to a few days, to validate that the new functionality works as expected, and that no regressions were introduced.

Here are some of the changes that we implemented:

- **Cultural change.** The first step was to understand that our mindset needed to change. The quality task is no longer QA's problem alone. Rather, it is a goal for the entire development group to strive for. Work on new features cannot begin until the high quality level that we defined is achieved This means that developers need to be active in the quality process and in the tasks that it involves. As for the testers, they were used to receiving a new delivery of the software only when it was stable and "Ready for QA". At this point they could validate new content, usually via the UI, and drive end-to-end scenarios. In the new reality our DevTester has to work much closer with the developers and "play" with new capabilities even if they are only half-baked. This was the only way we could find the issues and fix them in short cycles.

- **Organizational change.** The cultural change was described above impacted the way our organization is structured. The development and the QA departments used to be two separate organizations. Under the old model, the QA department used to be the "bad cop". With the move to agile, we moved to feature teams, where each team includes developers as well as DevTesters. Product managers also provide services to several teams. The feature team lead is responsible for delivering a certain number of user stories during the sprint at the level of quality defined by the release manager. A backlog item cannot be set to "Done" if it hasn't gone through the defined quality process. QA is still maintained as an autonomous organization, with QA testers and managers, making sure that they have all the tools and skills in place to validate quality.

Offshore – We have found that in most cases it is extremely problematic to work in globally distributed feature teams., We aim to co-locate our teams in the same geography in order to

increase collaboration. Where the teams are split due to constraints, we invest a lot of effort to increase collaboration.

- **DevTester.** As in most software organizations, our testers had relatively high technical skills. But that wasn't always the case, and in addition, most of the testing activities were done through the user interface, in isolation from the developers.. Our testers' performance was usually measured by the number of test cases (mostly manual) that they executed, and the number of high and critical defects they detected. We knew that this had to be changed. We learned from our customers and from market trends that we need to take our testers up a level – turn them into DevTesters. The main skills that a DevTester should have are:

    - Technical skills – The DevTester should be able to describe the system architecture and its main components, and understand which components will be affected by a specific change. In addition they need to be familiar with the technology stack of the application they are testing, drill down into log files and find the root cause of issues. The DevTester, in general, has a similar technical background as the developers, but in some areas at a lower level than the developers in their team. In addition, we expect that the majority of the testing will be validated at a lower architectural level than the user interface for two main reasons: Firstly, tests at the API level run much faster, critical for integrating them with our continuous integration (CI) process. Secondly, in many cases we need to validate the feature at an early stage, before the UI is ready.

    - Be the end user ambassador – The DevTester still needs to be able to drive the end-to-end scenarios in the application, know what the common use cases are, and understand the implications if there are issues. They need to know who the main personas that use the products are and how they do their job.

    - Tools – Be familiar with developer tools: IDEs, source code management, CI servers,

configuration and deployment tools, databases, as well as automated functional- and load-testing tools. They are not expected to be experts, but their work environment has changed. We expect them to be capable of writing automated test suites, run basic load testing, and integrate them with the CI server.

- Collaboration – Need to work side-by-side with the developers, be able to speak the same language, empower each other and work as a team.

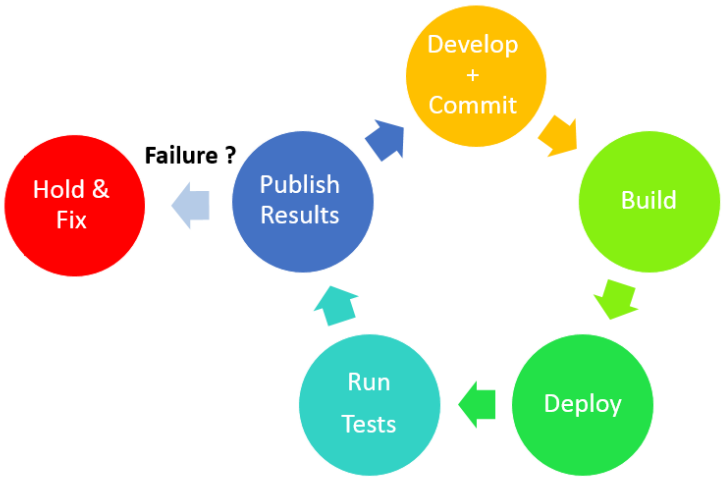
We had to run through an assessment process with our testers to validate who could function as a DevTester. Those who were found capable went through a training program to teach them the technical skills they need. Those who weren't capable were reallocated either to another project, or to a different role in the organization. In general most of the testers were capable of making the transition.

- **"In-cycle" quality activities** – The majority of the quality activities are executed during the sprint. The DevTesters plan their testing strategy together with the developers and product owners during the planning phase, and perform the testing before the end of the sprint. A few still need to be run after the iteration:

  - Manual vs. automated testing – One of the most critical points to understand when discussing manual vs. automation testing is that these are not the same activities. We cannot compare them, and the motivation for each activity is different. The automated test suite is a key tool for our teams to quickly create a quality baseline and to check for regressions on the main flows. The automated tests have defined flows with expected results for each action. These repetitive actions are automated and integrated into our CI process. We also use the automation suites to validate multiple environments that our product supports. We started to validate working with ATDD (Acceptance Test Driven Development) to accelerate the automation process as part of the user story definition. Automation can be done at all levels – Unit, API and UI.

The manual activities are focused on exploratory testing, where the goal is to find issues in more complex flows that require deep customer understanding together with a creative mind to find the non-trivial issues. Our DevTesters together with the developers perform these two activities during the sprint, and they are mandatory activities that are required in order to mark user stories as "Done".

- Continuous Integration – One of the biggest success factors for the change was building a robust continuous integration process. For every commit that is done by a developer we



initiate a full build--deploy-test process.

A key factor for a successful CI process is a short feedback time, so that a developer can quickly validate if a change that he made introduced new issues. In our case, we wanted to be able to get that feedback in less than 30 minutes. The build and deploy process lasts about twenty minutes, leaving us around ten minutes for our automation suite to run. We execute all of the unit tests, as well as the majority of our API tests, and several end-to-end UI tests. In addition there is a nightly job that runs longer, as it deploys on more environments and executes more intensive tests. The combination of both gives us fast feedback on the changes that happened. Our tools enable us to gain a good understanding of each change:

what functionality was added, who contributed to the change, and the change's quality.

- "Out-of-cycle" quality activities – Although we aim to complete all of the quality activities during the cycle, some activities have to be postponed. These usually include performance testing and security validations. Low-scale load testing and basic security checks are run as part of our CI process, and executed by the DevTesters, in order to get "smoke"-level testing. These provide us with a basic benchmark to ascertain if any scalability and security regressions were introduced during the cycle. However, in some cases more robust testing is required, calling on our experts to run more intensive and complex testing. We work together to analyze which changes require these activities, and try to start them as early as possible. CoEs (Centers of Excellence) are still maintained to support such tasks.

Another major safety net is our internal farm. As we use our SaaS-based product for our own development, we first deploy the release on our internal environment, validating that we achieved the quality that we targeted and that the new functionality works as expected. Only then do we "push" that change into our external production farm.

These are the major changes that we have made. We have gone a long way to reach the point we are at today, and there are still areas that we can still improve. Our quality activities today enable us to release quickly and provide quality for our customers.