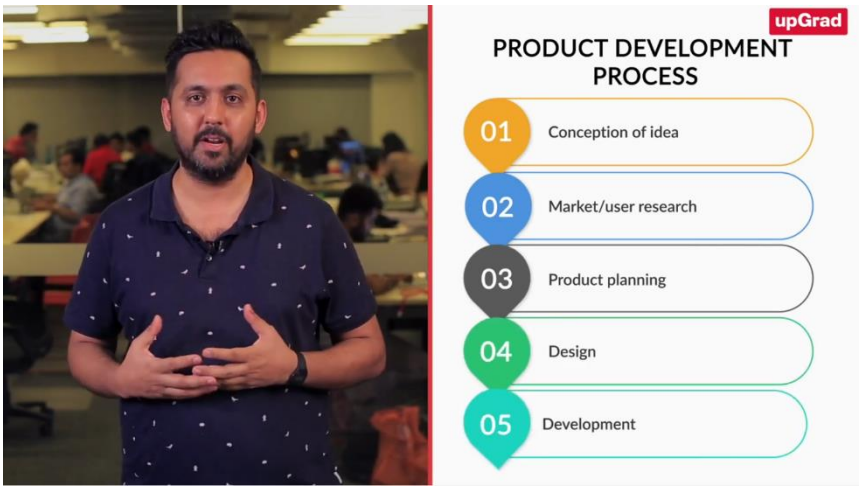
Transcription

Introduction to Agile Methodology

Hello there. I think now you're pretty familiar with the various methods to conduct user research, and how to develop product artifacts. And by now you also know everything you need to about design sprints, prototyping, and MVPs.

The next step now is to build your product. Though, before that let's first get some questions answered. First, what goes into developing a product? How is it done? What is waterfall methodology? Our subject matter expert will help us with all these queries.



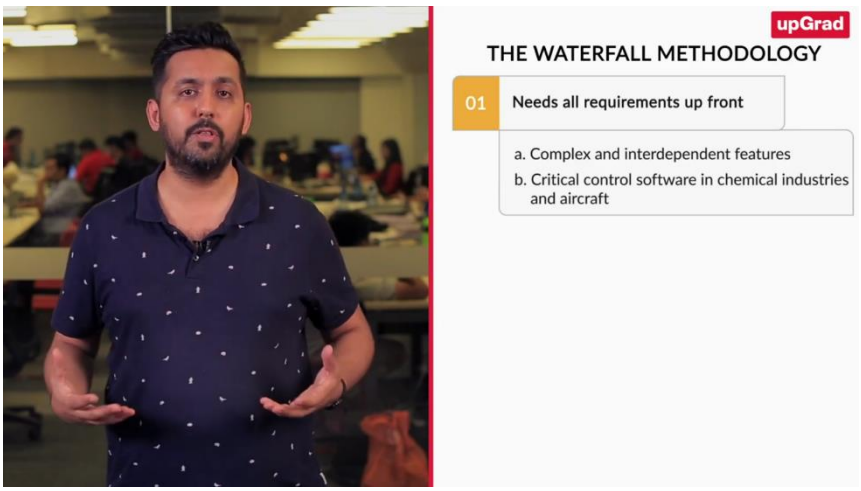
Every product that we develop has a series of steps that they go through, like conception of an idea, conducting market or user analysis, planning features and specifications, design and development of the product then testing it before deploying, and finally maintaining it if required. We call this the product development process.

Sometimes people call this as product development life cycle, but it's nothing but the process that product goes through when it's being developed. Traditionally, product development is done in a sequential manner. Taking cues from the much of all the manufacturing and construction industry.

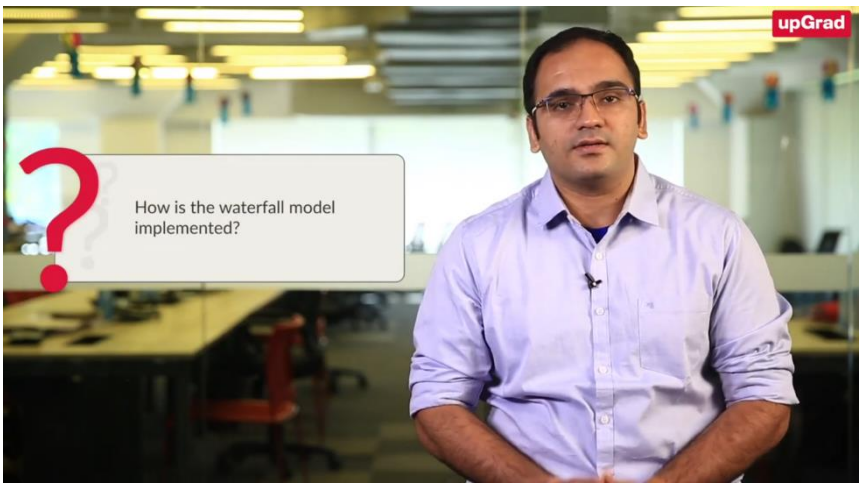
Widely known as the waterfall model, it is a sequential software development process in which progress flows steadily towards the conclusion.

Like a waterfall, through the phases of conception analysis, design development, testing, deployment one after the other.

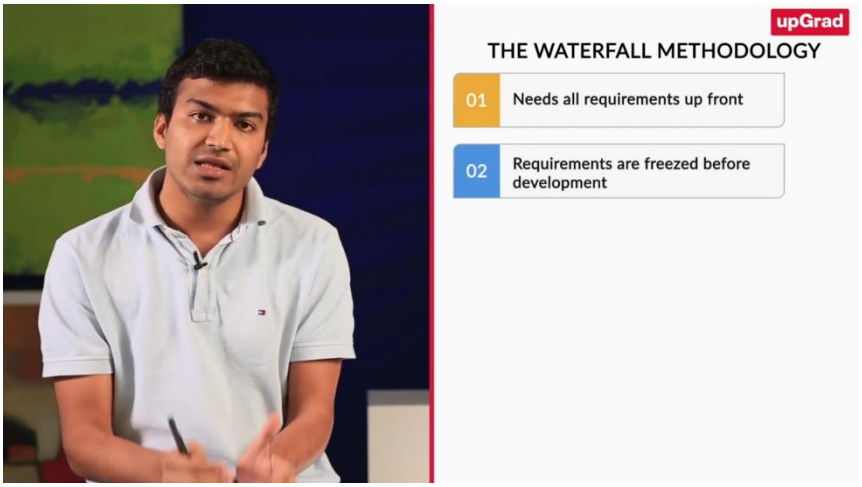
Consider windows or Mac OSX, which are the operating systems for computers. These operating systems have literally thousands of features, which are interlinked with each other. So, for developing or creating a next version, all the requirements should be detailed out at the beginning before developing them. Once the requirements are figured out, the design for all the features is done and then proceeding to further steps.



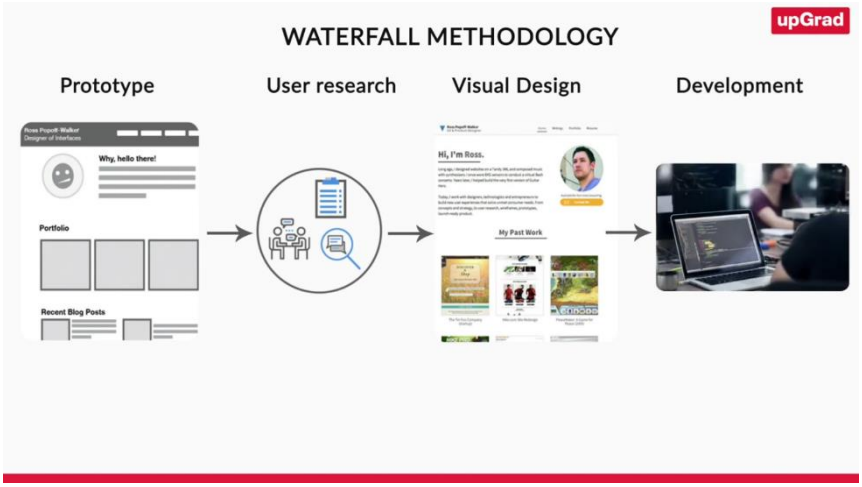
This method is also successfully used in developing mission critical control system software, used in chemical industries and aircrafts where you know all the requirements upfront.



You have seen how traditional methodology fares well for extremely complex and interdependent features. Let's find out more about how it's implemented for the development of any product.

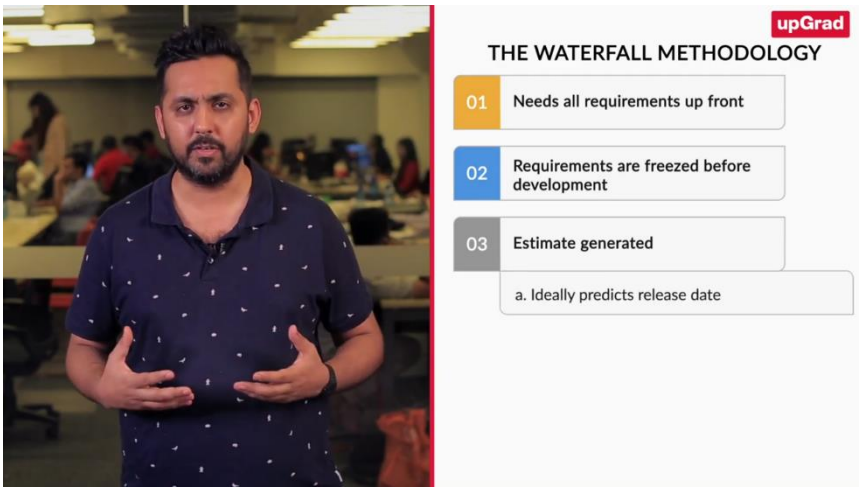


Waterfall is a method in which we have to freeze on a particular stag of a product development. And then we develop that part. I like to give an example of a project, which essentially talks about waterfall. So, we had this project where what we did was, we first delivered the UX.



