Introduction to this Course

Agile methodology has taken the software development industry by storm.

Everyone wants to be agile, but what does it really mean and how do you achieve agile development?

This computer science course cuts beyond the agile methodology hype and teaches you the fundamental agile concepts that span a wide range of methodologies.

It analyses the key agile ideas, their benefits, their limitations, and how best to take advantage of them to enhance your software skills and show employers that you have mastered an essential component of today's IT industry.

The course is divided into six parts:

1) The Agile manifesto and the context of agile methods

2) Agile principles: what key methodological ideas underlie the agile movement?

3) Agile roles: how does agile redefine traditional software jobs and tasks, in particular the manager's role?

4) Agile practices: what are the concrete techniques that agile teams use to apply these methods?

5) Agile artifacts: what practical tools are essential to the work of agile developers?

6) Agile assessment: among agile ideas, which ones are essentially hyped and useless, which ones are actually harmful, and which ones will truly help you effectively produce high-quality software?

Benefits of this course:

This course takes a strictly objective view of agile methods, enabling you to retain the best agile principles and practices.

By analyzing agile methods in depth and showing you how to benefit from them, it will make you a better developer, equipped to deal with the challenges of ambitious software projects.

What you'll learn?

1) The Agile manifesto and the context of agile methods

2) Agile principles

3) Agile roles

4) Agile practices

5) Agile artifacts

6) Agile assessment

Course Description

Chapter 1: Context:

Introduction to the agile context: manifesto, principles, methods, and values.

Chapter 2: Principles

Detailed discussion on the agile principles.

Chapter 3: Roles

Detailed discussion on the roles in an agile project, and how they evolved from the traditional roles.

Chapter 4: Practices

Presentation of the main agile practices, namely meetings, development, releases, testing, management.

Chapter 5: Artifacts, assessment

Description of the main agile artifacts .

Chapter 6: Assessment

Final assessment on the agile methods.

Final Exam

GRADING POLICY

The sum of the points achieved in the Assigments ("Quick questions") altogether is worth 10% of the final grade. Each correct answer to a quick question is generally worth 1 point.

The sum of the points achieved in the Even Weeks Lectures time ("Homework" quizzes) altogether is worth 30% of the final grade. A correct answer to each quiz question is generally worth 1 point. Occasionally, there are questions that are made of several sub-questions, each worth one point. One quiz is droppable (won't be graded).

The sum of the points achieved in the final exam is worth 60% of the final grade.

Students who achieve at least 40% of the available points (computed considering the appropriate weights described above for quick questions, quizzes, and exam) will pass the course.

SUBMISSION DEADLINES

Please follow the weekly schedule I provide on cs moodle.

Lecture 1: Content

Context

Introduction to week 1

The Agile Manifesto

Agile Methods

Official Agile Principles

Agile Values

Assigment 1: Context quiz (to be uploaded on cs moodle soon)

Introduction to week 1

Agile processes are one of the most important developments in two decades of software engineering. And I'm not exaggerating.

Everywhere in the industry, people are looking at agile methods and often practicing them or some form of agile development.

Every software employer is going to give preference to candidates that master agile concepts.

In this course, we are going to review agile processes so that you can take the best out of them.

Now, many presentations of agile methods are partisan. They really want you to kneel down and to start praying and adopt this or that agile method.

This one is different. I'm not preaching one method or another.

The only thing I'm really preaching is software quality and software productivity.

So we're going to see what agile processes are about. But we're not going to advertise for a particular method. And in fact-- surprise, surprise-- in agile methods, there are some good aspects and some not so good aspects as well, as in every human endeavour.

And it is very important to keep one's cool and to decide what is best for you, for your project.

So this is going to be an analysis, without any bias, of agile processes as they exist today.

The course is divided into six parts.

The first part presents the **agile manifesto** and the context of agile methods.

The second part presents the **agile principles** what key methodological ideas underlie the agile movement.

The third one presents **agile roles**. How does agile redefine traditional software jobs and tasks, in particular, the manager's role?

