Chapter 2

The Scrum Team

What is a scrum team?

Multidisciplinary, self-organizing, non-hierarchical -- what do these mean? It sounds like a recipe for anarchy or utopia, and in fact, it's both. A Scrum team left to its own devices without the many specific disciplines of Scrum is exactly what Dilbert's Pointy Haired Boss thinks of agile when he says, "That means no more planning and no more documentation. Just start writing code and complaining."

However, it's the interaction of these elements of team structure and organization that allow the various Scrum practices to produce working software efficiently while keeping everyone engaged, empowered, and enthusiastic.

PMO: Project Management Office. The body that provides the full suite of skills and services required to service the project management needs of all teams in a traditional software development organization.

The multidisciplinary aspect of the Scrum team is the reason that the team is capable of taking responsibility for the entire project. There is no testing team, no architectural team, no user experience team, no programming team, no domestic and remote teams, and no PMO. There's no one else to blame if the team fails to deliver, because there is only one team, and that team has all the skills within it that are required to do the job. The team is also empowered to identify and seek out skills that it needs to incorporate into itself to be successful, if the team feels that some skills are lacking. During a project, the composition of a team can change as the team's needs change, and some team members may not be solely dedicated to the team full-time (although I strongly discourage this practice as it causes costly task-switching), but in the end, if you're on a Scrum team than for that iteration, you are just as responsible for meeting the team's iteration goal as everyone else.

What are Pigs and Chickens?

Just Google "pigs and chickens" because I will not re-tell the joke. The important thing about the concept of pigs and chickens is that you can tell them apart and yet they both count. Briefly, pigs are the committed ones, they are the ones whose day to day life is the project. Their calendar is full of stuff to do, but if any of it isn't adding

tangible value to the project, then they're doing something wrong. These are the people who are going to spend their whole day coding, testing, designing, or what-not to push the project goals forward every day.

Chickens count, too, and pigs can easily forget that. That's why the idea of "pigs and chickens" is valuable, because it reminds them all that there are two sets of players who both matter. Chickens are very concerned about the welfare of the project. They want to know how it's going, when it will be done, how it will work because they are going to use it, sell it, or in some way derive value from it. Keeping these people informed and happy is important, just so long as they don't keep the pigs from doing their job. Chickens could be salespeople, marketing managers, end users, beta testers, friends of the CEO, whatever. They have a right to be involved, but they are not actually responsible for the deliverables.

The way I do scrum, pigs do the heavy lifting. They decide who does what, how and when. They should be in every scrum meeting and every iteration planning meeting and every retrospective. They should contribute ideas and concerns. I consider it part of my job as a scrum master to help all pigs feel committed and valued, coaxing reluctant opinions and helping the team to work as a flat organization of equals committed to a shared goal. How they do that is the team's responsibility.

I also feel that the scrum master has a responsibility to the chickens. Chickens are welcome to listen in on meetings, but not to interrupt them. Good ideas, valuable input, and questions that come up from chickens in meetings can almost always be directed to the appropriate team member for discussion after the meeting. But it's important that chickens are allowed to feel involved. I make sure that they can attend meetings, view demos, access bug lists, backlogs, and tracking data. They may have valuable input for the product owner, but they will only be happy and buy in to the process if they feel that their right to information is being respected.

What is a self-organizing team?

Perhaps the best way to describe the notion of a self-organizing team is to contrast it with the alternative, command and control, model of team organization. In the familiar command and control model, one person or a committee is tasked with deciding what team members

should do, how they should do it, and how to check to ensure that they are doing what they are supposed to. In a software environment that usually means a project manager who makes certain promises to a client based on a detailed plan who then parcels out the work to programmers (the command half of the equation) and then sticks his head in their cubical periodically asking, “is it done yet?” to which the typical answer is “90% there!” That’s the control part of the equation. The weaknesses of this approach are that it empowers management at the expense of programmers and invests in the managers responsibility for something that they can’t really control. That’s not to say that programmers really have complete control over feature implementation, but they are a lot closer to the work.

In a self-organizing team, the team collectively takes responsibility for goals and is therefore more committed to them. The team parcels out tasks to the people who are most interested in doing them or most qualified to do them, again leading to higher levels of commitment. A team empowered to reach its goals in the way that works best for them can adjust to the work habits, social dynamics, and personalities of the team members to maximize efficiency in ways that a directed team can not. They also require less management overhead, reducing costs.

If you start with the assumption that most people would prefer to feel good about their jobs and the quality of their work, giving them the tools to succeed and to improve makes sense. Generally speaking, I’ve found that programmers more than most, want to enjoy and be mentally and emotionally fulfilled by their work. That’s where the learning aspect of the agile team comes into play. Giving the team a structured approach to learning from mistakes and improving their process and practices allows them to take an active role in improving the quality of their own work life by improving the pride they can take in their work. Everyone wins. More about this when we talk about agile retrospectives.