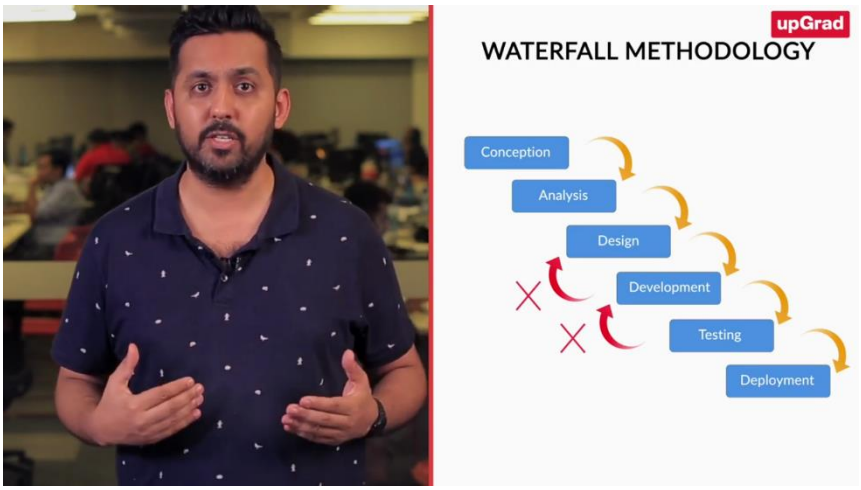
We developed a prototype, we use this to do some user testing, we iterate it a little. And then we did the visual design on it, where we implement all the colours, all the interactions, the look and feel of that particular product, essentially like dressing up the product. Post that, we actually went ahead and developed it.

Now that makes a lot of sense. The waterfall method is quite a sequential process of product development. But does it fair well in all situations or are there any shortcomings?

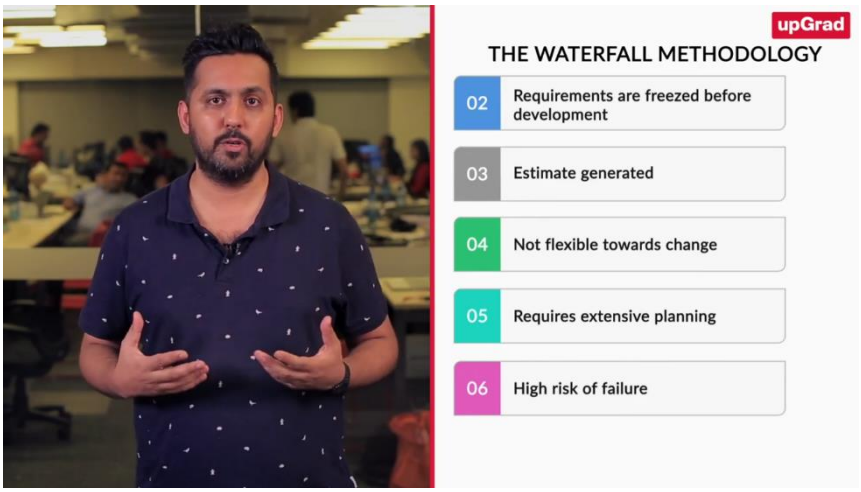


If nothing changes over the course of product development, then estimates generated from the waterfall methodology will result in a predictable release date. However, as we know, change is inevitable and will come in many different forms.

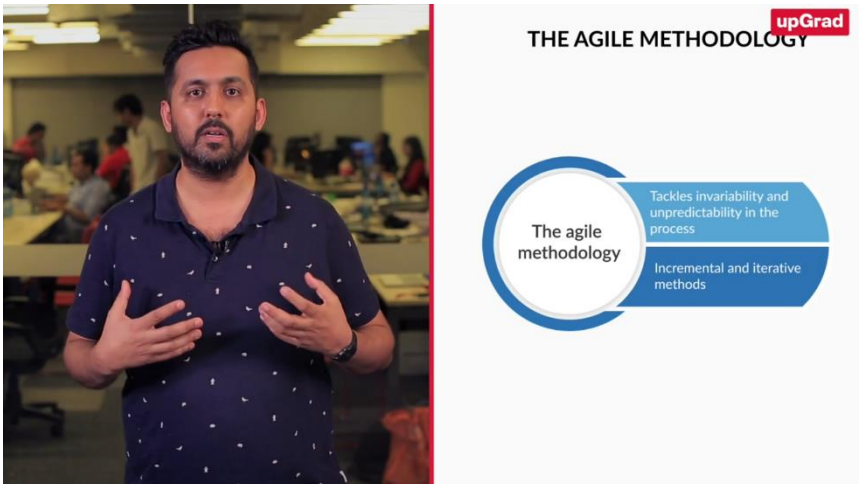
For example, you might learn more information about a feature and would need to adjust requirements or your team might realize once they begin the build, that it is more complicated than initially imagined. These factors would change the delivery dates and add risk to the project.



As this process is sequential, once a step has been completed, developers can't go back to a previous step, not without stratching the whole project and starting from the beginning.

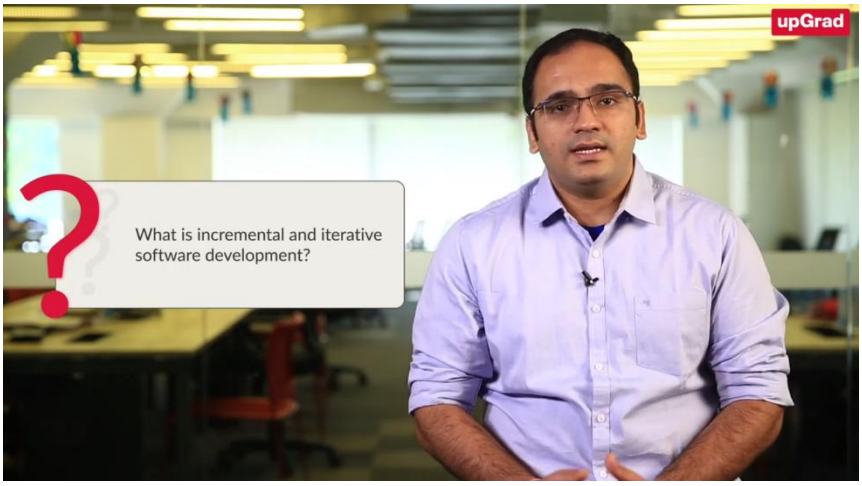


There's no room for change or error. So, the project outcome and an extensive plan must be set in the beginning and then followed carefully.

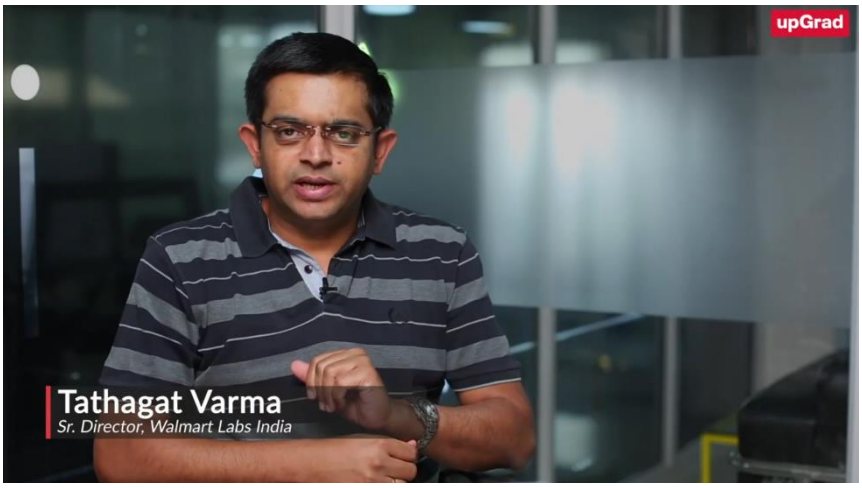


But the risk of failure is inadvertently high. In order to tackle this software practitioners brought agility in product development, which is now widely known as agile methodology. Agile came about as a solution to the disadvantages of the waterfall methodology. Agile approaches help teams respond to unpredictability through incremental iterative work cadences.

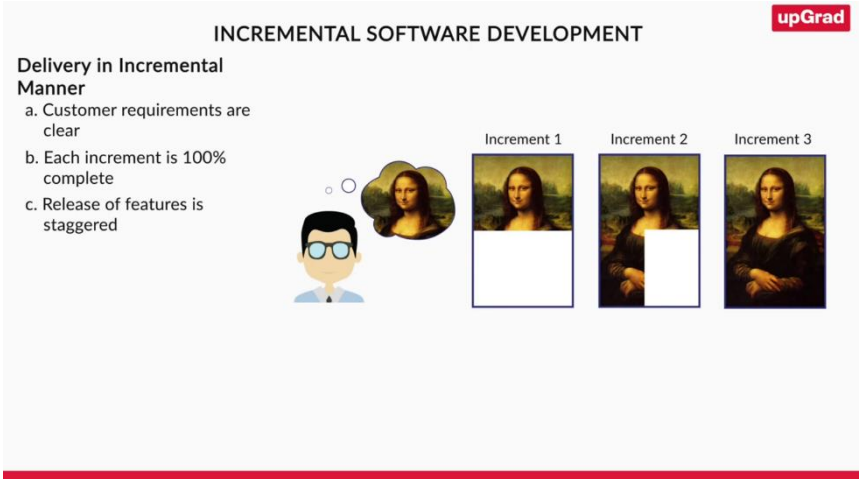
So, in this video, you learned about product development and the various steps involved in developing a product. How it is done traditionally in a linear sequential manner and the shortcomings of the waterfall methodology due to its flexibility to address changes. In the next video, you'll learn more about the agile methodology. See you.



Now you've already seen that the traditional or waterfall methodology failed to sell well for fast paced technology-oriented products. And that's how software practitioners came up with the agile methodology. Let's understand agile in more detail by learning more about it. So, what is agile methodology? What is incremental and iterative software development? Let's ask our subject matter expert.



To understand some of the ideas behind agile, let's start our journey from incremental and iterative software development. Let's say you are an artist and someone commissions you and says, please draw a painting of a Mona Lisa for me.

