Agile Development in a Geographically Distributed Team

Tanay Gavankar

Large projects that require many developers often run into issues of synchronizing work and maintaining order. The agile development process helps organize developers and is heavily based on the assumption that the developers are geographically close. In order to successfully distribute an agile development process geographically, there are four primary “zones” of consideration that must be addressed. These four zones consist of team coordination, routine group communication, social robustness management, and coding quality practice. Mozilla has successfully addressed all of these issues, and thus has become exceedingly proficient at distributed agile development. I was on the web development team at Mozilla, and we were able to build our product with team members scattered across the globe.

When adapting the agile development process to a distributed environment, the first consideration to take into account is how to handle team coordination. Developers have to know what they are in charge of and what the other members of the team are working on. In traditional agile, the team is coordinate by means of daily standup meetings. However, this model is not feasible for a distributed team. At Mozilla, my team was extremely geographically scattered. In a development team of 5 people, no two people (not counting me) were in the same location. I was in Mountain View along with my mentor, one developer was in New York, another in Michigan, and a final in Florida. In addition to the engineers, the product team was primarily based in Europe. While the difference in time zones was only 3 hours between the west coast and the east coast developers, the product team was 8 hours ahead, and we had to interact with

them quite often as well.

Since accommodating everyone for daily meetings would prove difficult, Mozilla relaxed this requirement. The daily standup meeting only had to be attended by developers rather than the entire team (which would otherwise include the product team members). By doing this, they reduced the time difference of whom the meeting needed to be scheduled around from 8 hours to 3 hours. So, the daily standup meetings were scheduled for 9am PST every day. These meetings were held via Skype video conference. By using video conferencing, the meetings enabled real-time voice communication, which is significantly faster and more efficient than, for example, text chat. Also, the video allowed for a more personal experience and made it simple for diagrams and visual aides be shared with all the meeting participants. It was also integral that Skype was used rather than the Mozilla corporate video conference tools. By using a free, publicly available software, team members were able to join the meeting from anywhere they had internet access, even if they were not in their official office. This meant that it was significantly easier to attend the meetings, especially in the case when a member was running late and had not made it to the office by the time the meeting started.

In addition to how the meeting was held, the content was also important. During our meetings, the team lead would first go over any pressing issues that had been brought to his attention (such as information from higher-ups, new/changed deadlines, or modifications to the short-term product goals by the product team). After this initial phase, a roundtable began where every participant would discuss what exactly they worked on the day before and then what they plan on working on for that day. The

general rule for explaining was that if what they worked on was a regular bug fix, they give a brief overview of the bug, and an estimated timeframe for when it should be fixed (or if it is already fixed). In certain cases with more convoluted bugs and resolutions, the developer talking often went into more detail explaining the cause behind the bug, why it was not caught earlier, and the implemented solution. This was extremely important as it ensured that all the developers would be conscious of any future bugs that may be similar in nature, and thus saving valuable developer time in the future. This indepth explanation was also given if the developer was working on a new feature, especially if they had encountered new issues during the development of this feature.

However, it is still important that the product team meets with the developers. So, at Mozilla, since the product team was not part of the daily standup meetings, a weekly meeting was held that was for both the developer and the product teams. This meeting was held every Tuesday at 9:15am, which was a time that all timezones were able to attend. For the members in Europe, this meeting was at the end of the work day, and so it was better to have the meeting only once a week rather than every day. This meeting was held via a phone conference call where members could dial in. The number of people in this meeting was much larger, and so using a tool like Skype would not have been as beneficial. During this meeting, only high-level goals and issues were discussed and only the team leaders would talk. So, it was more of an informational session to make sure all the team members were on the same page. After the team leaders spoke, there was an opportunity for any of the team members to bring up any questions or concerns they had.

In addition to the standup meetings, an internal wiki was used for weekly status reports. These reports were written by each team member every Friday afternoon and would encapsulate what they worked on for the past week as well as what they plan on doing next week. They were essentially a condensed version of the standup meetings, except in a written form so that anybody could check them. They were not checked by managers, but were rather just a form of simple recordkeeping. Also, Bugzilla was used for issue tracking and assignment. Every time there was a new feature or a bug, it would be filed in Bugzilla. Then, the bug would either be taken by or assigned to a developer. This enabled accountability for a feature or issue, and so team members could determine who was working on a specific task by simply looking at the bug. Using Bugzilla as the central hub for task management also simplified milestoning and triaging. Since all this was done entirely online, it fit well into the distributed agile process.

The second zone that must be addressed when distributing the agile development process is routine group communication. In the classical agile process, an ideal setup would be to have all the developers in the same room, thus allowing face-to-face communication whenever its needed. However, as is evident, this is not a reasonable expectation for a distributed environment. Mozilla has overcome this issue by simply tweaking its culture. Rather than talking in person, it is implied that all communication should take place over IRC. They have a large IRC network with many channels; each team gets its own channel. On my team, we had one channel for the developers and one for the entire team (both developers and product team members). In addition to these, there were also channels for supersets of the groups, such as one

channel for all of webdev, etc. We were expected to use IRC for communication even if the person we wished to talk to was sitting next to us. This hierarchical structure of channels made it very easy for getting help. At Mozilla, it was an unspoken rule that everyone should be positive and nurturing. Because of such a culture, if somebody ever had a question, they could determine what the scope should be (specific to a project ranging to a broad webdev question) and quickly ask the appropriate channel. By broadcasting your question to everyone, responses were much quicker along with the benefit of having multiple people's opinions on the solution.

