

Managing international distributed test teams in Scrum

These days test teams are forced to deal with many challenges when they adopt one of the different Agile methods; among them is "Scrum". There are a many different possible activities- daily meetings, Sprints etc. There are also a variety of terms that the average tester and test manager at different levels must contend with.

Let's assume that we understand a particular activity. We have grasped what the managers are looking for and what concerns the Scrum Master. Moreover, let's assume that the test engineers, TL's and the test managers understood why this Scrum approach is appropriate (If there are still unclear areas here, this is a whole different article). Now we are facing a totally different set of problems:

- How are the scrum teams built, and how is distributed team communications handled?
- How does the team run in everyday life, when parts of the team (both developers and testers) are in different parts of the world?
- What do we do when several test teams test multiple different products during the same time frame?
- And on and on.

The following article provides examples of how we (as a test team) dealt with the problems above and describes mistakes we made and lessons learned. Obviously, there are many more ways to deal with this set of problems. Where relevant , we will share with you the different thoughts we had before making a decision.

Following is a bit of necessary background:

Our test team was assembled from teams in the US, Israel, and China while development was divided between teams in the US, Israel, China, and Ukraine (outsourced).

1. The product line which the QA department was in charge of, included 4 different "off the shelf" software products, where each one had a distinct life cycle, it's own targets and goals, release versions etc. Moreover, there were different one time projects that occurred during the year (between 5-6 a year).
2. Between the different products, there were many overlapping areas, meaning test engineers could efficiently test more than one product.
3. The tests were divided into 3 major sections:
 - Manual tests (including requirement analysis, test designs etc.)
 - Automation development "on the go" for newly added functionality.
 - Customer simulation labs, which were established to simulate the main customer's environment, working procedures, configuration and up to date data from these customers.

4. All of the test engineers were divided amongst 5 scrum teams. History shows that "team formation procedures" happened once a year on average. (Team formation consisted of the assignment of all developers/testers/analysts etc, to scrum teams in order to fulfill the backlog as efficiently as possible).
5. Company standards – each scrum team should include at least 1 QA engineer (Most of the times it was more than one). No engineer could be part of more than 2 different scrum teams at once (Due to the overlap between products, it made sense that some engineers might test 2 different products in these overlapping areas).

How were the teams built and how was communication handled within a distributed team?

When we met in order to form the scrum teams according to the product's future needs, we could either disregard geography and form teams based on capabilities and knowledge only, or position geography as a limitation to guide the team's structure. The 2nd alternative basically means that a given scrum team won't have members from 2 different countries.

We chose the 1st option as our model, meaning capabilities were the leading factor. Multiple scrum teams were formed with engineers from different parts of the globe. We did however manage to assemble the teams with people residing in no more than 2 different time zones; to facilitate communications. (Examples – US, Israel /Ukraine, Israel /Ukraine, China /US, China) .

The major disadvantages of this approach are the time zone challenge of daily communication among different team members and understanding the team's status along the sprint.

We overcame these problems by using a few techniques (most of them easy to use):

- Daily meetings: At first the local team (where most of the team members were) held the daily meeting alone. The other remote parts of the team would send daily updates by email and receives updates the same way after the daily meeting. As time passed it was understood that despite the comfort associated with this offline approach, moving to an online meeting with ALL team members, prevented dragging issues along, and improved communication and relationships between team members. Plus, issues were raised more frequently. Hence, with the absence of a sophisticated video conference mechanism, most daily meetings were done by phone; parts of them on Skype + webcam. The teams suffered some difficulties during the 1st few weeks, but after 2-3 weeks, meetings were held as if all were at one geographical place (Of course some teams had it better, some worse, but on average it was more than satisfactory).
- Personalized communication – the team members tried to communicate face to face by using video, chat and phone, and less by documents and email. This made information transfer significantly easier despite language barriers.