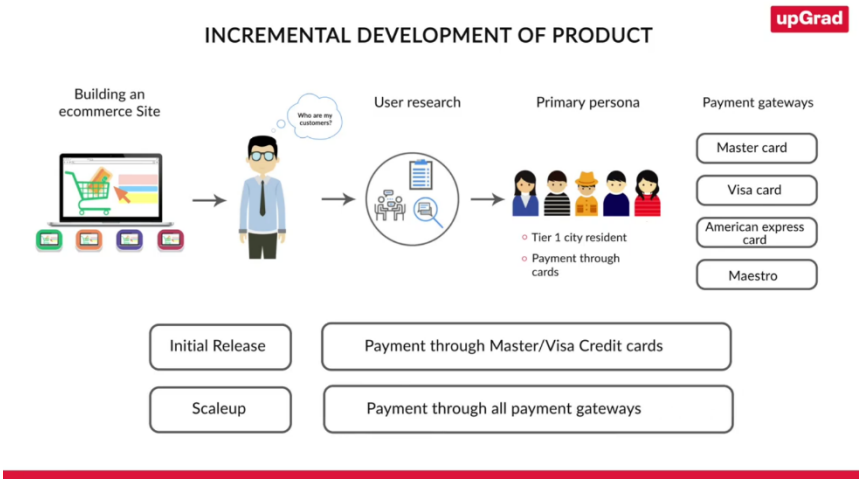


Now, if it is very well known that this is the painting of Mona Lisa, it's clear in your mind, you can probably take it from the net and you can start painting on that. But let us say our customer says, Hey, I don't quite know whether you have the ability to draw it on time. I don't know about your effort estimation. I don't know about your quality, whether you can deliver it or not. Can you deliver me some of the components of that? So, it's possible that you might say, okay, I will deliver you the copy of Mona Lisa in an incremental manner.

So, what does that mean? Maybe what I'll do is I'll probably draw the first top half of that, and it's going to look exactly the same as how it's going to look like towards the end. That means, it does not require any more human touch.

When the customer says, Hey, this looks fine. These are the proportions that look fine there. Why don't you go ahead with the rest of it. I then start doing the second part of that. Now that's a great way for me to stagger the features of Mona Lisa and give still the same thing, but then give it in two or three increments.

Each of the increment is hundred percent complete in all respect, and it allows my customer to have a look at it and make an assessment of my quality, my workmanship. And also, I can see as an artist, whether I know my estimates are right, I know the raw materials that are needed or not in terms of time and so on and so forth. A great way for me to basically stagger the release.

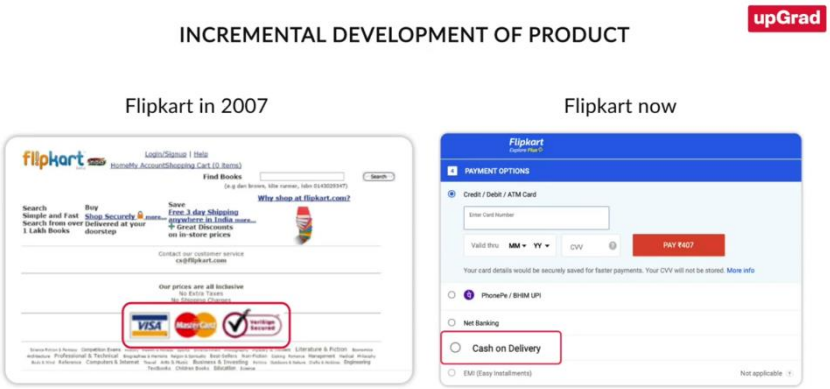


So, let us say, you are building an eCommerce site, right? You want to sell something on the net. Now you might have a challenge in terms of understanding who is the customer that I'm going to serve. Maybe you do an initial study and

you find that my customers are in the big metro cities of India, Mumbai, Delhi, Calcutta, Chennai. And you realize that they have a certain habit.

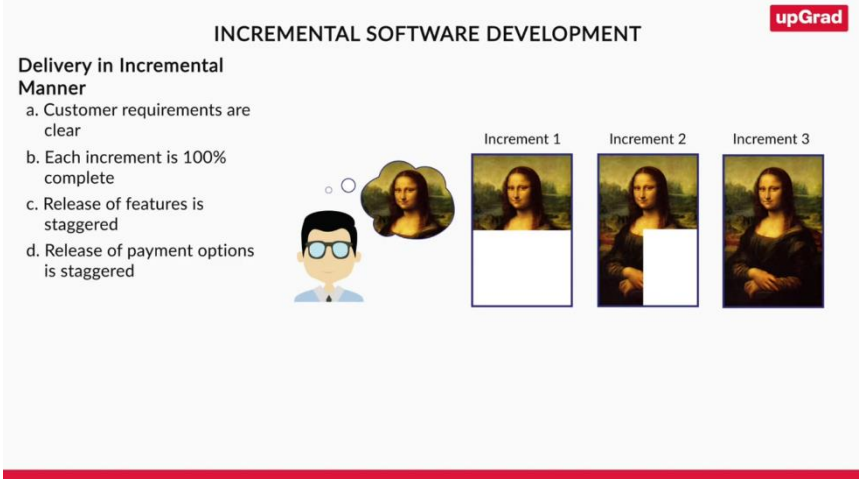
So, they like to, they're comfortable with swiping their card. They are looking for some kind of a thing. You start making a certain kind of an offering to them. In some cases, you might also decide to say, Hey, the payment gateway integration with MasterCard, with Visa, with American express, with all these things is too time consuming, too costly, let me focus on only one.

So, maybe you decide on one of the payment gateways and say, this has the maximum market share. You can start with that. And then as you see the traction from the early customers, you then decide to scale up. What does the scale up mean in this context? The scale up in this context means, okay, now I'm willing to accept all of the payment gateways.



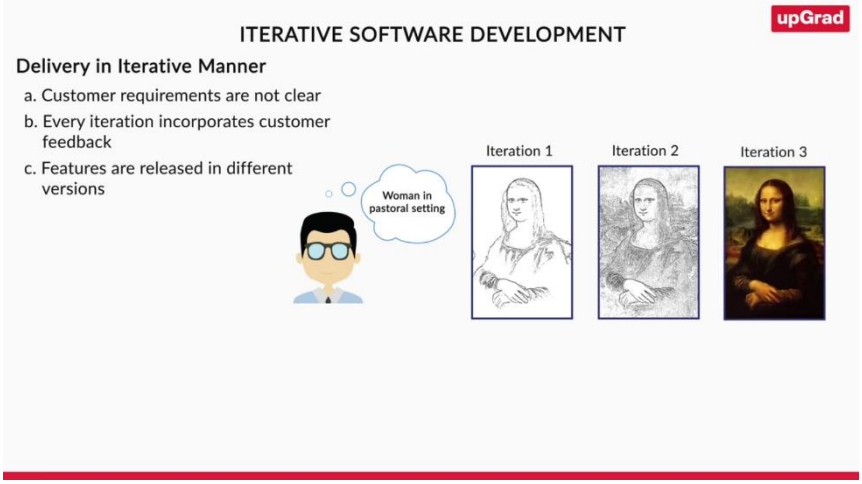
Take the case of Flipkart. Flipkart initially only offered a credit card, but over a period of time, they realized that if they have to go to tier B and tier C towns, they need to have cash on delivery. But could they have done that integration from the day one itself? I don't think that would have been a very pragmatic move because not only would the time taken to build the features be large, the cost would be high as well. And there may not be any immediate payoff on that because not too many people would be using it anyways.

But now that they have decided to scale up and they are going into other cities, that staggering of the feature or incrementally delivering those features makes sense.



So, I think staggering the features is a great way to offer value and grow incrementally as the customer traction grows with you, rather than breaking up a lot of initial cost upfront. And it might not be a very effective way of your time, effort, and money.

So, the incremental way of development or staggered release enables you to offer products or features, which are of value to customers without spending too much initially. Now let's find out what is iterative software development.

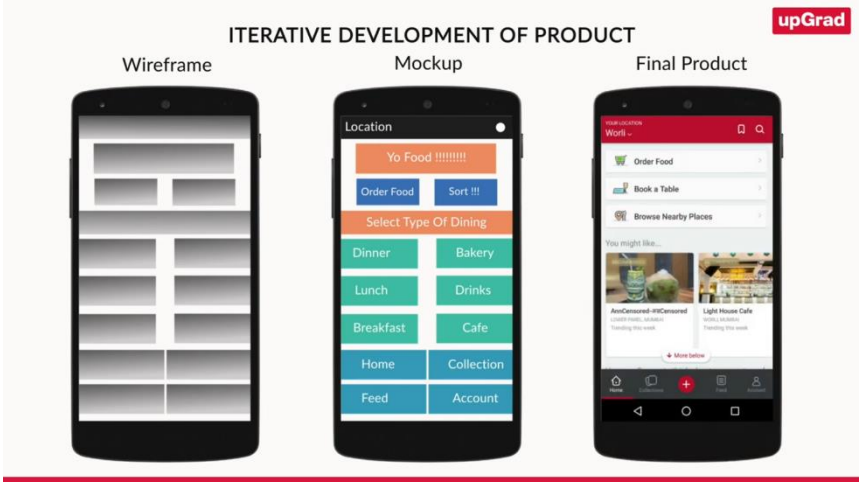


Let us say that my customer comes to me and says, Hey, I'm looking for, can you draw some woman there who is in a pastoral setting? And I don't quite know what he really has in mind. Maybe he's looking at Mona Lisa. I don't know. I obviously cannot make the whole Mona Lisa and waste my time because if the customer says, that's not what I have in mind.

I do a pencil sketch and I show to my customer, Hey, is that what you're looking for? Maybe the customer says, yeah, I like these. Can you make some changes here? Or the customer might outrightly reject it. Even if customer rejects it, I would take it as good news because rather than my final work getting rejected, it's better that I get the feedback now, and then I can act on it. If I have to throw away the work, I might as well throw away the least amount of work.

But if the customer takes the interest and tells me, Hey, these are the things that you can improve. And can you show it back to me? It is actually a very strong feedback to me, for me to improve my product quality. I do that. I go to the next level.

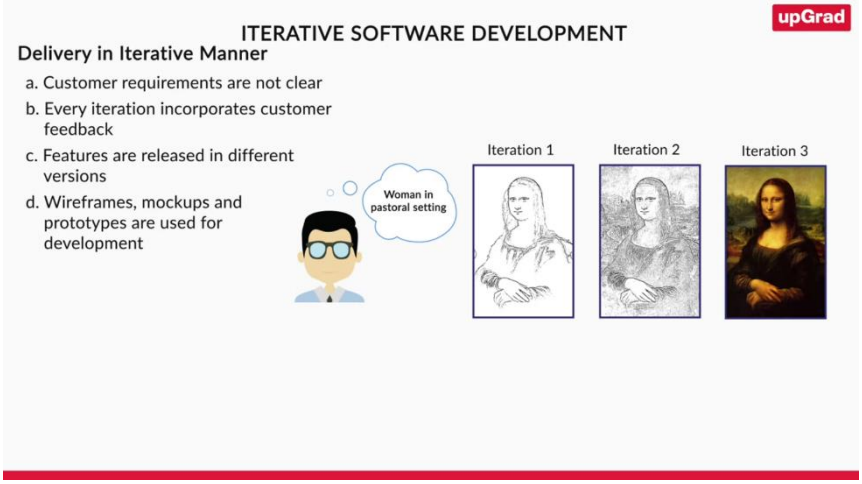
So, think of it in software sense. The customer has something in mind. Maybe I make a wire frame out of it.



