Week 5 (lecture in detail)

One of the consequences of the Agile principles studied in the previous lectures is a drastic redefinition of the roles that exist in a software project.

This lecture has three segments, and we're going to use these segments to study the roles as they exist in Agile projects.

One of the most important changes affects the role of the manager.

In the first segment, we're going to see what managers, traditionally, in non-Agile methods do.

And actually, they can do lots of different things.

In the second segment, we're going to focus on Scrum, which has been very influential here.

And we're going to see that in a project that is organized according to strict Scrum principles, there's only three roles, and we will study these three roles.

In segment number 3, we're going to broaden the perspective a bit, assume that we're not just working according to strict Scrum principles, and see what other roles can exist in Agile projects, particularly with other Agile methods.

What do managers do?

They manage, right?

Well, the reality is a bit more complex than that. And it turns out that in many projects managers do a lot of things. They are kind of jacks of many trades.

So in this first segment, we are going to review some of the many tasks and roles that managers fulfil in traditional pre-Agile projects.

One of the defining characteristics of the Agile approach, present in all Agile methods, is that they significantly redefine the role of the manager.

What do managers do?

In a traditional setting they do quite a few things. Here is a list of 10 tasks which are typical of what managers do. And certainly managers do most of these things, and in many cases they do all of them.

One of the primary tasks of the manager is to define the goals of the project, and of each iteration of the project. So that's about what a manager does. It's not good enough to promise some results.

We all have finite lives, so we have to define deadlines. And usually customers are very impatient. So that's also one of the manager's responsibilities, to define the deadlines.

Then on a day to day basis, a traditional manager will assign tasks. You do this today, you do that next week.

The manager is also going to be responsible for providing the interface to the upper management.

This also includes often a role of umbrella, of protecting the team from undue interference with upper management.

This umbrella role is of course one of the important roles of the manager.

Not only do we need to interface with management in our own company, we need to provide an interface with the customer.

Now the customer being total. That is to say, another division of the same company.

Or we might even be doing something for ourselves as a team. But most of the time, the customer is well-defined and separate from the team.

And it's part of the role of the manager to ensure that the interface with the customer's organization is smooth, and in particular, that the team is aware of what the customer wants.

That the customer is aware of what the team is doing, and comfortable with it. And of course that, at the end, the results are up to the customer's expectations.

In order for this to work, one must validate requirements. Because, of course, the customer is going to help define requirements.

But at some point, it may be that some of the requirements are unrealistic. And because the manager is responsible for defining goals and defining deadlines, it's his or her responsibility, if needed, to step in and say no, we can't do this.

It's unrealistic. Or we could do it, but not within the time limits specified.

So it's important to validate the requirements. Now the goals have been defined.

And at various intervals these goals will have been realized, or so the team claims.

So part of these goals-- and so it is part of their manager to decide whether these goals have been met, or how much of the goals have been met at a particular point in the development of the project. It's not only goals that count, it's the time at which they are met.

So it's part of the manager's role not only to define the deadlines-- that's kind of the easy part-- but to enforce the deadlines.

It's easy to say this is going to be ready in December, especially if you're in January. It's much harder to make sure that when December comes the goals have indeed been met. And of course, I'm talking about January and December, but typically in an Agile development process, and generally in a modern development process, many of the goals of more short term.

And the deadlines loom on the horizon, so it's important to make sure that these deadlines are met.

Now these are all management tasks in the traditional sense.

In the strict sense of the term managing people, and foreseeing goals and deadlines. But managers in practice do more.

Often a manager is a senior member of the software profession. That's not always the case.

You have managers who are more of the MBA type, who are more pure managers.

But much of the time, in software projects, the manager himself or herself is a software professional.

More experienced, hopefully, than the members of the team.

And so part of the role of the manager is to serve as a coach and to mentor the rest of the team.

A good manager is someone to whom more junior people come when they hit a snag, when they don't know what approach to use for a particular part of the problem.

And as a seasoned professional, the manager is here to coach people and to mentor them.

That's part of the role of many good managers. Finally, any well-organized team, and in general any well-organized company that does software development, will have defined some rules and some methodology for developing software.

