Table of Contents

What Should We Consider When Forming a New Team?

When you first bring Scrum, agile or Kanban into an organization you need to set up teams (see Why Do We Form Teams When We Transition To Agile? to understand why). Sounds easy, doesn't it until you starting thinking through the problem. That's when you start to understand that this may not be as easy as you thought.

This page describes some simple background rules to setting up new teams.

Background

If you read the books, the guidelines for team formation include:

- Size: Teams are typically 7 members, plus or minus 2, counting the Scrum Master, but not counting the Product Owner.
- Roles: Specific definition of full time Product Owner and Scrum Master.
- Make-up: Teams should be cross-functional, containing development, certification, documentation and subject matter experts, whatever is required to deliver value.
- Characteristics: The best initial Scrum Teams are formed with people who have worked together previously, people who understand the problem or domain and people who know how to use the technology.
- Co-location: Scrum Teams work best when co-located. And if you cannot do co-location, then at least ensure that team members are in the same time zone - co-timezone?
- Team structure: Oriented toward the delivery of value from the end user / customer perspective.
- Consistent: Teams stay together.

Approach

While it is optimal to keep Teams together for a period of time (they learn how to work with each-other) changes will happen as people decide they want to work with others, on other things or find they are not suited to the role they have taken on. We need to be open to these discussions so that we can improve the overall team structure of the organization. Often the Team itself will come up with a proposal to re-configure. But note that these changes should be relatively rare. The general approach to team formation is to “bring work to the team” not “form specific team to do this work”. From a management perspective the additional benefit is that it reduces the amount of “resource allocation” activities you have to do (if you have the “resource spreadsheet” with FTE's down to small percentages of allocation, you are in for a treat is you adopt a team structure more oriented to the value delivered).

For us the basic objective is to set teams up so they generally can take a requirement and turn it into

working software / delivered value and improve our delivery, our predictability, and our quality.

We aim to build a series of “functional teams” – a team with members that has membership from code development, QA, product management, UX, writers, etc - whatever is required to get the value delivered. This cross-functional group is needed so they can deliver value to “potentially shippable” state including QA and documentation.

Sometimes there is a need to form a “component team” where the team is responsible for a component (or more) in a product, and so provide services to functional teams. Pros for component teams is that they potentially reduce duplication. Downside is that they make planning more difficult as there is a dependency set up for just about every requirement where you need to deliver value (see Why Should We Work Harder to Eliminate the Effect of Dependencies? for more information). For a large product (lots of teams working toward the same product or service release) I’ve found the the ratio of functional vs component teams is about 70:30.

The idea of the “functional team” is hard for many organizations that are used to thinking about setting up teams based on the architecture of the product. Reality is that while the “architecture” approach may seem logical, its is basically the idea of “component” teams in the extreme, which means that every requirement coming into the systems requires extensive dependency management to get something out the door. And we all know that increased dependency leads to increase risk associated with our release of value. The traditional approach is an example of being resource efficient at the expense of being efficient in delivering value to the business.

We basically put everyone into Scrum teams, even specialized people such as architects and DB experts. We establish a rule that says “no person can be on more than two teams”. Practically this means that a (very) few people that are on two teams. Reason is that if we put people on more than 2 teams, they spend all their life in meetings and so are not very effective. Some experts are on 1 team and then when another team needs their capability they act as consultant / mentors while still remaining on their primary team.

Some additional notes:

- In places where we have worked this, we did not change the reporting structure of any of the people as we went through this process. We basically said “this is your new role”. We then said to (say) development managers who became product owners “as a PO you receive your direction from the product management chain.” Benefit of this approach is that we didn’t have to set up new roles that straight-jacketed people in a new and different way. Downside is that some had a problem with the feeling that there was no formal role.
- We told people that teams were permanent, that if you have something new that you want done that you bring work to the team, not create a team that will do the work. Idea here is that team can become more than “the sum of the parts” if they stay together. Reality is that in some situations, we made adjustments. See What Is The Effect of Changing Team Members on Velocity? for more information.
- If you have a reason to create a team with more than 9 people, you add to the risk associated with having a successful team because the team needs to support a lot more communication in order to synchronize their activities. If you have a team smaller than 5 members, then you are adding to the risk associated with having a successful team in that the team might not have sufficient critical

mass. Both these conditions can be made to work, but they are not recommended. In particular I've worked with a couple of teams that were very large - 15 people. What we found was that after attempting to work that way for a while, the teams typically told us that the overhead associated with keeping everyone informed was too much. They typically had a proposal on how to split the team to become more effective and in most cases we (management) simply accepted that proposal.

- It is more important to co-locate (or failing that, co-timezone) a Team than it is to come up with the optimal team. Its not that teams with remote members cannot work, its just a lot harder. This may require some creative approaches to team formation - don't let this become an obstacle. There will be sometimes good reasons for teams to have remote members. Just understand that this makes it harder to be successful as a Team.
- When forming functional teams you don’t need to create teams where every team can do everything. If you have 5 teams (1,2,..5) you are forming, and you have 5 areas of expertise in the product (eg components A,B..E), then the first team might be made up of people with expertise in A, B and E, the second with expertise in B, D and so on. Idea is not to aim for perfection, but create teams that can deliver value in most cases.
- If all these disciplines (eg development, QA, UX, etc) are not available, the team is not absolved of responsibility for the work required to do a quality delivery. It just means the team will have to determine how to get the relevant expertise and there will be an impact on the amount of value the team can deliver while they learn how to do these activities. Practically, what does this mean? A lot of development shops will have a shortage of quality assurance, documentation, functional designers, subject matter experts and so on in comparison to the number of developers that have. There are some approaches which can be taken:
    - Make sure that scarce resources are allocated to the “most important work”. If some teams are expected to have reduced capacity, lets make sure that its not the ones that are bringing the most value to the product.
    - Where these people are available they can be allocated 50% to two teams (provided they are near to each-other).
- The “co-location” issue will often solve itself even if you don’t start out that way. In one shop I worked in, more than 50% of teams we formed had people in locations such as US and India. Basically large number of the teams that were formed that way came back to us and said “we need to stop having remote people as we are not being effective / productive.“ There was good business reasons for forming things this way (for example, the main reason we had for forming teams like this in this shop was that we were often dealing with new people in the remote location. Over time and through the mentoring of the team, the remote people were able to take on work and the team came back and told us they wanted to re-organize due to the overhead of remote communication. When the team told us they wanted to re-org, we let it happen.
- Product Owner and Scrum Masters have primary role, but also role to help the team. Start assuming 100% allocation to their role then let them adjust based on experience. In the shops I've worked in we did not experience a reduction in productivity when we set teams up this way. The general perception was that productivity improved.

Want to Know More?

For more information on the specific roles see:

- What is the Role of a (Development) Team?
- What is the Role of a Product Owner?
- What is the Role of a Scrum Master?
- Why Shouldn't We Set Up Dedicated Defect Teams?
- Why Do We Form Teams When We Transition To Agile?
- What Is The Effect of Changing Team Members on Velocity?
- What Kinds of Problems Do You See When Agile Teams Churn Team Members a Lot?

Also see:

- Self-Forming Teams is an alternative approach to this problem
- How Do We Get All the Work Addressed When Our Specialists Cannot Be Everywhere?

Consultant, Tools, TeamFormation, Kickoff, FirstSprint, FAQ, Team