The wire frame actually has no content in that, but just tell us that, Hey, this is how the whole thing is structurally going to look like.

The customer says, this looks fine, can you go ahead and show me something. Show me something moving here. I might create a mock up out of that with one or two functionalities that actually you can click on this and it will do this and come back and so on.

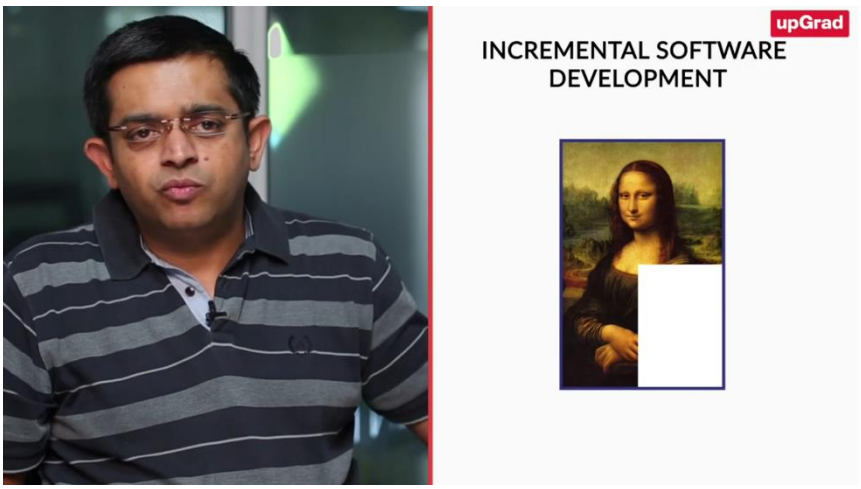
The customer likes it or does not like it. Either ways, I get the feedback. Based on the feedback. I once again go back and do that.



Now in a very iterative, I'm basically going back and refining the content that I'm delivering to the customer. So, this was this is how the iterative software development kind of grew.

So, wireframes, mock-ups and prototypes, which you've studied previously are nothing but iterations of the product. You take user feedback in each iteration and incorporate the same in the next iteration until you get what the user has in mind. In the next video, let's find out how agile methodology deals with changes.

In the last video, we have seen how incremental and iterative development is done. Let's find out more by learning how does agile methodology come into picture?

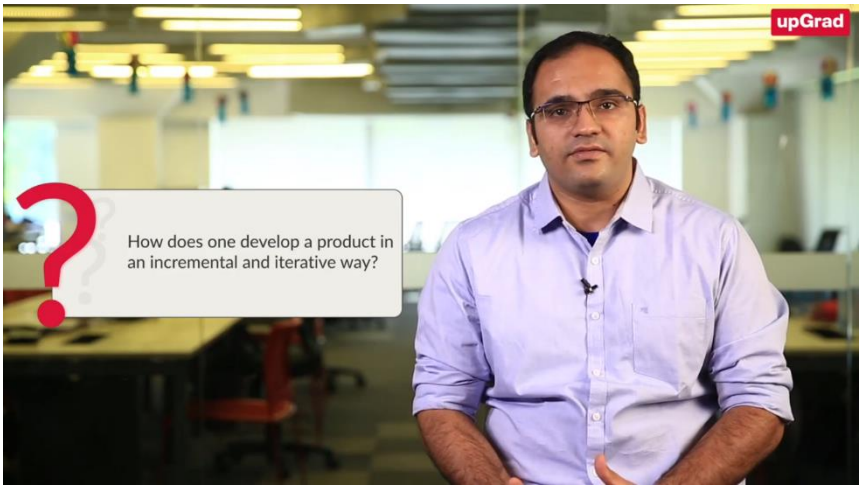


In the nineties, we saw there was an incremental school of thought. If you were very clear about what is to be delivered, all you could do was stagger the releases and mitigate some of the risks around it. That was one good way.

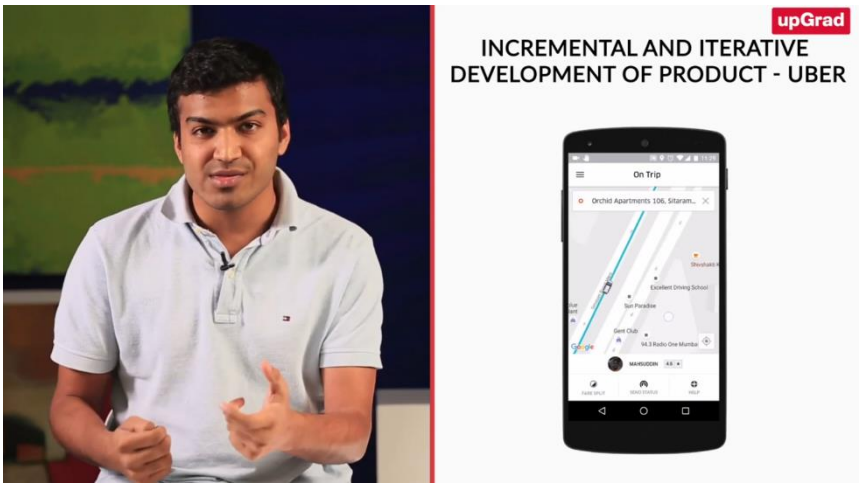
And then you had the iterative software development, especially well-suited for the problems where you did not quite know what exactly the customer had in mind. But you still had to get started. You could potentially take some of the early prototypes and build on the top of it and go in a spiral manner and do some of the software development.

Obviously, both had their own strengths and people started saying, Hey, look, we will find the ideas from both of them. Interesting. Can we merge these two together?

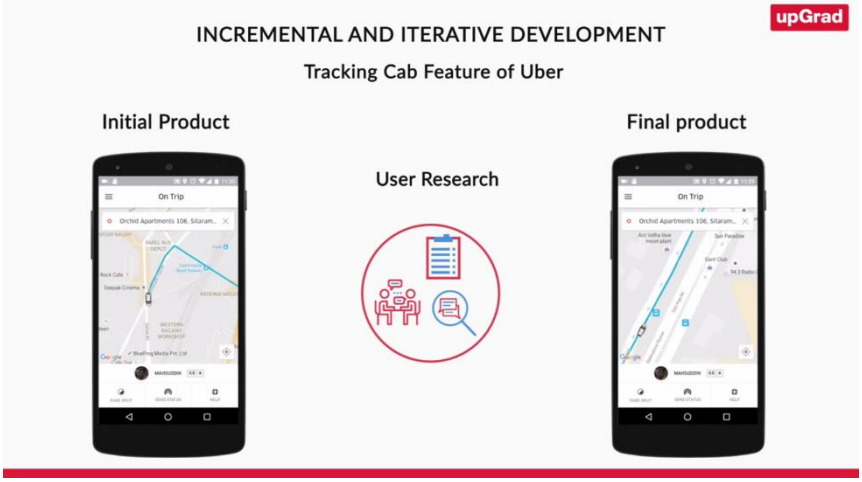
So, people started merging them and that's how some of the agile methodologies were born because people started putting the incremental and iterative in the same way of doing it.



So, agile is both incremental and iterative. But how does it work exactly developing products in an incremental iterative way? Let's see how agile is implemented at Uber in an incremental and iterative way.



Uber has this amazing feature of where you see a car moving on a screen. This engages user in a phenomenal way. It helps you show where your cab is. And essentially, uses a brilliant form of technology to enable a user, to know that where your cab is and how much time the cab is going to take, and in which direction the driver is going to go. So, Uber came to this feature after a lot of iterations and a lot of user feedback.



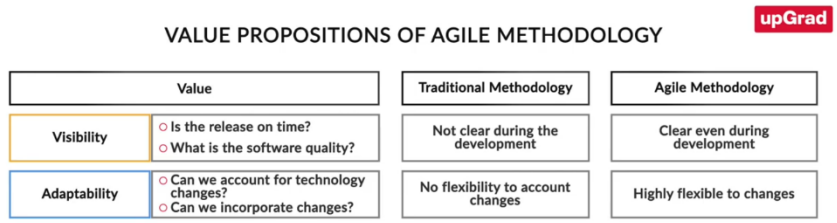
How they first initiated was that they were actually showing a car at a particular point. They were not showing how the car is moving. They showed it to users and they realized, Hey, users are not able to comprehend this particular feature.

Post that, what they did was they thought, Hey, why don't we do something like where we show a car moving on that particular screen. That was actually a very iterative and very incremental change towards that particular feature. So, this is how agile is.

In this video, you learned how the incremental way of software development works, releasing features in a staggered way. You also saw how the iterative way of development is done taking customer feedback into account. And you have also seen agile is an incremental iterative way of product development. In the next video, we'll take a look at how effective agile methodology is.

Life indeed has become super chaotic. Add to that, the technology platforms that change so rapidly. Keeping pace or plan is hard. In fact, if there is one thing that I can predict with certainty, it's unpredictability and change.

Coming to the point, these incremental and iterative ways look useful in fighting against unpredictability, chaos, and neck breaking competition. Let's learn more about how effective agile is compared to traditional methods. What are its value propositions?



Agile is all about creating value throughout the cycle. In a traditional project, what tends to happen is, we have a lot of visibility when the project kick-off happens. And then after that, we are busy in a lot of internal activities, but we are not able to deliver something of value to the customers.

And towards the end of it, once again, when we hopefully deliver the software, there is a lot of visibility. But there is in between time when there is no visibility except the status reports. And let's accept it. The status reports don't always give the ability to predict, is the software going to be on time? What will be the quality of the software?

We probably only have the design documents, but there is no such thing as really actually touching and feeling the product as opposed to reading the documents about it. So, the visibility of an agile project is constant throughout because you are always seeing a working software at any point in time.