The next part, the fourth one, highlights **agile practices**. What are the concrete techniques that agile-type teams use to apply the methods?

The next, and fifth, part describes the **agile artifacts.** What practical tools are essential to the work of agile developers?

And the final part is an **agile assessment.** Among agile ideas, which ones are essentially hyped and useless?

Which ones are actually harmful?

And which ones will truly help you produce software better?

In the words of the title of the textbook that goes with this course, we look at the good, the hype, and the ugly.

I hope you enjoy this course.

Agile methods are one of the most important ideas on the software scene today.

This course will enable you to master them and decide for yourself what they can do for you, for your projects, and for your career.

The Agile Manifesto

This first lecture is called context, and as the name suggests it introduces the fundamental ideas behind Agile methods and the general background.

There's four segments. The first segment is about the **Agile manifesto**. That is, the foundational text that at the beginning of this century introduced the fundamental Agile principles to the world.

Out of the manifesto, came a number of full-fledged methods. And these methods are the topic of segment number two. The methods embody the **agile principles**.

In segment three, we take a look at the key **Agile principles**, as described in the Agile Manifesto. That is to say, as seen by the originators of the Agile ideas. Now, this is not necessarily, as we will see later the best way to present Agile principles. But this is what we will look at in segment three.

And finally, in segment four, we will review **Agile values**. The values are at a more abstract level than the principles. They are a kind of general philosophy that underlies the entire Agile movement in from which everything else follows.

The Agile Manifesto is the 16-year-old text which first presented Agile ideas. And it's remarkable to see how aptly it still characterizes the Agilist's view today.

So let's see what it has to say. Agile methods came into prominence at the time of the publication of the so-called Agile Manifesto in 2001. This was, as the name suggests, a pamphlet, a manifesto, designed to attract the world's attention on the need to develop software differently.

The manifesto was the work of a number of people. The original signatories of the manifesto, they're listed here. Most of them, perhaps all of them, are in fact software consultants. And the manifesto gained a lot of visibility, and has continued to be quoted widely since then. You can find it online at agilemanifesto.org.

And here's what it says. We need to go through it in order to understand what Agile is about, as seen in the eyes of its own creators. It's a text that contrasts certain old ways of doing software, shown here on the right.

At least viewed as old by the authors of the manifesto with a number of alternatives. They take care to indicate that they do not completely reject the stuff on the right. They say while there is value in the items on the right, we value the items on the left more. We are uncovering, they say, better ways of developing software, not just by talking about it, but by doing it. By actually developing software, we are practitioners. And by helping others as consultants to do it. Through this work, we have come to value the stuff on the left over the stuff on the right. Individuals and interactions over processes and tools. Working software over comprehensive documentation. Customer collaboration over contract negotiation. Responding to change over following a plan. So you can see the difference between the stuff on the left and the stuff on the right. The stuff on the right is really traditional, or viewed as traditional management-oriented techniques. Techniques that emphasize projects that are strictly

managed. And the stuff on the left is more into individuals and collaboration and informality and people. So going through the four elements here.

Individuals and interactions rather overly managed processes and tools.

Working software, second point, rather than comprehensive documentation.

There's a tendency in traditional projects to insist on documenting absolutely everything that the software is going to do.Well, the Agile view here is to say documents are only documents. And really what counts is software that actually runs. This is the last word which solves all questions. Documents only say what the software should do. Software, by definition, says what the program does.

3, customer collaboration over contract negotiation. Of course, you need to make sure that your software satisfies the needs of the customers. You can go have the lawyers involved. You can go weeks describing and discussing exactly what it is that the customer wants, and accepting it or not. It's much better to involve the customers, to involve the people representing the business from beginning to end in the project, in the development.