Beyond that, there are a number of difficulties to consider:

1. Time differences – In case different parts of the team have many hours difference (In our case US – Israel/Ukraine is 10 hours, Israel/Ukraine – China is 6 hours, US – China is 16 hours, meaning the team is talking but in a different day, which by itself is hard to grasp) is something that needs to be taken into consideration with every step the team makes.
2. Cultural differences – At once (the minute the team is formed), both the QA engineers as well as the developers are required to work smoothly with their peers who may belong to an entirely different culture. The difference can be in terms of anything; the way they talk (even if all sides speak English, different accents may make it impossible to understand at first), to the way they think, working habits, and much more.

The way we approached this issue was by going through a cultural differences workshop which stressed the key differences and where everyone needed to be careful. Over time it became much easier, although it did take a significant amount of time.

3. Exercise – At first communication between the team members is problematic. A simple message which is supposed to be conveyed readily by speaking or writing requires a relatively a long time. The effect of this (and I have personally encountered it many times , with some people after years of practicing scrum) is that many team members avoid making online contact. The name of the game here IMHO is exercising it as much as possible by talking directly on video/phone/chat rather than by offline (E.G. mail).

How to operate scrum development and tests when different parts of the team are around the world?

Beyond online communications, Scrum teams need to maintain some helpful aids to manage their joint information.

- Scrum board – Initially the teams did it upon an Excel spreadsheet, which was at a shared point, and thus accessible for the entire team regardless of their physical location. Today it is managed by scrum management software (detailed below), upon which the teams can view and update the tasks / tests / User stories easily. By that we prevented the need for multiple scrum boards for each team (in every site where part of the team was located).
- Scrum management software – Examples for those are Rally, Version One, Green Hopper etc. By using this software, the teams have online access to all of the relevant data for managing their tasks – from product backlog, user stories details, scrum boards, STP's, STD's, impediments, sprint backlog etc. Every detail that is being update by the product owner or scrum master or by one of the team members is seen on the spot, and some of the daily meetings are being done in front of the boards within the software. One of the advantages of using such a tool is that all the material regarding the different user stories (from requirements documents, any discussion done regarding it, added material, and of course the development state) is in one centralized place and convenient to use.

Real World Experience

When the company went through this transition, we knew we wanted to conduct the functionality tests as close to the development stage as possible, and at the same time provide relevant automatic tests while the 1st cycle of tests were done. Up until that point there were many cases where there was a time gap between end of development and the 1st testing cycle period.

We realized we had a gap to bridge in the 1st few sprints (We should have fully tested the functionality that was developed during the months before, plus the new functionality which would be developed during the first few sprints). To overcome this problem, a special task force was created with testers, CS engineers, test oriented developers and more. The purpose of this team was to eliminate the gap. After 2 sprints the goal was achieved. Today the testers in the different scrum teams provide tests covering approximately 80% of the functionality developed within the same sprint. Within that sprint the testers develop automated tests for about 50% of that functionality (So much more room for improvement). Naturally, if a given function is supposed to be potentially shippable at the end of a certain sprint, it is developed, tested and fixed within that same sprint (Luckily, this need was rare required).

During the first few months it was hard for the team members (Developers and testers), to move away from the traditional phases of software development – develop -> test -> fix -> re-test and so on. Well, as we all know, even in traditional methods there are parallel processes (Example – writing an STD during the design phase), and yet it seemed that despite trying to make the processes as parallel as possible in order to reach the finish line faster, we were still under the influence of the "old" way of thinking. Step by step, our methods changed. For example – an early design for a function, which was done by a developer, turned out to be test oriented. This meant that there was much thought about how to develop it, such that parts will be testable in parallel to other parts being developed, and of course without hurting the development efficiency.