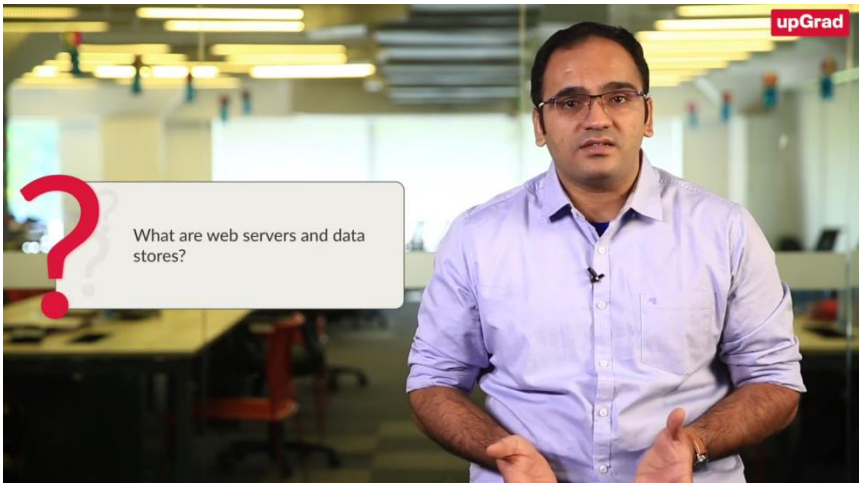
Let's talk about adaptability. Agile methods are not about doing what we call is the big design upfront. In a traditional project, we do the big design upfront. That means we lockdown our options and say, okay, this is how my architecture is going to be, these are my data stores. These are my web servers. These are my offline storage and so on and so forth there.

And then we start getting into a long gestation period of developing the product. What if the technology changes in six months down the line? Are we in a position to incorporate those changes?

Most likely, no, because in a traditional project, it's very difficult for us until we have completed the entire coding, until we have integrated all the modules to actually start changing it in a more effective manner. We need to understand the impact of that. And that makes it difficult to handle.

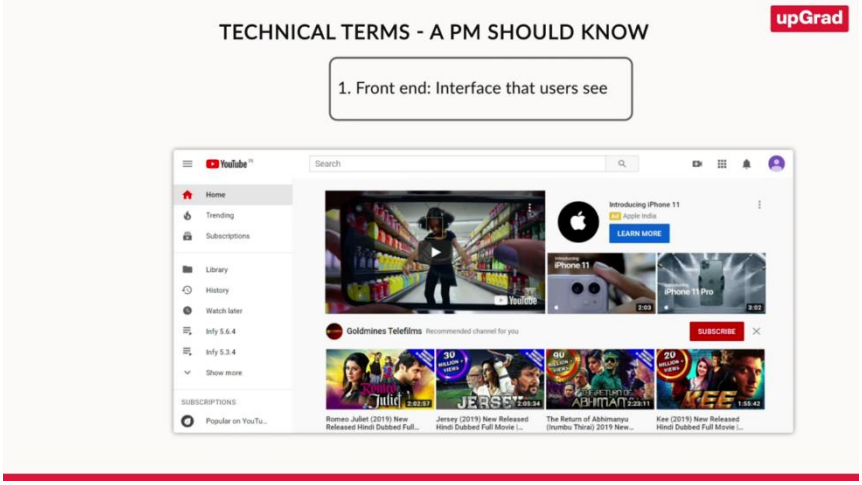
Compared to that in an agile project, we only do so much that we are able to basically demonstrate the functionality of it. And we kind of grow it incrementally as we saw from the examples of incremental iterative development.

So, we are never, in all likelihood, we are never more than two or four weeks away from the last stable version of it, which has been tested and feedback has already been obtained from the customer.

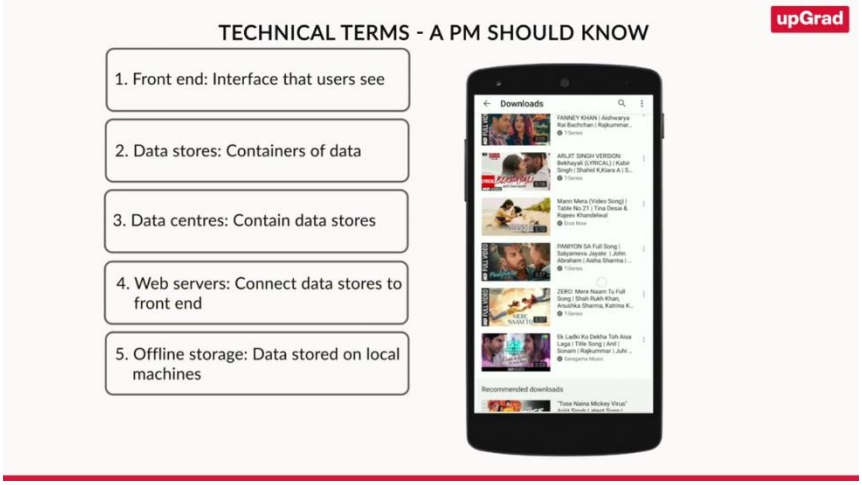


Let me stop you there. I felt bombarded with a lot of technical terms. What is web architecture? What do you mean by web servers and data servers? How are they related to product development?

Well, let's take an example to understand these terms properly. Say you are working on a project similar to YouTube, and let's look at the YouTube website and understand these terms. Now, as soon as you open youtube.com on a browser, we see an interface.



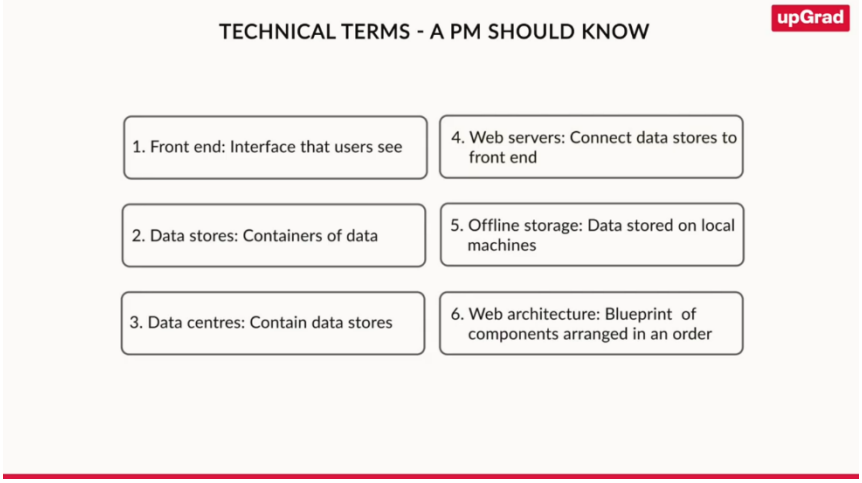
We see an interface where we can select videos, like, share, do all of these things. This interface we see is known as the front end, but that can't be all. I mean, where does this YouTube getting all these videos from? They must have a data stored somewhere. Yes, they do, in something known as data store.



Think of this data store as a large container of data. These large data stores are very huge. And generally, thousands of kilometres away from you in places called as data centres. But what connects the data stores to the front end? You guessed it; it's called the web server.

Apart from these data stores, sometimes some data is stored on your local machine. This is called offline storage. This is available instantly to you. This data can be accessed only by you. Now, if you remember the saved video section on YouTube, now those videos are stored on your local storage or offline storage.

Also, recent messages on WhatsApp mobile app are also stored on offline storage. So, you don't need to wait for a web server to respond. We'll cover this in greater detail in the upcoming modules. Now we have briefly looked at what web server, data stores, etc., are.



Architecture is nothing but a blueprint of how these individual components are arranged in order as well as specific choices concerning these components. These choices can range from what type of web server you want to use, how big it should be, etc.

That was a short and crisp take on web architecture. We will delve into details in the coming modules. Now let's get back to agile methodology. So far, you've seen that agile provides more visibility and enables easy adaptability to changes that surface once the project starts. More on that from our subject matter expert.

Let's talk about the business value. In a traditional project, when do we deliver the business value? Not until the very end, because all we are delivering are documents and status reports. Whereas in an agile project, right from the first iteration, we are delivering something of value to the customers.

VALUE PROPOSITIONS OF AGILE METHODOLOGY

upGrad

Value		Traditional Methodology	Agile Methodology
Visibility	○ Is the release on time? ○ What is the software quality?	Not clear during the development	Clear even during development
Adaptability	○ Can we account for technology changes? ○ Can we incorporate changes?	No flexibility to account changes	Highly flexible to changes
Business Value	○ When do we deliver value to customers?	Only at the end	In every increment and iteration
Technical Risks	○ Does tech stack work together? ○ Is it the right functionality? ○ Can the expected performance be achieved?	Cannot be mitigated - long development cycles	Can be mitigated - short development cycles
Social Risks	○ Can the team collaborate? ○ Can the team execute well and deliver on time?	Cannot gauge team's performance till the end	Can gauge the team's performance in short time
Market Risks	○ Can we incorporate market changes? ○ Can we incorporate requirement changes?	Difficult to incorporate changes	Easy to incorporate changes

These could be very small features that the customers could test them, play with, give the feedback on. And if they like them, they can even potentially push them into production. So, this is a tremendous value that our customers have never seen before.

And talking about the risk. What happens in a traditional project is we have 8 or 10 or 15 developers who are working together as a team, but they are hardly ever integrating their software together. And after three months or six months or nine months, they start putting it together. And because they have never integrated the software before, it runs into invariably a lot of problems.

Compared to this, in an agile world, the team decides to deliver a working software in the first two or four weeks of its existence. Now imagine the power of doing it. It is mitigating the technical risks around it. What are the technical risks?

Is this technical stack going to work together? Are we going to get the right functionality out of it? Can we deliver the right performance out of it? This can be mitigated at the higher order.

The second part of that is all about the social risks. What are the social risks? Can this team come together? Does the team have the right chemistry for it to collaborate with each other? Can they actually do the estimation of the task properly? Can they actually work together and deliver on time?

These are important parts of it. We only get to discover them typically six months or nine months down the line. But in this case in the first two weeks or four weeks, the teams have to go through the entire cycle.

And finally, if we talk about from the customer in the market point of view. What if the market changes? What if the customer does not like what we deliver? These could be very substantial risks in a long run project. But when it is a matter of few weeks or a couple of weeks, we can very easily get a feedback from the customers, whether that particular UI design, for example, or whether that particular interaction style, does it meet the customer requirements or not. And the customers can also give the feedback much sooner than later there.

