Lecture 8 in Detail

To enable the values, the principles, the roles, and the practices of agile development, which we have reviewed in previous lectures, we are going to need some concrete support.

And that support will come in the form of agile artifacts, which are the topic of this lecture consisting of just one segment.

Some of the artifacts we will talk of-- virtual artifacts are quite abstract.

We could also say intellectual artifacts.

They're just notions. Other artifacts are concrete, quite tangible. We will talk of physical artifacts.

In practice, of course, many of the physical artifacts are there directly to support the virtual artifacts.

So from user stories all the way down to task boards, we are going to see what agile artifacts are.

The artifacts of agile development support the practices and principles that we have seen in previous lectures.

There are of two kinds.

Some are completely intellectual-- say, virtual.

And some are physical-- say, material.

In this presentation of agile artifacts, we are going cheerfully to mix the two because they're closely connected.

But for example, the storyboard is there directly to support the notion of user story.

So we're going to have an alternation of virtual and material artifacts.

Indeed, we start with user stories.

Well, we have already reviewed user stories, so this is just a reminder.

Remember that a user story defines an atom of functionality.

And it's often in the style "as a" "some rule" I want to some task so that some business goal.

So we've seen this in some detail. We've also seen the limits of using user stories as a tool to define requirements.

To express uses stories, historically, agile projects have used so-called story cards, of which an example is given here.

It's a kind of historical example because it's an example that comes from the original extreme programming project, the C3 project, as it was called, which was a payroll project at Chrysler.

And well, there's nothing that surprising here. It's a standard office card on which the various elements of a user story are codified.

Many projects still use this kind of paper-based description of user story, story cards.

On the other hand, you can find on the market many software tools that give you the equivalent with computer support.

Now, this is for user stories. And you have the equivalent for tasks.

Remember that when you're implementing an iteration, you are going to introduce to define a number of tasks for each user story.

And in the same way that we need to describe a user story and follow its progress, we need to do the same for a task card.

So here, you have, again from the C3 project, a task card.

Along with user stories, you may have heard the term use case.

Now, use cases are actually used far beyond the agile methods.

They are one of the diagram types in the UML analysis and design notation, Unified Modeling Language.

What's the difference between a use case and a user story?

Well, the general idea is the same.

In both cases, we are describing some piece of functionality by explaining the interaction of an actor, in particular, a human user, with the system.

So use case is often defined as shown here. It describes how to achieve a single business goal or task through the interactions between external actors, such as people, and the system.

A typical example of three use cases is this one here having to do with some e-commerce system where you see the various steps that an online customer can go through or will go through in this particular use case-- view the products, then purchase the products, then manage the customer information, and then create a product review.

And another use case involves another human user, marketing administrator.

And you also have use cases that involve other actors on the right.

The typical way of differentiating use cases from user stories is to say that use cases are bigger than user stories; here is a more specific definition of the distinction by Alistair Cockburn.

It's not necessarily everyone's definition, but it's quite interesting to look at.

He says it's not so much a matter of scale as a matter of granularity of the description.

Think of a user story, he says, as a use case with much less precision.

So more concretely, a user story is simple where a use case is more complex.

The user story is written by the customer whereas the use case will be written by the developer.

Well, the first assumption may not be realistic because not all customers can or want to write the user stories themselves.

So that even in the first case, we may need the help of the developer.

More important is the difference in accuracy and completeness.

The user story is a first step.

It could be inaccurate.

It could miss exceptional cases.

On the other hand, the use case should include enough to be taken as the reference for development without requiring further interactions with the customer in normal cases.

So the user stories, in this view, are really starting points for discussions.

And the use cases are starting points for the development, meaning end points for the requirements part.

Another important artifact coming from Scrum is the product backlog. So it's a list, so it's another virtual artifact. It's a list of what?

It contains backlog items.

And a backlog item is a description of potential features.

So it could be, for example, user stories.

But it could also be broader than that.

And usually, it's at a higher level of abstraction.

It's more a description of functionality, after functionality, prioritized by business value.

So is the key criterion here is the business value.

And indeed, the items and the product-- and indeed, the product backlog is a property of the product owner, who, as you remember, is, in Scrum, the person responsible for defining functionality for customers.

It is, on the other hand, open and editable by anyone in the open tradition of agile projects.

It includes estimates of business value.

But it also includes, for every element, an estimate of the development effort.

Often, of course, software development is a trade-off.

When you choose what functionalities to develop, you are going to look at how much each functionality is worth to the customer and how hard it is to implement.

