Extending the reach of collaboration

By Johannes Brodwall

If you have ever worked on a software project, you know that it will succeed or fail based on whether everyone on the project understands what the need in order to pull in the same direction. Agile projects use techniques such as retrospectives, collaborative planning and pair programming to spread the knowledge of information inside the project.

Working on distributed projects opens new challenges to collaboration. As a matter of fact, these challenges aren't new, it's just that the distribution exposes them more clearly.

To enable good teamwork in distributed projects, I have found ways to extend classical methods from Scrum to a distributed setting. These techniques are useful even if your project isn't distributed.

The first collaboration challenge is to make sure we're building the right thing. I like to say: If I'm not a developer and I get a bright idea, I also get a problem: The idea isn't worth much as long as it's just sitting in my head.

In order to share a vision for a project, a release or just a new feature, teams find it useful for the customer of the project to communicate directly with the development team. A good way to approach this communication is to refine the vision together. I've seen two approaches with good success in doing this:

The first approach is to create a "product box" together. Let's say that we were shipping the product as a CD in a box. Write the headline on the box, the feature list, put a picture there. The customer and developers can do this together as a group exercise.
The product box facilitates communication about many of the most important questions: What's the name that will be printed on the box? What is the tagline that will be used to promote the product? What features and benefits will be listed on the back?

Creating a product box is a fun experience for the team and the working with a physical thing makes it even more enjoyable. However, because it's about creating a physical object, it may not be a good match for a distributed team.

Another approach is the popular elevator pitch statement: Let's say you're taking the elevator up with the CEO of the company and he asks what you're working on. Can you describe it in the time you have before the elevator reaches the top?

A good template to use for an elevator pitch is this one: “For *some stakeholders* who *has a goal*, the *name of the product* is a *kind of product* which *gives a feature.* Unlike *a good alternative* this *gives some advantage*.”

For example: “*For a busy sales officer who wants to find good prospects to contact when visiting a city, the CRM customer radar is a mobile map application which displays nearby customers and prospects on a map. Unlike searching for customers in the CRM portal, this shows relevant information at a glance*”.

This statement covers the important parts of the product: Who is it for? What is their goal? What can they do with it? What could they do instead? Why is this better?

I find the elevator pitch statement especially useful in a remote setting, since it gives a clear and verbal task that can be completed relatively quickly.

When we create an elevator pitch, I like to gather 2-5 people from the customer organization and a similar number from the development organization. Often, we’re forced to do this as a video conference. Often, the customers are all in one room at one end of the video call and the developers are all in another room at the other end of the video call.

In this setting, I like to use the exercise as a chance for customer and developer team members to get to know each other. So I break up the workshop into teams of 3 (or 2 in a clinch). The teams should be composed of people with different backgrounds, so that each team has at least one developer and one customer.

The teams set up separate video conference calls between its two sides (one person on one side and two on the other). They spend 10-15 minutes creating an elevator pitch together. Afterwards, everybody rejoins the big workshop and we create a shared elevator pitch that takes the best from the work of everybody.

In addition to the benefit of getting to discuss the details of the product together, this workshop creates personal ties between members of the development team and customer team which will benefit the project for a long time.

This is only one of several exercises you can run to help the team and customer describe what they will create. But now, let’s talk about how they are creating it.

The difference between a well-crafted program and a poorly created is enormous. The well-crafted program may be less than half the size (and effort) than the poorly created one. The well-crafted one will fail much less often and when it fails, it will do so in a way that is easier to repair.

The difference comes down to learning. As programmers learn about their programming language, how to approach a programming problem in general and what technical solutions are available to help (or hurt them), they will create better programs. The learning never stops.

The most reliable way I've seen of integrating learning into the daily lives of teams is through pair programming. Pair programming is a technique whereby two programmers share the same task, as well as a screen, PC and keyboard. Most programmers get help from one of their pair on occasion, say when they are trying to fix a difficult bug. Some teams use pair programming occasionally and some teams use it almost exclusively.

I've had the benefit of working on teams that used pair programming almost exclusively on three occasions. In all of these case, the culture gradually became one of continuous learning. In two teams, we switched who we worked with most days. Every day, you were either introduced to a new problem that someone else had some good ideas about or you introduced a new person to the problem you had been working on. In either case, I always found that there was something I could learn from my pair programming partner.