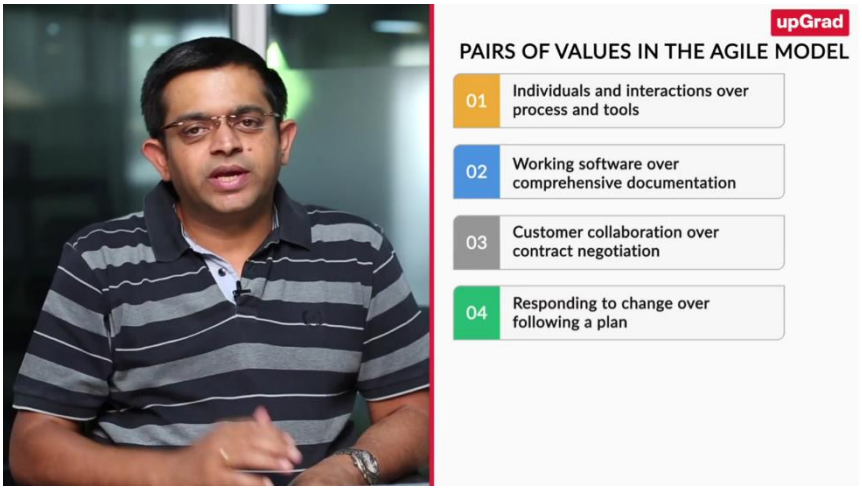
In this video you saw how agile is more effective compared to traditional methodology in terms of increased visibility. As we get to see working software, rather than status updates.

Increase adaptability, as there is no big design upfront. Increased business value and mitigated risk concerning changes in technology and the market. Also, the social feasibility of the team working together is tested within a short span in agile.

You also got a sneak peek into various technical terms like the front end, data stores, offline storage and how they together make up web architecture. In the next video, let's figure out which methodology fits best in a given scenario.

Let's talk about the agile manifesto now. Agile manifesto was created by 17 of the leading software practitioners back in 2001. And they came together and wrote just 10 lines that have had the biggest impact on the software industry.

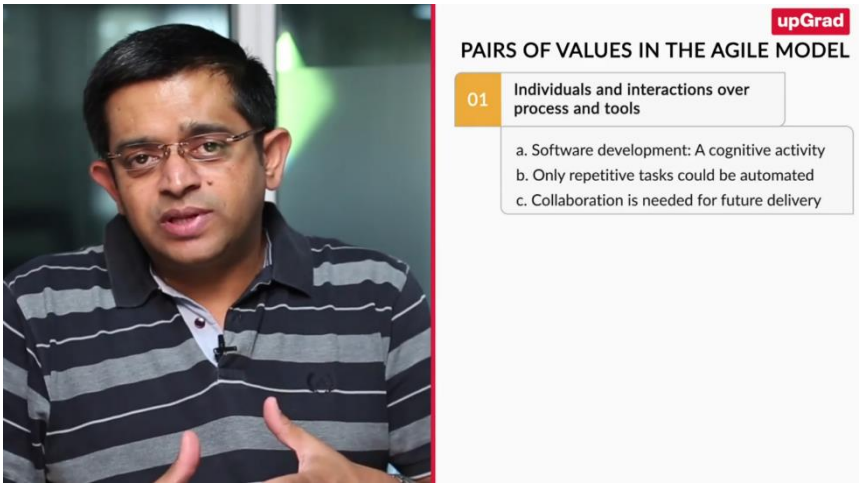
They said, we are uncovering better ways of developing software by doing it and helping others do it. Now these were practitioners and they did not believe in simply just being an armchair, a methodologist. They were actually practitioners. So, they were talking with a lot of experience.



They further said, through this work, we have come to value and they gave four pair of values.

1. The first one, individuals and interactions over process and tools.
2. Second one, working software over comprehensive documentation.
3. The third, customer collaboration over contract negotiation.
4. And the fourth one, responding to change over following a plan.

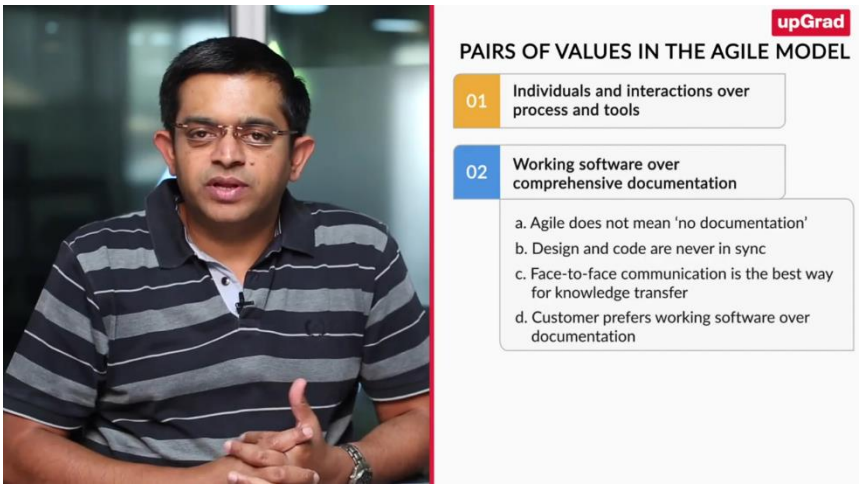
And what they said was, while we value items on the right, we value items on the left more. Now let's deconstruct each one of them, just so that we understand them a little better.



Individuals and interactions over processes and tools. What these people thought that software industry is a fundamentally cognitive activity. It requires people to think and solve complex problems. Some amount of automation could be applied to the repetitive tasks, but a lot of times the work we are doing is very novel in nature. And you cannot simply automate the whole thing.

Secondly, can one individual deliver a software? Let's say I am a great Java programmer. Can I deliver the entire software by myself? The short answer is no. We say it takes a village to raise a child. You need everybody. You need people with front end development skills. You need people with backend database, middleware, user experience, production operations, product manager. You need a bunch of people in the room together collaborating in order for a feature to be released to the customers.

And what do these people do when they come together? They need to interact with each other. They need to collaborate with each other. And no amount of process or tool can replace the efficacy of human collaboration. And that is why these people said we value individuals and interactions over process and tools.



Let's look at the second value, working software over comprehensive documentation. It's also time for us to correct a myth. A lot of people believe that agile means no documentation. Now well, that might make some of you happy that I don't need to write boring documents.

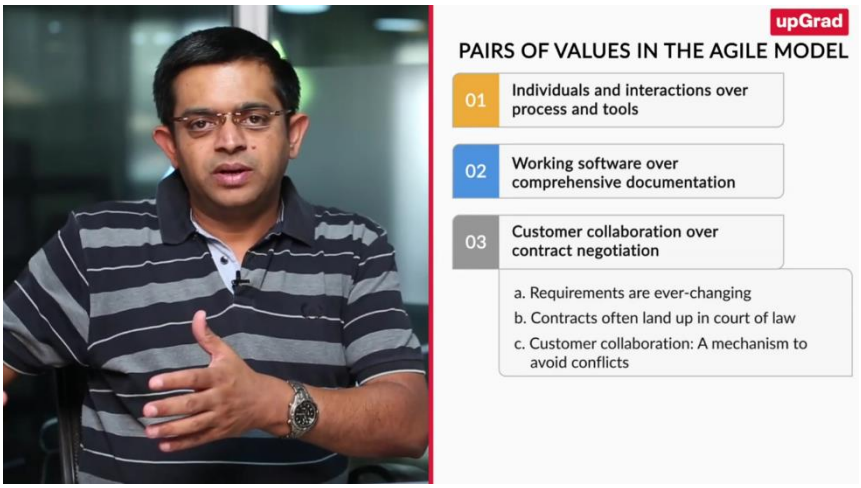
The truth is agile never said no documentation. If anything, it said, we don't really favour comprehensive documentation. People also talk about and say, Hey, I need the documentation because how do I understand the design of that? And I have a question to all the people who say that, how many times in your career have you seen the design and the code being in sync with each other?

The third argument people give us, oh, when we have new people in the team, we need the information to be exchanged and we need the knowledge transfer and so on. And I ask them the same question. We are talking about a tacit knowledge inside a software team.

A lot of times, if you go to a very expert designer or very innovative test engineer and you ask them, Hey, can you tell me how exactly you work? I can bet nine out of 10 times; they would not be able to describe you their process. But if you ask them here, I need to learn from you. How do I do that? They would say, why don't you pull a chair, sit next to me and watch me how I do that. And over a couple of hours, you can probably learn how I'm doing their thing.

So, what we believe in software industry and more say in the agile community is, the best way to exchange and communicate knowledge, especially the tacit knowledge is to have a face to face communication.

Finally, if you are a customer, if I give you a choice and the options were one, would you like to get a 50 page document to understand the design, and option number two, would you like to see a software, even though the software is not complete, but it will tell you how it's going to work, which will you prefer. I can bet 99% of us will probably prefer a working software. So, I think it makes sense for us to go with the working software.

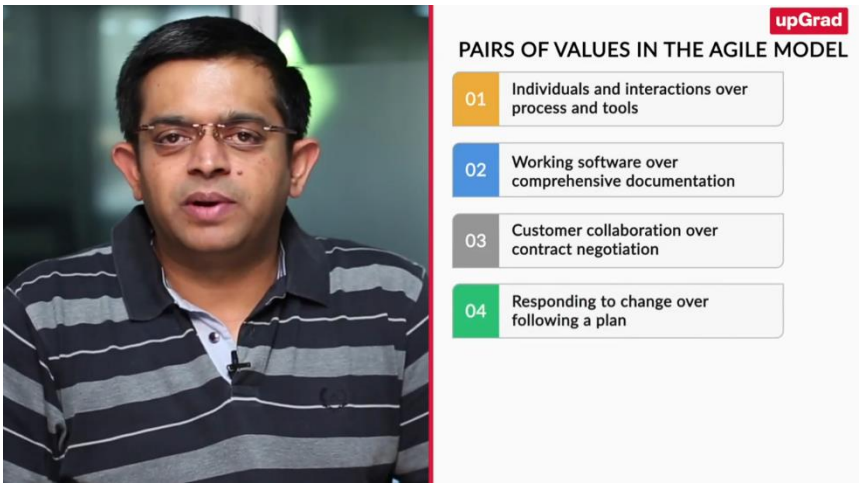


Number three, customer collaboration, over contract negotiation. There is a saying in software industry that says, walking on water and developing software is easy, if only they were frozen. And it's not an exaggeration, we never see the requirements frozen or complete. They keep changing even well up until after delivery and so on.

So, it's not easy. It's a surprise that if the software itself is not known and let us say, we have to deliver something after 12 months, we don't know what the eventual requirements are going to be. We don't know how they will pan out. How can we even write a contract around it, which is something which we don't even know what it's going to be like.

So, it's not a surprise that when you write these contracts, they found that a lot of these contracts were landing into the court of law, where people were fighting all the time as to, Hey, no, this is what was the interpretation and so on.