And finally, responding to change over following a plan. One of the characteristics of software, and of course we haven't waited for the Agilists to tell us that, is that software changes. Hence the word soft in the very name of the field. Well, what's the Agilists emphasize here is that plans are great. But in the words of the German strategist Helmut von Moltke, the best battle plan doesn't survive the first contact with the enemy. And it's the same thing in software: requirements and advanced descriptions of what systems should do are great. But as soon as you start implementing stuff and showing it to customers, they're going to want something different. Hence, what is really more important than following a plan is to be able to respond to change.

Slide (9) the 12TH PRINCIPLES

These general ideas are complemented in the Agile Manifesto by 12 principles. And we're going to go through them, because this is necessary to understand what Agile is about in the very words of its own creators.

First, our highest priority is to satisfy the customer through early and continuous delivery of valuable software. There are several ideas here. We need to satisfy the customers. So the prime role of business representatives, of people who are going to use the system in the end, and the emphasis on early and continuous delivery of valuable software. Software that doesn't just run, but does something useful. And it has to be early. That is to say right from the start, we're going to start producing software rather than producing documents, or requirements, or designs, or architectures. And it's going to be continuous. That is to say, throughout the development, we're going to produce software again and again.

2Th, welcome changing requirements. Well, between you and me, welcome is a little bit strong, because very few people welcome having to change what they have already done. But the idea is clear. It's that software is soft, and we have to accept rather than welcome perhaps, that requirements are going to change even late in development. Even when you're ready to ship, or you think you're ready to ship. Agile processes harness change for the customer's competitive advantage. So psychologically, this is quite important here. Change is not just something we accept as a necessary

evil. It's something that we welcome, in their words. Because being able to react quickly to requests for changes is going to give the business a competitive advantage.

3, deliver software frequently from a couple of weeks to a couple of months, with a preference to the shorter time scale. That is to say, weeks rather than months. So we are done with the old style of going away for months to work separately in the various parts of the system, and then trying to merge everything. Instead we are going to have very short iterations where we put everything together.

4, businesspeople and developers will work together daily throughout a project. So it's not that we talk to the customers, to our users at the beginning to define the requirements, and at the end to check that we met the requirements. We're actually going to involve them throughout.

5, build projects around individuals. So this is the emphasis on personal contact, and on individuals. In particular, individual developers and supported developers. Give them the tools, give them the support that they need and trust them to get the job done. Emphasis on trust.

6, the most efficient and effective method of conveying information to and within the development team is face to face conversation. Again, emphasis on personal interaction.

7, this is really a repetition because it has been said already. Working software is the primary measure of progress. Documents, requirements, designs, and so on are great, but they are not what is going to show you that you really progress. In the Agile view, the only measure of progress, or at least the primary measure of progress is the working software that you have actually produced so far.

8, promote sustainable development. Don't overwork developers. It doesn't work in the long term. Make sure that everyone can maintain a constant pace indefinitely.

9, technical excellence. Emphasis on making things right, on doing things right, and on having good design and high quality code.

10, simplicity, which is defined here as the art of maximizing the amount of work not done. We'll come back to that. It is essential. So emphasis on finding the simplest possible solution.

11, the best architecture's requirements and designs emerge from self-organizing teams. So the idea is that we should not over manage developers, but we should trust them to organize themselves. And of course, these are all points to which we're going to go back throughout this course.

And finally, **number 12**, at regular intervals, meet and discuss what you have done in order to reflect on the successes and failures, and in order of course to adjust the team's behaviour accordingly.

This is Agile as viewed by its creators of course, at a high level of abstraction. And we're going to see the details of the practices, artifacts, and other techniques that make it possible to put these general ideas into practice.

So what we've seen in this first segment of our first lecture is a set of principles defining Agile as seen by its creators. It's actually quite a few principles, 12 of them. And they give us the general idea. To benefit from this course, it's going to be useful for you to have access to the corresponding textbook. It's called Agile, the Good, the Hype, and the Ugly. And in particular, if you want to go deeper into some of the material that I will cover in this course, the book will be useful.