How does a non-hierarchical team work?
Effective problem-solving within a team depends on having a shared goal and empowered team members. That’s the essence of the self-directed team. The non-hierarchical nature of the team is essential for having a shared goal. When the boss says, “Your goal is to do X by Thursday!” then it’s not really your goal. But when a team of equals agrees to an answer to the question “What can we as a team

reasonably commit to in the next two weeks?" then you've got a goal you can get behind. It might seem counter intuitive to empower a team to tell their client or boss what their goal is, but remember that it is the product owner who set the priorities. Assuming that quality is a key ingredient of success, then forcing any unrealistic goal on a team not only degrades quality, but also drastically raises the risk of failure and is demoralizing to the team.

A non-hierarchical team is not one without a boss. The boss is the backlog. They are not making up things to do that have no business value; they are committed to delivering the backlog in the most efficient manner possible without sacrificing quality. If they are not, then the problem lies not in the method, but in the team selection. The basic assumption is that people like to do their best work. That doesn't mean that people like to deliver the most work. Not all people are workaholics and, really, that's not a healthy way to live or to create quality software. It just means that eight hours passes much more pleasantly for everyone when they know they are doing valuable work that is making a positive contribution to a goal that they have a stake in. People would rather the hours fly by in interesting activity than drag by in meaningless inactivity. People would rather feel good about themselves then live in fear that their shoddy work will be uncovered.

With the right people, the non-hierarchical, self-directed team is the ideal structure to let individuals shine, grow, and contribute their very best efforts.

What does a ScrumMaster do?
Coaching is the primary role of the ScrumMaster. If a ScrumMaster could go an entire iteration without speaking or writing a single email, that would be a success. The ScrumMaster's primary role is to ensure that scrum practices are followed, communication is happening effectively, and to remove any impediments to optimal performance. It is not to manage anything or anyone. The ScrumMaster is no one's boss. The ScrumMaster is no more responsible for the success of the project than any other member of the team. He or she may be the recorder, updating the burndown chart, or not. The ScrumMaster might conduct planning meetings and retrospectives, or not. It is only the ScrumMaster's responsibility to ensure that somehow, these things are done and done correctly.

The ScrumMaster also protects the team from distractions and the iteration from scope creep.

It makes for an unusual role, in that the ScrumMaster has practically no power and no responsibility, but that role is critical. To be successful, a ScrumMaster must be valued by the team, including the product owner. ScrumMasters can only influence through respect and diplomacy. In my experience, a good ScrumMaster makes all the difference, though, and is highly valued by development teams and clients alike, because they do for the team what everyone most wants. The ScrumMaster helps make the team better.

What about the others? Testers, programmers, UI specialists?
Task allocation is done by the team depending on team member's skills, experience, and interests. While it's conceivable that a programmer might take a design task or a tester take a programming task, there are no specific roles within the team. It is simply expected that the team has the skills within it to do whatever testing, programming, design, system administration, or other tasks are required to achieve the sprint goals. If it doesn't, then the team must ask the Scrummaster to find an addition to the team who has the required skills.

This is not to say that there isn't a tester or a programmer or a designer, but only to say that all team members are equal and equally responsible for meeting sprint goals. There is no testing department or design department to blame; there is only the team.

What does the Product Owner do?
The role of the product owner is perhaps the most crucial. It is the product owner's responsibility to represent the interests of all of the stakeholders, deciding what gets done and when and how every feature works in order to deliver the maximum value to all stakeholders and to the business.

The product owner does this in several ways. He or she constructs and maintains the backlog of work to be done and sets priorities to determine the order in which features are delivered. The product owner communicates the vision to the team, so that everyone involved understands the end goal.

In the iteration planning sessions, the product owner helps the team to understand the goal of every user story so that the team members who implement the features can see them from the end user's point of view as well as understanding the business case behind every story, so that they are able to make the best decisions as they face the myriad challenges that each feature represents.

During the iteration, the product owner is accessible to the team to answer questions and provide feedback in a timely manner to keep the team moving efficiently forward.

Finally, it is the product owner who decides when enough value has been delivered to release the product of an iteration to users.

The role requires a full understanding of the business case behind the application and the needs of the users as well as excellent communication skills and the full trust of the stakeholders. If any of these elements are missing, the project will suffer.

Why should there be a single Product Owner?
There should only be one product owner, and that person must have the trust and respect of all of the stakeholders. One of the biggest frustrations of in-house development teams is the way that stakeholders, especially those senior to the programmers, slip requests (demands) in at the most awkward times. People who don't program rarely understand that programming is not a technical skill. Knowing a programming language does not qualify one to be a programmer, nor does knowing how a computer works. Programming is an inventive, creative, problem-solving process. It requires clear thinking, focus, and creativity.

In an agile team, the product owner is the one who is solely responsible for the behavior of the product. The development team makes it work, the testing team ensures that it is stable, the Scrummaster helps the team keep the process efficient and effective, but the end result is solely in the hands of the individual playing the product owner role. This person represents all users and all stakeholders. They may have access to UI designers, subject matter experts, focus groups, and committees (Aarrrgh!) but in the end one of the beauties of agile models is that they put the customer back in the driver's seat. Just like an automobile driver, they may know their general destination, but they also have the freedom to make detours and to react quickly to changing circumstances, even arriving someplace better

than they had originally intended. Driving an agile software project is just as demanding as driving a motorcycle through rough weather on a crowded highway. The wrong decisions, or perhaps worse yet, indecision, can sour the entire journey.

