The Value of Cross-Functional Teams

What You Will Learn
This chapter discusses team organization. The heart of Scrum is the self-organizing, cross-functional team. Sometimes people forget the cross-functional part of it. There are two parts to cross-functionality. The first is that different skills need to work together. The second is that teams need to have all of the skills required to produce value.

Patterns of Challenge
Challenges in completing the work committed to in a sprint are common place. While there are quite a few of them, I’ve seen many fall into two categories:

1. Symptom: Testing is not completed by the end of the sprint. Reason: Testers do not collaborate on an ongoing basis with the developers.
2. Symptom: When multiple teams are involved, there is a tendency for the teams to work in layers instead of being truly cross-functional. Reason: Working in layers delays the integration of the function built.

Testing is not completed by the End of the Sprint
It is very common for Scrum teams to not complete testing at the end of every sprint. Even when this is done, developers are often moving on to the next stories to be developed. When testing lags the completion of coding, a significant amount of work is added to the development process (see Why Looking at Time is So Important). The best, and easiest, solution to this challenge is to specify tests before writing code. These test specifications need to be very specific. They are designed to answer the question “how will we know we’ve done that?” regarding a particular requirement or story. There are variants of this process called Acceptance Test-Driven Development, Behavioral Driven Development, Story Writing with Tests, and Specification by Example amongst others.

Our, admittedly biased, favorite book is Ken Pugh’s Lean-Agile Acceptance Test-Driven Development.

Splitting Teams By Layer
A few years ago I was doing a public Lean Software Development course in San Francisco. During one of the breaks two attendees from another city asked me to go to lunch with them so they could tell me about their specific problem. One was the director of the development group (about 80-100 people as I recollect) and the other was one of his top technical folks. They were describing the current structure of about a third/ of their group and it didn’t seem correct to them. They said that they were organized as three separate teams, each with their own expertise. These included a team of about 8 UI developers,
another team of about 8 mid-tier (business logic) developers and a third team of about 8 database developers as shown in Figure 1.

![Diagram of teams](image)

Figure 1: Composition of teams prior to course.

They had earlier recognized that by having the teams organized in the way they were delaying when functioning code was written. They told me that what I was saying in the course and how their teams were organized didn’t jibe with each other. After hearing me talk about flow, they sensed that this structure had inherent failings. The most significant one was that each team was working on their sprint somewhat independently. While they would start out with a set of agreed upon requirements, they would then work on their own and attempt to integrate the components at the end of the sprint.

This often proved difficult. The two attendees could now see that better coordination would occur if they reorganized the teams into cross-functional teams, that is, each team having some UI, mid-tier and database developers. This would enable each team to completely build an end-to-end stories. I agreed with their assessment and asked them why they had organized this way. They said it was because the teams wanted to organize this way and that Scrum says that teams are to self-organize so that they could organize how they wanted to.

I pointed out that Scrum actually says to have “cross-functional, self-organizing” teams. Their people had actually organized the way they wanted to without regard to what would work best. Not actually surprising since little explanation as to why Scrum actually works is usually presented other than you must build in stages. I then asked how their team had justified this structure. They told me the team said they needed to have the UI developers together to ensure a common interface. Pretty much the same argument was used for the mid-tier. And, of course, the database folks emphasized that a schema being worked on by several people independently would be risky.

Sounded very logical, and, given the team’s lack of training in Lean Product Development flow, could even have been true. but then I remembered my kids. In case you don’t know, I have four, with my youngest, at the time, being about 17. For those of you who have teenagers, or who have ever been a teenager, recollect the last time a teen wanted something and was talking to his/her parents. I’ll be willing to bet they didn’t give the real reason for what they wanted. Instead, they gave a reason that
sold. Teenagers are pretty smart, manipulating beings that know what works. Not all of this ability is forgotten when they become developers.

The challenge is that teams often want to create teams that might not be the best way to do so. Self-organization is good, but sometimes how the teams are formed that then self-organize need to be created with a bigger view. In this case, the reality was that the UI folks, the mid-tier folks and the DB folks liked hanging out with each other. They were each other’s buddies. But for them to say that doesn’t sell. To say you need to do it for system integrity does. Hence that was the reason provided. Consider this. If they were organized into three teams, each with 2-3 each of UI, mid-tier and DB folks, don’t you think the UI folks would talk to each other to keep a consistent interface? How about the mid-tier folks? It’s quite obvious the database folks would.