This approach went even further. A year after we adopted Scrum, a complex project was established for the leading product. A dedicated scrum team was assembled. That project required extensive infrastructure work, new complex algorithms, wide and complex GUI, and there was a very short timeframe. The team was assembled from engineers in the US and China. Only one of the team members had relevant Application knowledge, so there was a learning curve for everyone. The team got to work and made a list of development tasks and tests, where every module in this project would be testable from the user level (meaning through its dedicated GUI). Hence, from the moment the planning phase ended until the end of that project, it seemed that during every daily meeting, someone from the team asked: "just finished the last thing I have done, what is the best thing to do now?". The decision regarding what should be done next, was made by the entire team. Where the chosen next task, was not necessarily familiar to the chosen team member (meaning extra learning was required). Yet it was clear that in order to prevent bottlenecks, this was a necessity. In many cases, developers helped in testing (under control and support of the testers and test TL's), in part based on a written STD, and in part exploratory, according to the knowledge and expertise of that developer. On the other hand, testers helped developers perform better unit tests where it was

found necessary. That actually meant that almost anyone could do anything, with a bit of learning when necessary. The primary goal that the team took upon itself was to complete the project on time, and with the desired quality, and to reach the milestones (which were defined by the team) within every sprint. As a personal experience from that project, I can add that as the one in charge of the testers, as an observer, and a manager this was the project I had the most fun participating in.

What did we do when several testing teams test a number of products in parallel?

As you know, every sprint begins with it's planning, where at the end of it, the relevant scrum teams commit to provide certain functions. In our case there were many test engineers that by their capabilities and the company's products, were required to take part in more then one scrum team during a given sprint. For example – in case we have 2 different products, with an identical functionality to be developed for both. As a matter of fact, when the testers and TL's got to planning a sprint they faced a few dilemmas:

- Given 2 products and therefore 2 different backlogs, how do you prioritize between every task within one of them and every task in the other?
- What is the required capacity for each product? How do I as a tester or a TL define what it is for every product?
- Which scrum team, given the tasks which are it's responsibility, needs a particular tester more for the next sprint?
- When the sprint begins, how can we make sure a tester doesn't move between teams too many times causing unnecessary context switches?
- And on and on.

Our way to solve the above dilemmas consisted of 3 steps done sequentially:

1. Understand the overall picture in terms of constraints and the developer's plan: the QA manager publishes the priorities, meaning which versions for which products are about to release and what are the priorities between the products accordingly. (For example – version X at product #1 is 1st priority, version Y at a 2nd product is 2nd priority and so on). In parallel, every tester sits down with his scrum team to understand the developers plan, meaning what does the team see they will work on during the sprint, which developments are riskier and their meaning etc. Moreover, identify leftovers from previous sprints to the scrum team, the TL and the QA manager.
2. Have each tester / TL prepare an individual plan – every one prepares a plan as a base for changes, since in the planning things will change...for sure. This plan takes into consideration left overs from the last sprint (if there are any), relatively risky developments ahead, the different backlogs that need to be considered, and of course the constraints from phase #1.
3. Synchronize all personal plans, providing feedback and planning the tests: in case a given tester belongs to one scrum team, the issue is relatively simple, since only the relevant

product's needs and his scrum team's plans affects his sprint's preparation. When the situation is different, we gather opinions from all testers and prepare plans together:

- If there is a product that will have large gaps, which we can foresee prior to the sprint's planning, meaning if due to lack of resources, this product won't get the needed tests during the coming sprint. Needed response means testing every function developed, releasing a version (Release procedure) where needed, and automation developments to ease future tests.
- Is a different resource allocation from the tester's recommendation needed in order to cover for such gaps? After we give the testers this feedback, and making adjustments while having the testers take full part in the decision making, every tester has a general picture as to the tests he will probably do within the sprint. Of course this is not a final plan, but a base plan since during the sprint planning things will probably change, at least to some extent.

The purpose of the 3 phases above was to make sure that the resource allocation between the products was sufficient for the coming sprint including the following sprint's sniffing period.

4. Timing of the tests that were taken within the sprint – during the planning you can pay attention to "when each test can be done within the sprint". We tried to put risky tests first and of course tests which are dependant on others. In addition, we tried to unify tests that required similar actions to improve the entire process. The main idea is to map the sprint such that you won't get stuck with half of the tests in the last few days of the sprint.

At a personal note, I might add that the process above didn't always look like this. It took us some time to get the people abroad to follow this procedure, and till recently, there was still much room for improvement.