What is the time commitment of the product owner?
The workload depends on the scale of the deployment and on how much support and buy-in you've managed to generate in your organization for this project. It is up to you as the product owner to declare when the business value realized by an iteration is sufficient to merit deployment in your organization. The biggest commitment is at the beginning, when the product owner first creates and then prioritizes the product backlog. During development, at least on the scale that I am most experienced with which is medium to large web applications, iteration planning meetings take up about half a day at the start of each iteration and less if the backlog is well-maintained and clear. It may be as little as an hour, depending on the complexity of the features and the size of the team.

The strength of agile development is that rather than spending an enormous amount of time defining the project up front and then waiting patiently to see the results, you spend only a small amount of time daily driving the project and course-correcting as needed so that in the end, you arrive right where you want to be. The tasks are not difficult or time-consuming, but the responsibility is great. You must be prepared to devote a little time every day and during that time, to give the task at hand your full attention and concentration. If you're ready to do that, then you can be my perfect product owner.

How can I be a good product owner?
In our experience, it is the selection of the product owner that makes the most significant impression on the success of the project. The right product owner is the one who is most effective at discovering the needs of all stakeholders (users AND investors) and working with the development team to distill those needs into user stories and clear explanations that provide adequate direction to the development team. The wrong product owner does not consider other users, doesn't seek creative solutions to user desires, is indecisive, or lacks fundamental communication skills.

By working with some very different types of product owners I have come to the conclusion that the right product owner has these traits:

They understand the business case. No one commissions software for fun. Building software is expensive, and so behind every software project is someone who expects to make (or save) more money with the product than it will cost to build. A good product owner understands the business case behind the decision to build a custom product rather than to use something off the shelf. If she doesn't, then the project runs a serious risk of having its funding pulled. Even if you succeed at creating useful software, the cost and value-impacting decisions, made without reference (or reverence) for the business case, could still mean that the guys with the money won't trust you again.

They understand UI design or respect the opinions of those who do. Even with expert UI designers on the team, the product owner is the one who approves or proposes the UI design that will ultimately be the sole interface or barrier between users and the functions beneath. A product owner who imposes their own ideas on the UI, without reference to other users or to standard UI design principles, will drive the team to create a product that is a burden to users.

They have a capacity to focus. The product owner role is not a full-time job. Most product owners spend most of their time as accountants, human resource managers, programmers, project managers, or CEOs. However, when they step into the product owner role, the project demands their full attention. Anyone who can't sit still and think all the way through a problem for fifteen minutes to an hour until they arrive at the best solution is ill-suited to drive an agile development team.

They communicate complex ideas well. There is an art to writing user stories. Ideally, a function is described in two or three sentences that communicate all a developer needs to know in order to fulfill a user need. Of course, there can be supporting documentation, but essentially, a good user story speaks for itself, leaving nothing pertinent to chance but neglecting the obvious and leaving room for developers to implement the best solutions. Anyone can write a lot; it takes special skill to communicate well while writing little.

They know how to use a crayon. Pictures do tell a thousand words. While my favorite product owner is proficient with Photoshop and attaches mock-ups to his stories, a simple willingness to wander to the

whiteboard or sketch out a design on a tablet can often save a lot of confusion.

They are responsive. A distributed team has to do agile development in less than ideal circumstances. Ideally, the product owner is right there in the ditch with the team. A distributed team will use a variety of tools to communicate with the product owner, including Skype, email, instant messengers, telephone, and various task or issue tracking tools. A product owner who takes more than 24 hours to reply to an email can easily leave the team in a lurch as the role of the product owner requires being available to make decisions as the developers progress. That said, one of the things I've learned leading distributed teams is that successful team members on distributed teams think ahead. They know what feature or task they expect to do next and they know what they need to do it so that they can ask in the daily standup meeting before they start work on the next item.

They know how to provide constructive criticism. Part of the product owner role is observing development progress and ensuring that user stories are properly interpreted. Sometimes, a story might be properly interpreted by a developer, but when the product owner actually sees it, they realize there is a better way. That's why you choose an agile process, right? Because it's agile. We can change directions (or even horses) in mid-stream. However, that means getting clear and useful feedback from the product owner. I define the levels of feedback usability as:

Criticism: I don't like A.

Useful Criticism: When I do B, A happens and I don't like that.

Constructive Criticism: When I do B, A happens and I'd prefer if C happened instead.

They are organized. During the planning game, especially when the product owner and the development team are in different countries or even just different cities, it is essential to the schedule that the product owner be able to plan their time so that they are available to write stories, respond to email, and participate in conference calls on schedule. If they cannot, then development resources are wasted. Since agile processes avoid scope creep by using fixed iteration dates (timeboxed iterations), any delay manifests itself as reduced functionality (lower velocity) at the end of an iteration.

If the person appointed to play the product owner role is lacking in one or more of these traits, you will still get a stable, high quality, working piece of software at the end of the process, but it is not as likely to fit neatly with both your business case and your users' needs and it will not be built in a way that maximizes the potential efficiency offered by agile development processes.