House rules, style rules, rules regarding the software process, rules regarding testing, and software quality assurance.

In general, of course, rules are only as good as their enforcement, as their application. So it is part of the manager's role to enforce these rules.

To make sure that they don't just remain ink on paper or pixels on a screen. But that the team actually applies them, and abides by them.

So these are quite a few tasks.

What we see here in this first segment, is that managers traditionally do lots of things.

They have many responsibilities in somewhat derogatory scrum terminology that you find in the scrum literature.

Some of these tasks-- and you can go back to the previous list and decide which are which-- some of these tasks are what the scrum literature call nanny.

Taking care of the developers looking over their shoulders, guiding their steps moment after moment, time after time.

And some, particularly the last two, are more what the scrum literature calls more guru.

And managers do all these things typically.

Part 2: And the Agile approach sets out to change that traditional accumulation of tasks of the manager.

Scrum is very particular about roles in a project, and in strict Scrum development there's actually three roles that have to be fulfilled, not two and not four, but exactly three, and in this segment, the second one of our Roles lecture, we are going to see what these three Scrum roles are.

Scrum's major contributions are in the management area, so it's not surprising that Scrum takes a shot at completely redefining the traditional manager responsibilities that we saw in the previous segment.

In fact, in strict Scrum, there is no such thing, no such role as a manager.

The traditional tasks of the manager are split between three different roles, the self-organizing team, the product owner, and the Scrum Master.

The last two are people.

The first one is a group of people, which collectively takes a certain number of critical decisions.

So let's review those three roles in turn.

The team, as I said, is a group of people, so it's an actor in the same sense as the chorus in a Greek tragedy, but it has responsibility for many of the fundamental managerial tasks.

A Scrum team and in general an agile team, this is a theme that runs through other agile methods as well, in particular Extreme Programming, is cross-functional.

What does that mean?

Well, in the agile world, we don't want to have narrow specialties.

And we don't want in particular to have people who feel they own a particular expertise in the project and as a consequence a particular piece of the code or the product.

We will see this again in the discussion of practices when we review the practices called shared code ownership.

But here what's important is that the team is cross-functional, meaning we don't have anyone who is in charge solely of a certain area.

So we may have experts, for example.

We may have people who know more about object-oriented methodology, about databases, about networks, or any particular specialty that is needed by the project.

But no one owns any particular area.

And this means in particular in the Scrum view that we have a list of tasks to be fulfilled for a certain iteration.

And whenever someone becomes available, he or she takes one of these tasks.

Well, we want essentially anyone to be able to take on any task.

That's what it means by cross-functional.

No narrow specialization, no personal ownership of an area.

Scrum also specifies an ideal team size based on a famous paper in psychology about magic number seven, plus or minus two.

Well, the ideal size of a team is seven, plus or minus two members.

Does this mean that Scrum is limited to small or relatively small projects, medium-scale projects? Not necessarily.

There's a notion of Scrum of Scrums, which we will review also as part of practices, which is intended to tackle big projects. So as the name suggests, a Scrum of Scrums is a group of individual projects that are coordinated at a higher level, but each one of them remains a standard Scrum project, with this kind of number of members.

Part of the responsibility of the team, which of course is very important, is for a particular iteration, not for the project as a whole, but for a particular iteration of the project, to select the major goals for the iteration and the results that are going to be released by this iteration, known in Scrum as a sprint.

So that's of course a key role, especially since it goes with the next two.

The team organizes itself and its work, so you don't have a nanny, so to speak, which tells the team where to go and what to do on a step by step basis.

It's really the team that decides to assign the tasks to its members and organize the work.

The team can really do everything it likes as long as it thinks that is going to help reaching these goals and as long, of course, as what it does fits within the rules, within the guidelines that have been defined for the project.

And of course, at the end of an iteration and maybe even before, in some cases, the team will have to demonstrate its results to the product owner, who, as we are going to see, is the one who decides in the end what passes and what doesn't pass as a suitable result.

So these are very important responsibilities.

They correspond to much of what traditional managers do.

