In this lesson, we're going to learn about the

Agile Manifesto, which is kind of a foundation of Agile.



The Agile Manifesto was written by 17 individuals, and it consists of 4 values and 12 principles.

Let's look at the values first.

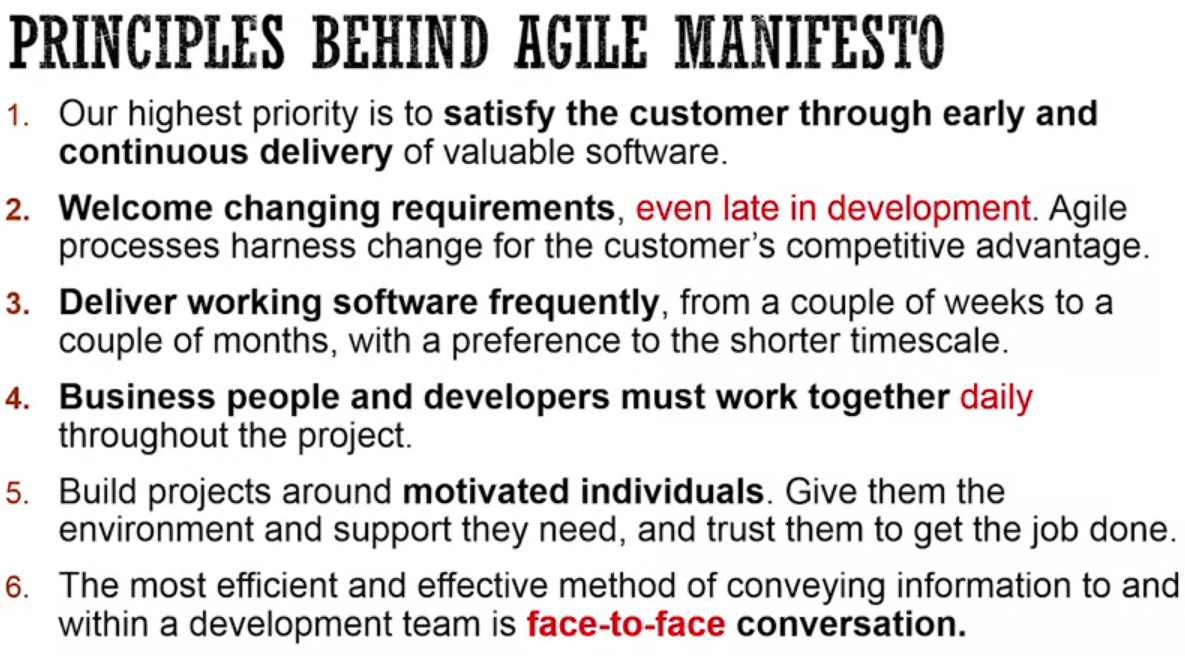
1. "We are uncovering better ways of developing software by doing it and helping others do it." Before we go any further, there are a couple of interesting words so far. The first one is "uncovering". So, what they mean by uncovering is that they are not done yet, they are still learning. And another one that I found interesting is, "doing it". So, all of these practitioners when they define these values, they actually came out of the practice that they were doing. So they were actually developing software this way, and based on their experience, they created these values. And then the third one is, "helping others do it", not telling others to do it.
2. So, "through this work, we have come to value:
   1. individuals and interactions over processes and tools,
   2. working software over comprehensive documentation,
   3. customer collaboration over contract negotiation,
   4. responding to change over following a plan."

Now, as soon as people read this manifesto or these values, they say, well, "does this mean documentation has no value or we shouldn't have a plan?" Absolutely not. Let's continue reading the manifesto to be clear. "That is, while there is value in the items on the right, we value the items on the left more."

So, there is value in documentation, there is value in planning, but we value the working software more than the documentation.

1. First, "individual interactions over processes and tools." What that means is that, a lot of times when there is a problem or when we want to improve our processes, we always look at, you know, is there an additional process that can help us? Is there another tool that can help us? And we do not focus on individuals, and their interactions. A lot of times, we don't invest enough in the individuals and interactions.
2. The second one is, "working software over comprehensive documentation." As I said, it doesn't mean that you don't do documentation. You do, wherever it's critical whatever is needed, you do the documentation, but then, what customers really care about is the working software. For them, documentation doesn't add much value. It does for you but doesn't for the user.
3. And the third one is, "customer collaboration over contract negotiation." So, a contract is important because it defines a boundary in which you need to work, but only with collaboration can you understand the user's needs and build a better software.
4. And then the last one is, "responding to change over following a plan." So again, a plan is needed because that helps you stay on track. But we should not say no to change, and build something that ultimately the user won't need. We shouldn't be blinded by the plan.

Let's look at the principles now. There are 12 Principles.



1. The first one is, "Our highest priority is to satisfy customers through early and continuous delivery of valuable software." Again, the point here is that we want to focus on what the customer values the most, which is the working software.
2. "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage." Again, embrace change is what this means. "Deliver working software freque ntly, from a couple of weeks to a couple of months, with preference to a shorter timescale." So again, the idea is to continuously learn. So, you build something, you show it to the customer and then learn. And then, "business people and developers must work together." This is about the collaboration between the individuals, so that we can build a shared understanding effectively. "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done." Pretty straightforward. The sixth one is, "the most efficient and effective method of conveying information to and within the development team, is face-to-face conversation." Again, talking about the individuals and their interaction. The seventh one is, "working software is a primary measure of progress." What this means is that you can create the planning document, you can create the requirements document, but the primary measure of measuring the progress would be the working software. If I just did requirements and design, I made zero progress in these terms. "Agile processes promote sustainable development. The sponsors, developers, and users, should be able to maintain a constant pace indefinitely." Again, very straightforward. "Continuous attention to technical excellence and good design, enhances agility." What this really means is that if you have technical excellence and you pay attention to your design, you'll probably build a system that will be easily changeable, or you'll be able to embrace change easily. The tenth one is "simplicity-the art of maximizing the amount of work not done, is essential." The eleventh one is, "the best architectures, requirements, and designs emerge from self-organizing teams." Again, trying to invest, or trying to encourage team members to work on the software and stay out of as much as you can. "At regular intervals, the team reflects on how to become more effective, then tunes and adapts it's behavior accordingly." This one talks about continuous improvement. So, the next question is, how does this solve the Waterfall model problems that we saw? So, if you really see all the principles, it kind of divides into two. One is talking about this adaptive nature of software development, where you build some, you learn some, and so on. And then, the other one is collaboration. So, the first one, the adaptive one, kind of helps you detect the translation issues early. Because you are building quickly, so you show it to the user and then if they don't see that aligns with what they were looking for, they will tell you. So you can detect those translation issues early, you can validate user needs early, and you can also detect integration issues early, because you're integrating them quickly. And then the second one is around people and interaction. That again helps you detect translation issues early because you're working closely with your business, so the misunderstandings are clarified out quickly. Now, every coin has two sides. So, again, Agile also brings new problems with it. So what problems does Agile bring? The first one is, since we embrace change and we only look at part of the system to design, sometimes the architecture, the design and the database modeling is very challenging. And since there is always this change, do you feel lack of control? And there is a little bit of unpredictability in what you're building, and that makes leaders of the organization a little bit uncomfortable. And then, another one is that since the customer has to be involved throughout the process, they have to spend a little bit more time on the system.