Another benefit to using IRC for routing communication over in-person meetings is attention management. People can easily lose their train of thought if they are interrupted by coworkers asking questions in person. However, by having all communication be on IRC, team members can focus on their task at hand until they are at a point they are comfortable looking at IRC. Context switching is an expensive operation, and so by reducing the number of times that happens as well as only having a team member change their attention when they are comfortable, reduces the total cost of development.

By deferring all communication to text-based IRC, this can lead to weaker interpersonal relationships. This brings about the third consideration of managing social robustness. Traditional agile exploits the developers' locational proximity to make them have "strong ties" with each other. The social hypothesis of strong ties states that two people have strong ties to each other if they interact with each other, feel affection for one another, and have a history of interaction. Strong ties are better in a development environment because they facilitate the creation of novel ideas and better team

communication. With a distributed environment, the requirements of having a strong tie are often not met. This results in a more distant relationship between team members, which may be embodied in more respect, and thus a less productive environment. At Mozilla, this issue was addressed through onsite work weeks (also known as “onsites”). Each individual team would have an onsite every few months where all the members of the team would be flown out to a central location for a week. During this week, the development process becomes purely traditional agile, where everybody participates in all meetings, all communication is done in person, and all the team members are sitting in the same room throughout the day. In addition to working together, it is also strongly encouraged for the team to participate in non-work related activities, such as eating meals or going out for a drink. In fact, these activities are so strongly encouraged that, often times, their cost will be expensed to the company. Social interaction both in and out of a work environment for strong ties to form between team members even if they usually work far apart.

The final consideration of distributing agile development is enforcing code quality practices. Traditional agile does not say much about code quality, but one of its key features is to iterate and progress quickly. Having faulty code causes stagnation, and thus goes against this philosophy. At Mozilla, there is a strict code review policy in place. Every commit, no matter how big or small, must be peer reviewed and approved. Github is used for source control, and fortunately, since Mozilla is all open-source, they are able to use all of Github’s features to their maximum potential. Each developer uses their personal account and forks the official Mozilla repository. The developer makes a branch for whatever they are working on, and submits a pull request for their patch.

Another developer on the team must review the pull request and give comments on Github itself. This is an extremely important detail; often times, it is more difficult to tell somebody that their code is poorly written face-to-face. By constraining all comments about a pull request to Github, the reviewer is able to detach themselves from personal feelings, and it makes the review be more about the code rather than be ad hominem. Upon review, the patch is eventually given then “r+” (approval), and then it is merged into the master branch. By having more correct and better code only be committed, it allows development to proceed much more quickly.

Furthermore, there is no formal limitation on when a developer can also become a reviewer. I was allowed to review my coworkers’ (albeit small, initially) patches within the first two weeks of starting there. This puts everyone on an even field, and also helps with the distributed process. Often times, because of time zone differences, only specific developers might be available for review, and there is a patch waiting to be committed that has be landed. In such a case, having anyone be a reviewer was a benefit as the developer does not have to wait for a specific person to give their approval.

While addressing these four considerations as Mozilla did has allowed them to successfully utilize a distributed agile development process, it still has its drawbacks. The largest drawback I noticed was that creative brainstorming was made immensely more difficult. Coming up with solutions for a problem becomes significantly trickier and slower when there is no face-to-face communication. When brainstorming, it is important to be able to explain an idea and thought process in a manner such that others can understand the solution. Furthermore, it is often helpful to draw diagrams

to assist in the explanation. With a distributed team, both of these become much more difficult. Mozilla uses the aforementioned onsite work weeks to flesh out ideas and brainstorm, but sometimes there is no opportunity to wait for one of those. This problem has not been solved yet by Mozilla.

Mozilla has been constantly adapting their development process throughout the years for a distributed environment. They targeted the four important considerations and addressed them with proper solutions. In particular, team coordination was handled by having subgroups of the team do the daily standups with the entire team holding a weekly standup. Written status reports as well as Bugzilla also allowed for simpler progress tracking and keeping the developers in sync. The second facet, routine group communication, was facilitated by IRC, but more importantly, but a culture that put text-based communication over face-to-face meetings. This was balanced by promoting social robustness, the third consideration, by holding regular onsite work weeks where coworkers would both work and socialize together. Finally, coding practices were upheld through a rigorous peer review process to ensure that progress was being made. While all these considerations help make Mozilla successful at distributed agile development, they have still not been able to completely solve the issue of how to hold creative brainstorming sessions. Mozilla has the distributed agile development process figured out, and so it can be applied to any company by simply addressing the four considerations mentioned.