What the agile saw was an opportunity to actually create a different perspective where they say customer collaboration is the most important thing. Because end of the day, why don't we create a mechanism for people, we as developers and customers to collaborate with each other, rather than putting it on a piece of paper and saying, why don't you understand what it is written? So, they place more value on the customer collaboration.



And finally, responding to change over following a plan. Let me give you an example, just so that we understand this with little context.

In 99, two people came together and created a company at Stanford and they named it Confinity. And they said, what problem are we going to solve? Those days, nobody had a smartphone, but people used to have PDA's or the personal digital assistants. Some of you might remember having the Palm PDA's for example.

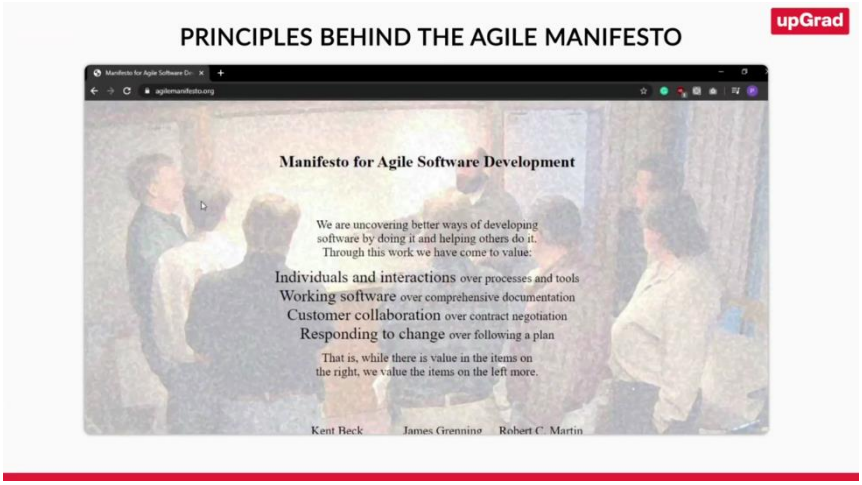
And they said, Hey, everyone is having Palm PDA. Why don't we solve a problem? Suppose I have to transfer $10 to you. I can take my Palm pilot PDA, and I can beam $10 to your Palm pilot PDA. Looks very Star Wars kind of a thing there. And it looks very cool for a techie to have.

And they saw that, hey, they had some few 30, 50,000 customers and they said, what do we do now? How can we take it to the next level? And somebody in their team said, why don't we do an experiment? I mean, we are trying to solve the problem of transferring money using a Palm pilot. Can we reframe the problem?

For example, why do we even need a Palm pilot for that? Can we transfer the money without palm pilot? Now, can you guess the name of the company that came out of that experiment? The name of the company is PayPal. PayPal happened not because somebody had a grand vision that the world would need one day a mechanism for us to transfer the money electronically.

PayPal happened as a plan B. Confinity was the plan A and they did not really find a lot of success in that. And instead of going down that path to somehow execute that plan, irrespective of whether we get the results or not, they decided to listen to the feedback, respond to those changes and change their direction. And the result was PayPal, right?

So, that's exactly what agile mean when they say responding to the change over following a plan. Sometimes we follow the plan mindlessly and we bring it to the completion. I call it as operation successful, but the patient died. There is no fun and really creating those kinds of plans where agility is lost at the cost of just completing the project, just because the original contract says so.

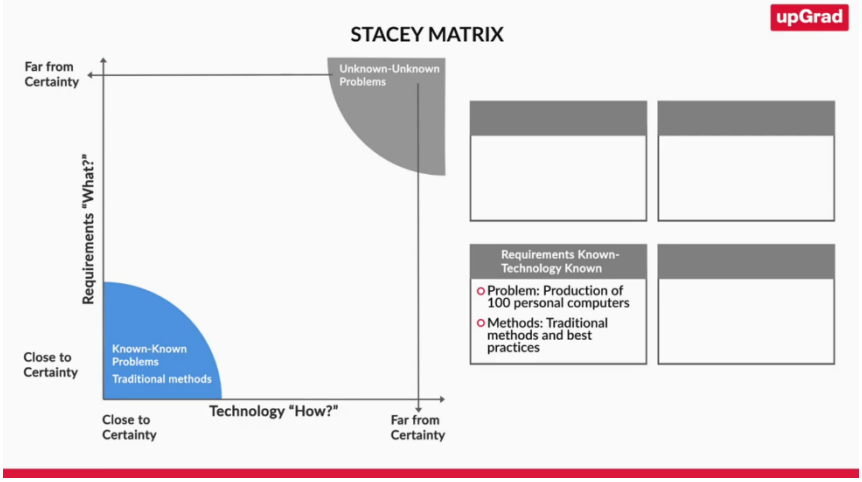


So, these were the four values. And along with that, they also wrote 12 principles. I would encourage you to go to agilemanifesto.org to explore the 12 principles, because it will take a lot of time for us to do in this session.

These 12 principles would actually give you a very strong foundational mechanisms, how this can apply and how you can make it happen inside your own teams there. So, I would leave you at this point on agile manifesto, go and explore the 12 principles there.

You've seen how agile is more effective compared to the traditional methodology to deal with unpredictability. But does that mean you can apply agile in all situations? When can you use agile or waterfall? Is it in any way related to the type of problem we want to solve? Let's find out from our subject matter expert.

As a product manager, when you create products, you're essentially either solving a problem your user is facing or bridging the gaps on the basis of user needs. So, in order to understand how to solve these problems, let's first talk about the different kinds of problems.



Let's try to map the problems in a very simple matrix by taking the Y axis, the how well do you know the requirements, ranging from closest certainty to far from certainty. And on the X axis, let's take how sure are we about technology to solve the problem?

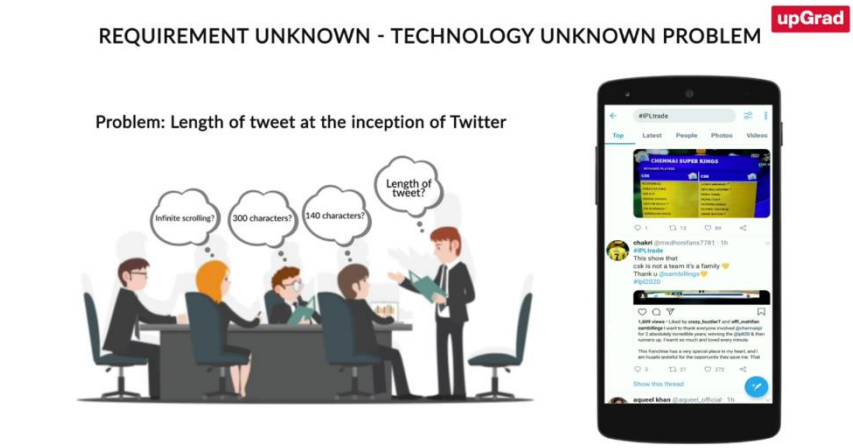
Again, ranging from close to certainty and far from certainty. Technology here can be the software, hardware, domain expertise or anything that helps us solve the problem. That is basically the solution to address the requirements. This is famously known as the Stacey matrix, which helps you understand the complexity of the problem and categorize them into some very simple buckets.

So, when I plot the problems on the Stacey matrix, on the lower quadrant, which are known problems, I know what is needed. I know how I'm going to do that.

If I take the examples of personal computer as my product, if I were to produce a hundred PCs with known specifications that are already available, it's very easy for me because I know exactly what I want. And I know what technology, what kind of hardware, what kind of design I need.

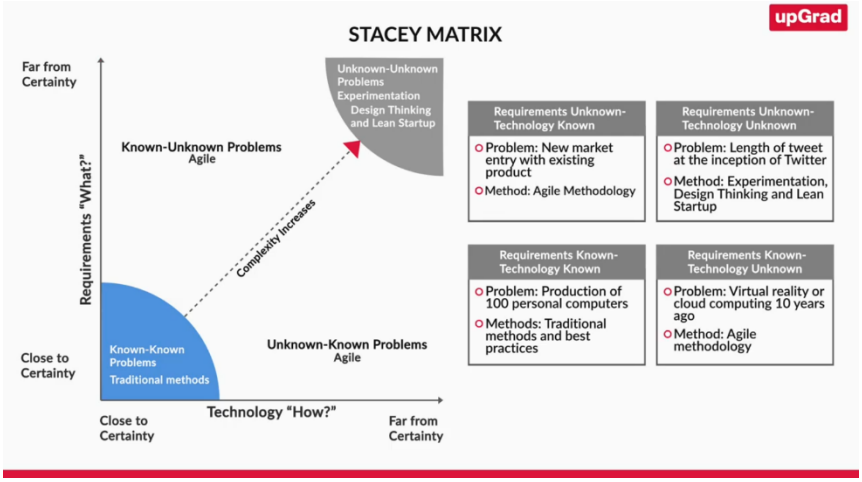
All I need is to provide the new raw material. And the process is already been predefined. And I just need to follow the process. I can apply traditional development methods and best-known practices, and basically follow the process in a very predictable and repeatable manner.

On the other end, let's talk about the extreme top corner, which is the unknown quadrant. These are the problems that you don't know enough about, and these are the problems we don't know enough about their solution as well.



So, it's an unknown kind of a thing. Let's say we're all working for Twitter and Twitter is yet to be born. At its conception, we're not sure what we are going to solve and how are we going to solve it? And we are all wondering what should be the length of a tweet.

One group says that it should be 140 characters because 140 is a standard character limit for, for the SMS. While somebody says that 140 is too less, so let's make it 300 characters. Someone might even suggest not to limit the characters and have infinite scrolling.



Here, we don't know the cause and effect relationship between the problem and the solution. We are not sure if something like 300 characters will lead to a higher engagement of the users or a lower engagement. Will infinite scrolling actually result in a more people spending time or less people spending time?

So, how do we decide that? We do a series of experiments and it may even be possible that when I do these experiments, I might actually come back and say, why are we even solving this problem? Maybe there is a better problem waiting to be solved.

So, this is a class of problem which we believe even agile methods, the way they are understood and practice today are not the best fit. There is entirely a new class of solutions like design thinking, lean start-up. And these are very powerful ideas that have emerged in the last few years, which are helping us solve problems in the unknown, unknown quadrant.

Now in between you can see that you could have a variant where you know what is to be done, but you don't know how it is going to be done, or you have the technology, but you don't know what to solve. The complexity is a little less compared to unknown, unknown problems, but it is higher than that of known-known problems.