Agile Methods

Agile ideas have been combined into a number of full-fledged methods, sometimes called methodologies. And in this segment, we're going to study four of them. They're not the only ones. There are a number of them out there, but these four are the ones that have garnered the most attention. We will start with extreme programming, which was only the first to hit the public. Then we'll continue with Lean Software and with the Crystal methodology. And we will end with the one that is the most visible today, that is really dominant on the scene, namely Scrum. Agile development is generally practiced in the form of one of the so-called agile methods. Agile methods including in particular, the four listed here. And there are a coherent sets of rules and practices that all draw their roots in the general ideas that we've started to see.

The four best known ones are XP, which stands for Extreme Programming, and whose author is Ken Beck. Lean Software was devised by Mary Poppendieck with her husband Tom Poppendieck. Crystal was devised by Alistair Cockburn, and Scrum by Ken Schwaber and Ken Sutherland. I've given here the names of the creators of these methods, because there's a strong personal connotation to each one of these methods that is closely connected to the personality of their creators and to the books that each one of these have written about the corresponding methods.

The first one to gain wide attention is **XP Extreme Programming**, which came out in the 90s. And XP Extreme Programming can be seen as a reaction against the culture that was predominant in software engineering circles at the time. A culture that emphasized process, that emphasized plans, diagrams. Things like UML, or CMMI. UML is the Unified Modeling Language. It's in particular, a diagramming convention for describing systems. CMMI, which is the Capability Maturity Model Integration, is a standard for defining best practices in software development. And these are all approaches that emphasize management, that emphasize process and documents. And extreme programming was in part a reaction against this. Drawing attention to the fact that what really matters in the end is the programs, and of course, the programming and the programmers. In the end, it's not diagrams that work or not. It's not documentation that tells you whether you have a working system. What counts is the software, the programs as we have already seen, in the principles of the Agile Manifesto. So Extreme Programming made a major contribution by rehabilitating so to speak, the work of programmers, and putting programs and programming at the center of software development.

Lean Software is an attempt to apply to software a number of ideas and principles which have proved their value in other engineering fields, in particular, in the car industry. There's the famous set of practices developed in particular, by Toyota in Japan, which have been very influential, not just in the automobile industry, but throughout industry, in particular industries, that make material things. And the Poppendiecks applied these ideas to software, emphasizing in particular the need to get rid of what they call waste. While waste is clear in industries that make material things. But the Poppendiecks claim that we should also be on the lookout for waste in software, and get rid of things like for example, useless documentation, which they view as waste, in order to concentrate on stuff that is going to be actually delivered to the customer.

Crystal is a little bit different from the others in that it is more of a combination of agile ideas and more traditional ideas. Crystal is actually the name for a set of methods characterized by various degrees of importance of process in management. Alistair Coburn, who developed the Crystal set of methods is trying to combine the best of Agile with a best of more process oriented approaches.

Scrum is the approach that has come to dominate the scene in recent years. And Scrum is less technical than, for example, the XP. It's not so much software-focused. It's more of a management method which emphasizes ideas that we are going to study in detail, such as the importance of self-organizing teams as opposed to teams that are closely managed by a manager, and the importance of a specific kind of short release iterations known as Sprints. It's important to note that in practice these days when we talk about Agile, we really talk about Scrum. This is the approach that officially at least, most organizations using Agile methods have really adopted. And it has somehow on the surface at least, wiped out all the others. This view however, is not entirely accurate. Because when you look under the surface, and when we you see what people who say they're using Scrum are actually using, well, they really use a lot of ideas from the other methods, in particular, XP. Especially when it comes to the more technical aspects of the process, many of the XP ideas, even if people don't necessarily realize that they come from Extreme Programming, have made their way into the standard form of Agile methods usually known as Scrum. When we continue at a high level of abstraction, at a bird's eye view of what Agile methods are, which is our purpose for this segment, it's important to know that initially with XP, one could view the use of Agile methods as a reaction against Dilbert's boss. So, so to speak, against the management aspect in favor of the view of the Dilberts of this world. And it is this rehabilitation of programming that I have talked about.