And they are farmed out in Scrum to the collective actor, the collective role of the team.

In meetings of a sprint or in meetings of a Scrum project, you are a going to have to decide who is permitted to talk freely and who is just on the sidelines, because meetings may be of interest not just to the team itself, but to some other participants, for example representatives of other projects with which the team is collaborating.

And there's a risk that the meetings get out of hand.

In order to keep the meetings focused and short, not everyone can speak at any particular time, so there's a distinction between core participants, the team proper, and fellow travelers.

The core participants are those who are permitted to participate actively in the meeting. And the fellow travelers are there to listen.

And they will participate actively, but only if asked. This is also known as committed people and involved people. And sometimes in the Scrum literature, based on an old 1950s joke which I will not repeat here, pigs for those who are committed and chickens for those who are only involved.

The next Scrum role is the **product owner.**

The basic idea of the product owner, who is a person not a group, is that he or she is responsible for ensuring the interface with the customer and making sure more precisely that the product meets the customer's needs and expectations.

So the product owner is going to define the product features.

He or she is going to decide on the release date.

And of course, the two go together, since product features are great, but only if they are delivered within our lifetime, so time is very important.

And we saw the discussion in Kent Beck's citation at the very beginning of the course on this combination between timing constraints and functionality constraints.

The product owner decides also on the content of any major release, not just the final product, but intermediate releases as well, although he is not in charge of individual small sprints, small iterations.

That is, as you remember, the task of the team itself, the self-organizing team.

There's a strong emphasis in Scrum on business aspects and in particular on profitability. A term that recurs in Scrum discussions is ROI, Return On Investment. And the product owner is responsible for that view of the project, which is not necessarily the first one of a typical developer, making sure that a project returns and delivers value to the customer.

The product owner prioritizes features. Of course, in a project typically we want lots of features, even if, as we've seen, agile has a strong minimalistic view, trying to limit the set of features.

Well, still, we might define more features than we can implement.

We might bite more than we can chew. And the need here is for prioritization.

We need to know ahead of time what features are essential and what features are nice to have but less essential, so that when the going gets tough, we know what to sacrifice.

And so this is one of the responsibilities of the product owner, prioritization.

There's a rule in sprint that a product owner can change features and priority, but only over a limited period, typically 30 days.

And the major responsibility in the end of the product owner is to decide whether a particular release or the final product has resulted in something that can be presented, can be handed over to the customer as satisfactory or not. So the product owner is the one who says, yes, this is good enough or no, you need to get back to work.

We cannot give this to the customer. And of course, it's a major responsibility.

The third and last role in Scrum is the **Scrum Master**.

The Scrum Master is a person. Again it's not a group, but it's a person in this case, who is responsible for enforcing the methodology, in this case, of course, the Scrum methodology.

So you can think of him or her as a kind of political commissar who makes sure that the team properly applies the defined philosophy, not only Scrum as a whole, but particular guidelines and rules that may have been defined at the level of the company and at the level of the particular project.

But a Scrum Master actually does more. So here are some of the tasks: to ensure that a team is functional and productive, to enable cooperation across all roles and functions of the team, to shield the team from external interferences.

As we've seen, there is sometimes a natural trend for upper management and other parts of the company to meddle into the daily work of the team.

And that can be quite disruptive.

It's part of the role of the Scrum Master to serve as what I already called an umbrella to protect the team from such undue interferences.

The Scrum Master, as I mentioned, is the political commissar who is there to enforce the Scrum process, in particular to organize the daily meetings, which we'll review as part of the practices lecture, also known as daily Scrums.

The planning meetings, the review meetings, he's in charge of that.

The Scrum Master quite importantly is in charge of helping remove impediments.

Impediment is an important notion in Scrum. It's an obstacle.

It's anything that affects the progress of the team, known as the ***velocity,*** another term that we will study in more detail, in Scrum.

In Lean terms, remember the previous lecture, we had this rejection of waste, this injunction to avoid waste. So we can also define impediments as anything that might produce waste, which of course we want to avoid like the plague.