In the last team, there was for most of the time just me and one other programmer. As we started to exhaust what we could teach each other, we would both do research to have something awesome to share the next day.

In a distributed setting, pair programming becomes more difficult, but with practice, it can work quite well.

The most obvious approach is to use a screen sharing program like TeamViewer or GoToMeeting. These programs generally have two problems: First, the voice and video is worse than that of leading video conferencing programs such as Lync or Skype. Second, the responsiveness of remotely controlling another computer can be frustrating, especially if the programmers are distributed at a great distance.

The solution to the voice problem is pretty obvious: Set up a Skype conference call next to the screen sharing program.

To give a more responsive programming experience, I use another trick: Put the project files on a folder shared with Dropbox or another file sharing tool. When I'm at the keyboard, I share

my screen with my pair. When my pair takes over and shares their screen, the files have been synchronized in the background. This way, both programmers get same experience as if they were working on their local machine.

The elevator pitch allowed us to start collaborating and remote pair programming helps keep the development going. But it's still important to take some time out and look at how we're working as a team.

Scrum uses the practice of a regular retrospective to achieve this. A retrospective is a workshop for improving the way we work together. In the retrospective, we look at what we have been doing lately and find ways to improve the way we work together.

For example: A common pain for teams is hand-overs: When someone starts developing on a task - is that task understood by the developer? Similarly, when a developer declares that they are done with a task, is that task in a state the the customer would recognize as having created value?

A team that collaborates well can really make a difference in this approach. Teams with a lack of trust between the customer and the developers will often come up with an increasingly stringent checklist before something is "ready for development", including a written, detailed set of test cases, UI sketches etc. For teams that collaborate well, I see a different outcome: the teams decide that when a developer starts on a task, they will spend a few minutes coming up with a general idea of how it will work and then talk with someone from the customer to verify and extend the knowledge.

How to work together on tasks is only one of the ways that a retrospective can help a team work better.

Running distributed retrospectives can be hard, but just like creating an elevator pitch together, it works best with a combination of smaller workgroups and the whole team. Here is my general approach to a distributed retrospective, inspired by the book Agile Retrospectives (Larsen and Derby):

First, make sure that everyone is ready to participate. A good start is to ask everybody to answer a simple question individually, for example "what single word describes to you the last few weeks of the project?" (In a distributed setting, it's important to address people by name, as eye contact and body language doesn't work so well.) This way, the retrospective makes it clear that everybody's expected to contribute.

Second, get some good discussion going. This works best in small groups. Just like with the elevator pitch workshop, this works well with groups of three which set up separate video conference calls. A good question to ask the groups is “list as many things that we achieved in the last few weeks as you can”. It’s easy for a retrospective to get focused on the negative, and this sets the tone for a positive dialogue.

Third, involve the whole group. After each group has shared their achievements, an open question to the whole group like “where do we go next” and “what impediments are in our way” work well.

Fourth, make sure that the group decides on actions to take. This is a good time to involve those who have been quiet during the meeting so far. Often, they have been observing and thinking and may have the best insights at this point. Make sure that the list is concrete and that we later can know whether something was done or not.

Finally, close the retrospective on a positive note: My favorite question is basically “what do you hope to see that we have accomplished a few weeks from now?”

Working in distributed teams requires the same collaboration as working in co-located teams. The distribution adds a few challenges of their own. But with a few rules of thumb, these challenges can be overcome.

First, smaller groups collaborate better than larger ones. This is especially true when the groups are distributed. In a larger group, someone will probably sit too far away from the microphone or outside the camera. Three is a magic number, because a group of three can never have two conversations going at the same time. Split into work-groups and rejoin to share results.

Second, remember that eye contact and body language doesn’t work as well. When you’re addressing someone, be sure to use their name, as they cannot see you looking at them.

Finally, the best collaborating distributed teams are those who have met in person. The informal familiarity that comes with teams having visited each other will ease the communication for the rest of the project.

The techniques outlined in this article shows how a distributed team can still be collaborating closely. Many of the techniques are also great for a co-located team. I hope you will find the chance to try out some of the techniques I’ve described on your project.