There's also another aspect to Agile methods which is not that emphasized in the Agile literature, but which it is really important to understand if you want to know what Agile is about. It's the idea that I call negotiator scope. It expressed quite clearly in one of the books of Ken Beck about Agile. By the way, you can check the bibliography. When I say Beck 05, I refer to a particular book. And the bibliographic list is in the materials for the course. Also, while I'm at it, you may have noticed these little symbols, such as shown here. These are not official symbols for the methods. These are mine, but they will be used whenever I present an idea that is particularly associated with Extreme Programming, with Lean, with Crystal, or with Scrum. So coming back to the negotiator scope contract idea, what Beck tells us is write contract for software development at fixed time, cost, and quality, but call for an ongoing negotiation of the precise scope of the system. And so what this means in plainer English is that in the ideal world people often renounce the traditional commitment to both quality, functionality, and to do all three of quality, functionality, and delivery. In particular, functionality and delivery date. So the general idea is we can promise you a certain delivery date, or we can commit ourselves to a specific set of functionality. But it is very difficult to commit on both. And so this is very much the world as viewed by consultants. And it is one of the implicit characteristics of Agile methods, while not completely implicit since Beck explains it here. But that is going to be one of the points that we will have to discuss when we assess the value of Agile methods, and how much one should retain in practice.

So what we've seen in this segment is that a number of Agile methods have emerged. I've cited four of them. Extreme programming, Lean Software, Crystal and Scrum. They shared the basic ideas, but differ in their goals and points of emphasis.

Official Agile Principles

An important part of the agile manifesto, making up the entire second page of the text, is a set of principles, 12 of them, which reflect the agilist view of the essential ideas of the agile approach. In this segment, we are going to review these principles. To improve our understanding of agile ideas, principles, and practices, let's go back to the official agile principles that we saw in the first segment. It's going to appear that as a way to understand the approach in depth, it's not the best possible source. Of course, this may sound pretentious or even arrogant since these principles were written by the originators of the agile approach themselves. So how can one say that he or she understands an approach better than its creators? But in fact, it's not so surprising first, of course, the manifesto is 15 years old and things have evolved since then. And maybe we understand things better. But more importantly, the originators of an idea are not necessarily the ones best equipped to explain it to the rest of the world. And we're going to see that this description is not the best possible one. It suffers, in particular, from a number of deficiencies.

SLIDE 21

First, some of the claimed principles are not principles but practices. **A principle** and a practice are not the same thing. A principle is general and abstract. **A practice** is regular, practical, and concrete. So for example, if I tell you, set some money aside for your old age, that's a principle. If on the other hand, I tell you, every month put 7.5% of your monthly earnings into a savings account, that's a practice. Now the practice is there, obviously, to help satisfy the principle, but it's still quite different-- one, abstract and general, the other one, concrete and practical. And so when we say, deliver software frequently, from a couple of weeks to a couple of months with a preference for a shorter time scale, this is a practice. It's not a principle. When we see at regular intervals the team reflects on how to become more effective, then tunes and adjusts his behaviour accordingly, again, it's not a principle. It's a practice. Now some of these principles are not practices but assertions. A principle should be prescriptive. **A principle** should tell you to do things in a certain way. **An assertion**, on the other hand, is a statement about the world. So here when we say, the most efficient and effective methods of conveying information to and within a development team is face to face conversation, well, this is not a principle. It's a statement about the world. It's an assertion. Of course, it could be turned into a principle by saying have face to face conversations often or something like that, avoiding, of course, turning it into a practice. But that's not the way it is written here. Another assertion is number seven, working software is a primary measure of progress. OK. Well, we could again turn this into a principle, but it's not a principle. It's an assertion. Another one is this number 10-- simplicity, the art of maximizing the amount of work not done is essential. Well, statement not principle. Also in a foundational document like this one, we might expect to see a set of principles which are individually separated, in which each provide a basic idea rather than repeating the ideas expressed by previous items. But here we have redundancy. For example, number one says, early and continuous delivery of valuable software. Number three says, deliver working software frequently. It's just repeating the same thing. Of course, sometimes it is good to repeat things but not in a foundational document that should express a number of independent