Ironically, organizing around the people you are closest too may cause problems. We must remember that people are tribal in nature. By that I mean that we tend to care more and associate with those closest to us. In our personal lives our families are the most important folks. In business, it’s the people we work with on a day to day basis. In large companies, we more identify with those in our department, then those in our group, finally those in our division, then those in our company more than other companies, etc., etc. There is nothing wrong with this. It is just the way we are.

But this can cause challenges when we need collaboration both along roles and teams if we structure our teams along tribal lines (e.g., UI, mid-tier, database). In this situation people tend not to associate that much with the other teams. On the other hand, if we organize our teams across roles as shown in figure 2, we are more likely to continue to communicate across teams with the roles which we identify with.

![Figure 2: Composition of the teams after the course](image)

Notice how this structure has several advantages, with few disadvantages. First, teams are now cross-functional. This means that complete functionality can be built at a story level by one team. At the end of the sprint the work should be completely developed, integrated and tested. Furthermore, although coordination must be made across the teams at the UI, mid-tier and database level, the people who need to do that coordination are a type of team already since they have a vested interest in keeping the
UI, mid-tier and database, respectively, well-coordinated. It is important to notice how our tribal nature works against us in the first organizational structure while it works for us in the second.

A side note is that the teams didn’t really mind reorganizing this way. Although they weren’t really given a choice, they now understood why it was a good idea. They were willing to try something that they realized was for their best interest but that required a little adjustment.

**Do We Always Want Cross-Functional Teams?**

There is no question that diverse teams make for better innovations. Steve Denning, in his masterful The Leader’s Guide to Radical Management: Reinventing the Workplace for the 21st Century\(^1\) says – “complex problem, like discovering ways to delight clients, is best solved by a cognitively diverse group of people that is given responsibility for solving the problem, self-organizes, and works together to solve it.” Scrum’s mandate for cross-functional teams takes diverse teams a step further by saying each team should have all of the skills needed to create the value needed by the customer.

Unfortunately, it is often much more difficult or financially expensive to create cross-functional teams than it is to create diverse teams. One reason for this is that we sometimes have certain people that are needed by several teams whose skill, or other abilities/knowledge is hard to replicate. We often think of these people being essential due to the skills they have, and this is, often the case. But there are other reasons that we may require someone’s presence. For example, they may have extensive knowledge about the problem domain they are working in. They may also have been present when the code was written years ago and know how it works – something that can be invaluable.

Although cross-functional teams have definite advantages, they are not always viable. Most all companies of any size will have people that need to be available to more than one team. This can be for any number of reasons. A simple example for large organizations is the role of the architect. Large organizations will not have a skilled architect for each team. Yet, architecture for large systems is an important skill to share across teams.

Sometimes the difficulty of cross-functional teams happens at medium size teams. For example, I remember a company I worked with that had several hundred engineers that sometimes needed someone with experience in metallurgical stress testing. This type of stress testing is incredibly complex and requires both a PhD and a lot of experience. I have a background in mathematics myself, but when I talked to one of the two stress test engineers about how they did their work I was seriously impressed. This is not the type of person you easily find or pay lightly. In this case, the two stress test folks were

---

split amongst several teams. Two were sufficient. Trying to get 8 more would have been exorbitantly expensive and unnecessary.

Even when we don’t have cross-functional teams, however, we do still need to make sure the right people are present at the right time to get the work done. This is where Kanban can be extremely useful and will be discussed in more detail later in Using Kanban to Coordinate People Working Across Teams.

**Summary – Pattern of Success**

It is important that teams self-organize but they must have the right composition of folks as well. Even the right composition is not enough – teams must work together as teams. In the first challenge I described how teams often had developers and testers but they are not working together. This is not a true team but two separate units under the guise of a team.

When splitting work up across teams, it is best to organize the teams to be cross-functional instead of by layer. This is not always the preference of the individual team members as they may not be aware that what they would like to do isn’t the most effective method available. This is where an opportunity for coaching by the teams’ leadership is available – provide the understanding of the laws of flow and recognize why one particular team composition is better than another. It is risky to always assume teams will structure themselves with the bigger picture in mind. In general, teams should be composed to enable them to work on the broadest extent of the value stream possible.