So examples of impediments are quite diverse, hardware limitations, assume a situation in which compilations and tests are taking too long because programmers don't have enough memory, so the Scrum Master may be the one who says, we need to give everyone an extra eight gigs of memory, because we're wasting time for no good reason.

Another completely different example is missing requirements. We saw this also in the study of waste. You stop because you don't know what to do in a particular area. And the customer expert who was supposed to give you this information is not available and is not giving it to you.

So that's the task of the Scrum Master, to make sure that those kinds of blocks do not occur or are resolved if they do occur.

You might be waiting for some software from some other group or from within the group, some library element that you need for your work and it's not ready yet. That's an impediment. You might have management interference or bureaucratic delays and it's going to be part of the task of the Scrum Master to fight these things and make sure the impediments go away.

In strict Scrum methodology, the Scrum Master is only a Scrum Master and not do development. And so if there is not enough work to occupy a Scrum Master full-time for your project, the idea is that a Scrum Master shares his time between several projects for which he is only Scrum Master.

To go a bit more into the assessment mode here, it's a pretty dangerous idea, because it's not so good to have in a project people who are only talkers and not doers. And we may want everyone to roll up their sleeves and participate effectively.

So there's no absolute rule here.

But the general observation is to be wary of people who are just advice-givers and who don't have to live up to the results of their own advice and apply it as developers.

I saw a comment on a forum-- the precise reference is in the bibliography-- this comment from a reader from India saying:

I've seen the trends that organizations in India look forward to hire people with technical skills, not just methodological, but actual technical programming skills. Especially in India, they do not consider Scrum Master as an independent role, but always clam, that is to say combine it with the developer role. They call it Technical Scrum Master.

I would recommend being careful of hiring people just to give advice. It's probably better if the Scrum Master has extra time to make him or her develop along the rest of the team.

In strict Scrum, there is no manager, no role of manager. The tasks, as we've seen, are split between the team, the product owner and the Scrum Master.

On the other hand, Scrum authors realize that for some companies this is going to be too much and that they will still want to have a project manager. And so they have some advice for managers if there are any:

The managers should do certain things and not do some other things. So they should support the team in its use of Scrum, like the Scrum Master does. They should contribute wisdom, expertise, and assistance, so that's kind of the coach and mentoring role. They should not play nanny. They should not assign tasks, get status reports. Such micro-management is part of the team's responsibilities. Instead they should play guru, mental coach, and so on. So you can see here that we're kind of looking at a manager who is also probably the Scrum Master, at least has some of the responsibilities of the Scrum Master. There's also the advice that managers may need to evolve their management style, for example use Socratic-style questioning to help the team discover solutions rather than imposing a solution, as a traditional Steve-Jobs-like manager or Henry-Ford-like manager might do.

What we've seen in this segment is the three Scrum roles. Scrum, and in general agile methods, strongly curtail the traditional role of the manager.

Part 3: In the next segment, we'll see some other roles which replace part of the tasks in other methods. But in Scrum, we have three well defined, precisely defined separate roles, the team, the product owner, and the Scrum Master.

Not all Agile methods are limited to the three Scrum rules that we saw in the previous segment. And in the last segment of this roles lecture, we are going to study a few other roles that you will encounter in the application of Agile methods. Scrum is not the only Agile method, and other methods, in particular methods that preceded Scrum, have come up with their own definitions of roles, replacing some of the traditional roles of developers and managers and complementing some of them.

An interesting idea from Crystal, which of course is not entirely new since many projects have had in the past something like this, is the ***expert user***. ***An expert user*** is, as the name suggests, a person with expert knowledge of the project area, of the project domain, who can answer questions and even suggest solutions to problems. It's important here to find the right person There's always a risk of getting someone who claims to be an expert and maybe is not. So for example, if someone has too much time on his or her hands, you may be a little bit suspicious because real users typically are very busy, real expert users are very busy with doing work. You also should not just limit yourself to testers from the development team. And so what Crystal advises here is to have a minimum of once a week meetings two hours with expert users and the ability, of course, to make phone calls any time during the week. But of course, this is one of the ways to obtain accurate information about the needs, the actual needs, of the actual users and avoid developing something that will be recognized too late as not fitting those needs.