Of course what we all like is the low-hanging fruit, functionality that is of very high business value and low development effort.

But it doesn't always happen that way in practice.

And so we have to make trade-offs. And one of the purposes of having the product backlog is to make sure that we can make these trade-offs explicitly by having those two estimates for each backlogged item.

Now this is, as I mentioned, a virtual concept.

And we need often a visual representation of it, a material representation of it. And that is the role of the task board. So a task board, or a storyboard-- there's several variants here-- is a way to visualize the progress of a particular iteration of the project or of the project as a whole by showing the various elements visually on a board.

So the board could be a metal board on which you put magnetic elements.

Or it could be just a plain wooden board or any other material on which you put Post-Its as in this example here. And the general idea is that these items are going to move from left to right-- the Post-It notes, for example.

So the typical organization is the one shown here.

This is a storyboard.

So you have the stories in the first column, and the rest is tasks for these stories.

You can see here that we are showing the progress for two user stories.

And in columns two to five, the elements are tasks.

So you can see that the number of tasks in the second column are still to be done.

They haven't been tackled at all yet.

The other extreme in the last column, you have tasks that have been completed.

And of course, seeing them is a really good way to boost the morale of the team to show what has been already accomplished, although, of course, this could promote some humility because we also see everything that remains to be done.

In the third column, we have the list of tasks that are currently being worked on.

And in the next column, which is not always there but which it is a good thing to have, we have the tasks that have been finished but not yet verified. And of course, they're not really ready to be included until they have been tested and more generally verified.

And so the benefits include, in addition to the two already mentioned, transparency.

Everyone sees what's going on.

And everyone has a good picture of the progress of the project, and that includes more technical people and more management oriented people.

It fosters collaboration.

Because if you see something on which you might help in one of these columns here, you are encouraged to help. It helps prioritization. It helps focus.

It helps people and the teams self-organize.

It helps basing decision on empirical criteria about the real progress of the project rather than on the illusions or wishes.

So these are the advantages that are announced for this approach, which is widely used.

The next artifact is again a virtual one, which will have a material counterpart for the one following it. It's velocity. A really difficult problem in software is to try to measure numerically how far we are in the development and how fast we're progressing.

And Scrum's velocity is a really valuable attempt to quantify this notion of progress.

Now, it's not really a velocity.

It's kind of a misnomer. Because a velocity is some kind of progress over time.

It's a ratio. Well in velocity, there's no, in the current concept, no notion of time.

It's just a measure of progress.

And it's very simple.

It's a number of items delivered.

So depending on the measure you choose, it could be the number of tasks completed, the number of users stories implemented, the number of backlog items that have been taken care of.

But the general notion is the same.

It's that you keep track of that.

And how do you keep track of it in practice?

Well, there's a very interesting notion.

It's the material counterpart to the virtual notion of velocity.

It's the burn down chart.

So it's a very simple idea.

It measures progress by tracking velocity-- or rather, the number of elements that have not yet been implemented.

So if we had a completely regular process where the velocity, so to speak, is constant, then we would have the blue line shown here.

In practice, of course, it's going to be sometimes below, sometimes above the blue line.

But the idea is that each day-- and this is a daily process-- we track.

Remember that one of the roles in some agile projects was the tracker.

We track how much remains to be done as a result of tracking how much has been done.

And if we are below the blue line as at the beginning of this hypothetical project, we are progressing faster than we have to.

And if we are above the blue line, we are progressing more slowly.

In principle, the curve should not be increasing.

Because if it increases, it means that either we have found that some supposedly implemented functionality was not implemented properly and needs to be re-implemented.

Or it means that someone has added elements to the backlog which, as you remember, during an iteration, is a no-no. But of course, in practice, things are not always optimal, and we might have, at some point, the curve increasing.

So this is a really great idea even though it is very simple, because it is one of the first attempts in software engineering to have a simple and clear and effective way of tracking progress in a precise and particularly quantitative way.

One practice that we saw in the discussion of practices was the idea of informative workspace.

And open workspace was a related principle in extreme programming in Crystal in Scrum.

And so as an artifact and to finish this discussion, we may, for the record, mention something called the bullpen, which is a big artifact in this case, and a material artifact.

It's the place where people work. And as you know, in agile methods, we favor communication very much.

So there's this idea of a single open room where everyone can, whenever needed, interact with everyone else.

So what we've seen in this discussion of artifacts is a whole set of virtual and material tools often paired with each other, a material tool directly supporting a virtual tool.

And they directly support agile concepts-- in particular, agile principles, agile practices, and agile rules.