principles. And as if this were not enough, number seven repeats essentially the same idea once more-- working software is the primary measure of progress. I think we understood with number one and if not, with number three. The principles are also remarkable by what's absent in them. For example, in all agile development, testing plays a major role but there's not a word in these principles about testing. It also doesn't help that some of the claimed principles, like number 10, in addition to be assertions are also wrong. If you assert something, you'd better be right. Now here it says, simplicity equated to the art of maximizing the amount of work not done-- well, it says it's essential. Sure, simplicity may be essential. The art of maximizing the amount of work not done it is also essential. But they're two completely different things. Simplicity is not obtained, in general, by working less. Anyone who has worked on difficult engineering or scientific problems-- in fact, on many kinds of problems-- I think that writers, novelists, would probably say the same thing-- knows that simplicity is rarely attained the first time around. Very often when you come up with a first design, that is too complex. And you realize it's too complex. And you work on it until it is simple enough. So simplicity is not obtained by not working. Simplicity is obtained by working more.

SLIDE 22

This was expressed quite well in a famous text by the French writer, Antoine de Saint-Exupery, in the 1930s. He was talking about aircraft construction. And he uses terms like calculators, which certainly meant something different than what it means today. A calculator then was a person. But still it's quite applicable to software. It seems, he writes, that the sole purpose of the work of engineers, designers, calculators, is to polish and smooth out, lighten the seam, balance that wing-- it's a beautifully written text-- until it is no longer noticed, until it is no longer a wing attached to a fuselage, but a form fully unfolded, finally freed from the ore-- a sort of mysteriously joined whole and of the same quality as that of a poem. It seems that perfection is reached-- this last sentence says it all very well. It seems that perfection is reached not when there is nothing more to add, but when there is no longer anything to remove.

SLIDE 23

Now if you want the same general idea expressed by someone who is much closer to us, both in time and in discipline since, of course, he was a computer or an IT person or a software person even, Steve Jobs, in a also famous interview to Business Week's in '98, said that's been one of my mantras-- focus and simplicity. And simple can be harder than complex. You have to work hard to get your thinking clean, to make it simple. But it's worth in the end, because once you get there, you can move mountains. So this is another example of why the principles are not quite right.

SLIDE 24

And in the next segment, we're going to see how better to describe the agile approach, starting, of course, from the manifesto and from the principles expounded there, but trying to be more precise and more effective in capturing, in general but also in detail, what agile methods are about. So what we've seen is that the official principles give us a good starting point for understanding agile. But they fall short of explaining exactly what the method is.

Agile Values

The official agile principles which we have just reviewed are fundamental of course, because they come so to speak, from the horse's mouths. From the very people who originated agile ideas. On the other hand, they only provide one perspective and we're going to need a more general perspective. In fact, our description is going to be based on a number of levels, and the first of these levels is the values level. What are values? Values are the general philosophy that underlies the agile approach and from values follows everything else….Agile principles, agile roles, agile artifacts and agile practices. As we now prepare to study in more depth and detail what agile development truly is about, we've seen in the previous segment that the official agile principles do not really give us enough. They are helpful of course, very helpful to get started but they do not provide the precise guidance that we need. What we are going to use instead is a description along several axes, six of them in fact. Values, principles, rules, practices, and artifacts and this is what the rest of this course is going to explore before we come into the final assessment of agile ideas.

SLIDE 26

The values are general ideas that underlie the agile approach. They're more like a philosophy, a vision. They're not yet principles that are precise enough to be used as principles but it's fundamental to understand them if you want to understand the rest of the agile approach. Then come the principles and we are going to have two kinds of principles: **managerial and technical**. **Managerial principles** apply to the management of software projects. **Technical principles** have to do with more software specific aspects.

Then come **roles**. It turns out that agile lessors define new roles in a software team and we defined some of the existing roles such as, the **role of the manager**. Then come practices and here again we have two sub-categories, **managerial and technical**.