In XP, in extreme programming, there is a strong emphasis, on the role of ***customers***. We have seen the notion of embedded customers. And we have noted that it's not such a practical idea. But still, the notion of ***customer*** is very important. And in this source, which you will find in the bibliography, there is a number of interesting advice, pieces of advice, as to the responsibilities of the customer in an extreme programming in an Agile project. So for example, customers should not meddle too much into technical decisions. Of course, these days almost everyone thinks they know something about programming and about technology. That may or may not be the case. So it's the developers who are responsible for the technology, not the customers. For all the added roles that we have in Agile methods for the customers, it doesn't mean that customers should be responsible for everything, and they should also know their place. This is basically what this text tells us a bit more politely. The customers are also responsible for helping you analyze the risks. In particular, weighing the various user stories, the requirements elements against each other. It's difficult for the developers to know what is more critical, what is more risk prone, and what is less critical. And here the customers provide a fundamental help. ***The customers are responsible for providing precise user stories so that developers know exactly and, more importantly, accurately what users need.*** They should be the ones who say what are the stories with the maximum value so that features can be properly prioritized. And they should collaborate with the team, not treat the team as an external body, but really attempt to work closely with them and help them at every step.

So in the same style, this same article which I recommend gives a number of elements of advice as to what developers should be responsible for. I'm not going to go through the entire list.

I'm going to refer you to that article. But let's pick a few examples.

Implement only what is necessary. This, of course, is part of the minimalistic spirit of the Agile approach. And the idea here is that you should refrain from your own impulses, which are natural for a developer to want to do, always more to add more features, well, make sure you implement only the features that are fundamentally useful for the customer, and not just all of those that please you for their intellectual elegance and value.

The responsibility of the developers also include ***working closely with customers to understand their stories, not just to get started on the basis of a vague description and make interpretations that may or may not be justified***, but try to understand in depth what a customer really means when he has specified a certain user story. And in general, ***developers should constantly communicate with customers involved in the project.***

It's of course part of the developer's role in this self-organizing team to estimate the work for each story.

Those are some of the examples of the important responsibilities that developers have in an Agile project.

There are also a number of rights.

We've seen notions like ***personal safety and responsibility***. So a developer has a right to give his own estimate for his own work, to work sensible and predictable schedules. Schedule only work that can be done, not have unrealistic expectations which cannot be met and which as a result affect everyone and lower the morale of the team.

Produce code that meets the customer needs.

And it's also important for the developer to be ***shielded from having to make decisions that are really not developers' decisions, but business decisions.*** Of course, the boundary's sometimes fine between strictly technical and strictly business decisions, but we should be aware of these problems and not unfairly force onto developers decisions that are really management or customer decisions or business decisions.

In some sources, you find a special role of **tracker**, who is a person responsible for keeping track of the schedule.

When we talk about artifacts, we'll come back to this notion of velocity, which is the ratio of what the team is actually doing to what it should be doing in an ideal scheme. And so this is something that we're going to track in an Agile project, in particular in a Scrum project. And we may have a special role assigned for this. And it's particularly important then to look for signs of potential problems, like ***changes in velocity, like too much overtime work, or too many tests that fail rather than pass before we move on.*** So these are numbers that measure progress, and it's important for someone to be responsible for measuring that progress.

We have seen the notion of Scrum master, which is a bit specific to Scrum. In general, in Agile methods prior to Scrum, there was a notion such as **coach,** which is a role with the following responsibilities, some of which are familiar because we have seen them in the Scrum master: to guide the team, to mentor the team, to lead by example as a good commander, to teach when necessary, and often to teach just by doing. Sometimes you just watch someone doing something right, and that's the best way in some cases to learn, to offer ideas to solve difficult tricky problems, and to serve sometimes as an intermediary with management. So in Scrum, this is part of what the Scrum master does.

What we've seen in this last segment of our roles lecture is a variety of roles, not exactly identical to those of Scrum, but replacing or complementing many of the traditional software development roles.