The practices are prescriptions as to what you have to do regularly in order to be an agile team and some of them have to do more with **management of software projects** and they're really independent of the application area then of software specific, they could be used in other areas. And the **technical practices** like for example pair programming, are strictly applicable to the software area.

And finally, it turns out that agile methods often use some specific **artifacts**, either virtual or quite material. For example, we're going to use certain kinds of borders to record progress and we are going to have some kind of so certain techniques for interacting between the different members of the team and those are going to be the artifacts of the actual approach.

So in order to get a complete description of the approach, we will have to study these six dimensions and to conclude this first lecture, let's look at the agile values.

SLIDE 27

So once again they are general ideas that are directly applicable as principles, that's coming in the next lecture, specifically devoted to principles. But they are the general vision, the fundamental philosophy that defines agile methods and that is common to all such methods. New reduced role for manager, no big upfront steps, iterative development, limited negotiator scope and focus on quality, achieved through testing.

The first one, new reduced role for manager, is quite important. Every software project traditionally at least, has a manager and often the manager plays many different roles. The manager is the person who assigns day to day tasks to developers and other members of the team. The manager is the one who is often responsible for the successful completion of the project. And in particular, responsible for making sure that the customer's needs, the users' needs, are met and as a result, he or she is going to be tasked with assessing whether a particular version of the development is satisfactory or not. Managers also often serve as mentors, as coaches, they provide interface to the rest of the organization and to the client organization. These are some of the many rules that traditional managers exert and what we're going to see with agile methods is that they drastically reduce the role of the manager, in particular managers in agile approaches do not assign tasks to project members. This may sound surprising if you haven't seen agile methods before, but this major traditional task of the manager is removed from the manager in agile approaches. And is handed over instead to the team, the self -organized team which takes care, according to the agile credo, of assigning tasks to each member, to its members.

B, no big upfront steps There's a very strong rejection in the agile world of steps that you have to perform at the beginning of the project. Traditionally, projects are going to do, at the beginning, a step of days, weeks, or sometimes months, of defining the requirements for that system. What it is that that system should do, another front step is design or architecture. These have often a significant amount of time, again weeks or months devoted to building the software architecture and the system architecture of the system under development. This is very much frowned upon and in fact rejected by agile approaches, which instead want the project to start coding right away, to start producing working software right away. It's not that they reject tasks like design or like requirements, but they made them continuous. They want you to do requirements as you go and to do design as you go.

C Interactive development. We have done in agile approaches with the old style of having very long phases with reconciliation of the different pieces of the development at the end of such a multi months phase. Instead, the development in agile methods is iterative, it goes step by step and each step is typically a few weeks or even a few days or even in some cases in some extreme variants of agile methods, one day. A more typical length for an iteration is a few weeks typically a month, but this is very different from traditional approaches in which the various members of the team went away for a long while and reconciliation happened at the end. Here, the development is step by step, very progressive with short iterations.

D Limited negotiator scope, there's a very strong emphasis in the agile view on making sure that the system does only what's needed. So as to limit waste in Lean terminology that is to say so as to avoid spending time on tasks that do not produce immediately visible added value for the customers, for the future users. So we are going to limit scope in various dimensions which we will see in the next

lecture. But in particular we are going to make sure that we only build functionality that is strictly needed.

And finally, **the focus on quality achieved through testing**. This, as we've seen a focus on excellence in agile development and the idea of quality in agile methods is very much identified with the idea of testing. And testing plays an important role in all the methods in particular through the notion of regression testing.

So these are the five values that underlie absolutely everything that we are going to see about agile methods of course. We are going to see more concrete, more practical realizations of these ideas through the principles, roles, practices and artifacts. But to understand the detail of the specific techniques it's essential to understand what the underlying values are.

SLIDE 28

What we've seen in this last segment of our first lecture is that beyond specific principles, practices, roles, and artifacts, agile ideas are defined by general values representing a general vision, a general philosophy of how software development should proceed.