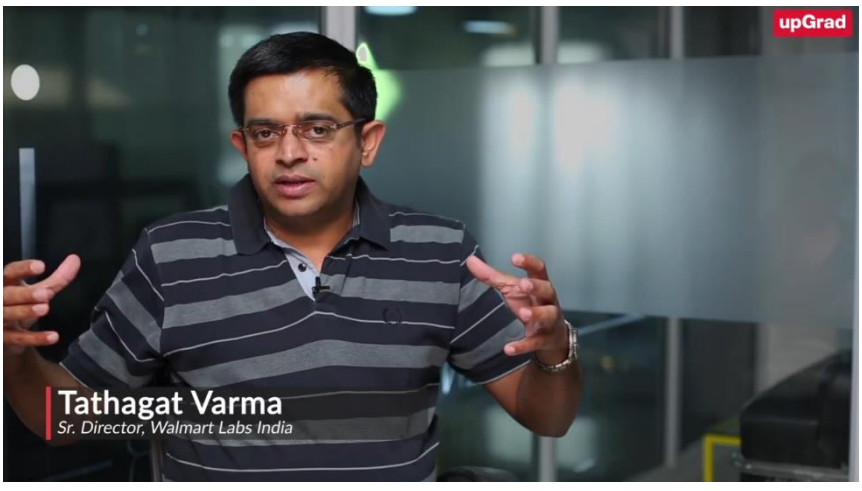
Let's say if you have only one of the variables known to you, that is, you know the technology, you know you can make a web server out of something, but you don't know exactly what the customer has in mind.

Is he looking for a solution that requires him to have a payment gateway integration, or is he looking for something that allows them to do only lead generation? This would be the case of new market entry with existing products.

On the other hand, if you know the requirement, you know we want virtually virtual reality or cloud computing 10 years ago, but then technology was not very mature or developed. This falls in requirement known, and technology is not yet very certain.

What you will find is that the agile class of solutions are a great fit for the problems for the case in the middle. You can do a series of experiments. You can figure it out using an early feedback mechanism and add value to your product iteratively and incrementally.

So, agile seems to be the best suited for most of the cases. Let's find out if that is the actual case with the industry.



I would urge you to think of it whenever you are talking about in the context of somebody saying, Hey, we follow the agile process that is the best in its class. Or we follow a standard process. Because my definition is there is no such thing as a standard process. Every team has to figure out its own process. Every team has to find out what is it that allows them to go the fastest.

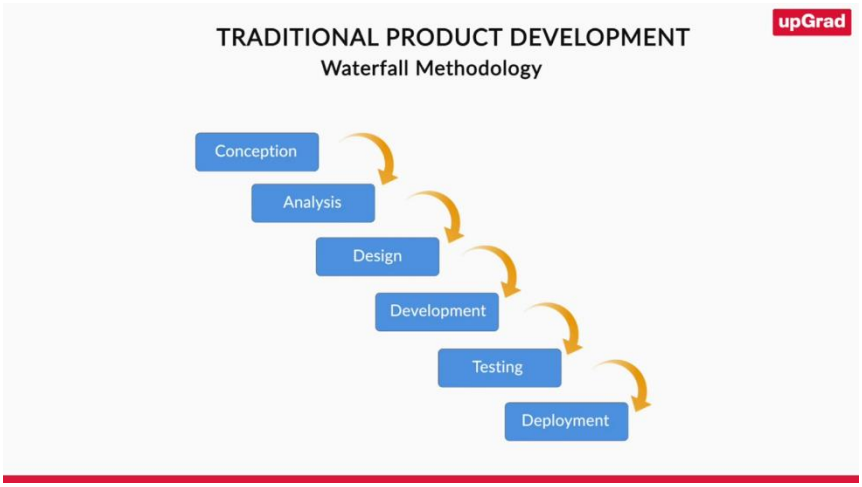
If there is a team of novices, obviously they will need much more checks and balances, much more help. If there's a team of experts, that team might decide to solve the same problem, much more efficiently, much more effectively. So, every team must figure out what is the right way of doing that?

You now know how to evaluate the complexity of a problem using the Stacey matrix. It helps you analyse the problem and choose the method that best suits the scenario. Known-known problems can be solved by traditional methods and unknown-unknown problems by new techniques like lean methods or design thinking.

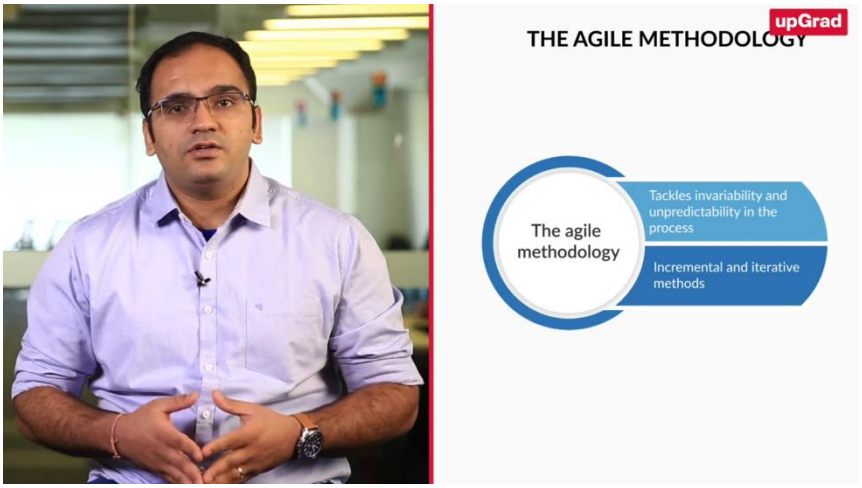
Anything that falls in between can be approached in an agile way by doing things incrementally and iteratively. Next up, a quick summary of all that you've learned in this session.

This was an interesting session. Let's summarize what you've learned. You first started off with learning about the product development process, which is nothing but the process through which product goes through when it is being developed.

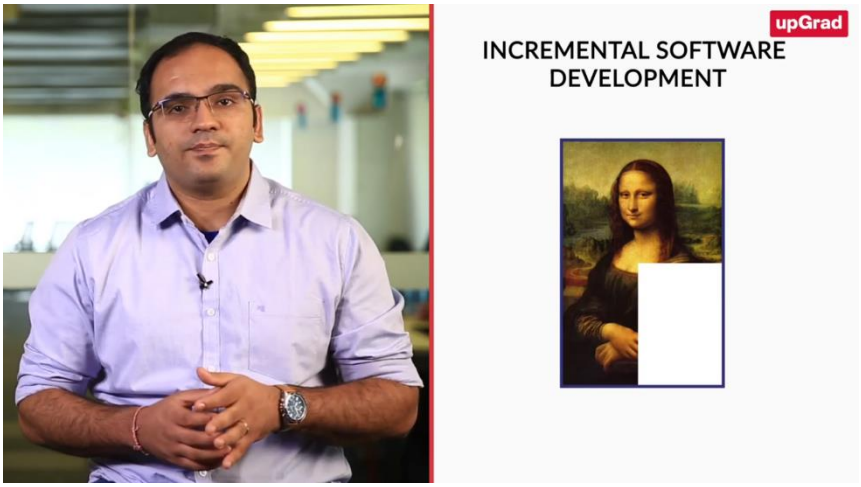
It comprises of various steps, like the conception of an idea, conducting detailed user or market analysis, product planning, design development, testing, deployment, and maintenance, if required.



Traditionally, it is done in a sequential manner, moving from one stage to another, in a linear way, just like a waterfall. Though the waterfall method works well if the requirements are clear in the beginning. You saw that changes could come in different forms, that's delaying, or sometimes even scrapping the entire project.



So, the agile methodology came as a solution incorporating changes and operating in an incremental and iterative way.



Then to learn about agile, you first discovered about incremental software development in which features are developed in an incremental manner where each and every increment is a hundred percent complete. This is done when your customer requirements are clear.

Later, you learned about iterative software development, where features are delivered in an iterative manner taking customer feedback into consideration. You also saw how both incremental and iterated development methods have their advantages in dealing with changes, unpredictability, and variability in the system, and how agile incorporated both incremental and iterative ways of product development.

VALUE PROPOSITIONS OF AGILE METHODOLOGY

Value		Traditional Methodology	Agile Methodology
Visibility	○ Is the release on time? ○ What is the software quality?	Not clear during the development	Clear even during development
Adaptability	○ Can we account for technology changes? ○ Can we incorporate changes?	No flexibility to account changes	Highly flexible to changes
Business Value	○ When do we deliver value to customers?	Only at the end	In every increment and iteration
Technical Risks	○ Does tech stack work together? ○ Is it the right functionality? ○ Can the expected performance be achieved?	Cannot be mitigated - long development cycles	Can be mitigated - short development cycles
Social Risks	○ Can the team collaborate? ○ Can the team execute well and deliver on time?	Cannot gauge team's performance till the end	Can gauge the team's performance in short time
Market Risks	○ Can we incorporate market changes? ○ Can we incorporate requirement changes?	Difficult to incorporate changes	Easy to incorporate changes

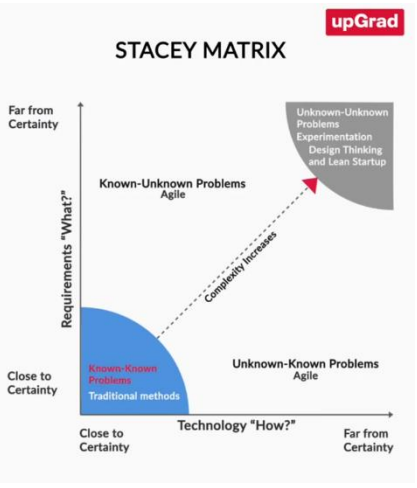
Later, you understood how effective agile is in comparison to traditional methodology. Agile has many value propositions in terms of high visibility, adaptability, business value, and mitigating social technical and market risks.

TECHNICAL TERMS - A PM SHOULD KNOW

1. Front end: Interface that users see
2. Data stores: Containers of data
3. Data centres: Contain data stores
4. Web servers: Connect data stores to front end
5. Offline storage: Data stored on local machines
6. Web architecture: Blueprint of components arranged in an order

You also learned a few technical terms like the front end, being the interface of what user sees. Data stores as the places where data is stored. Offline storage as the data that is stored on the local machine. And web architecture as nothing but the blueprint of how these components are arranged.

STACEY MATRIX

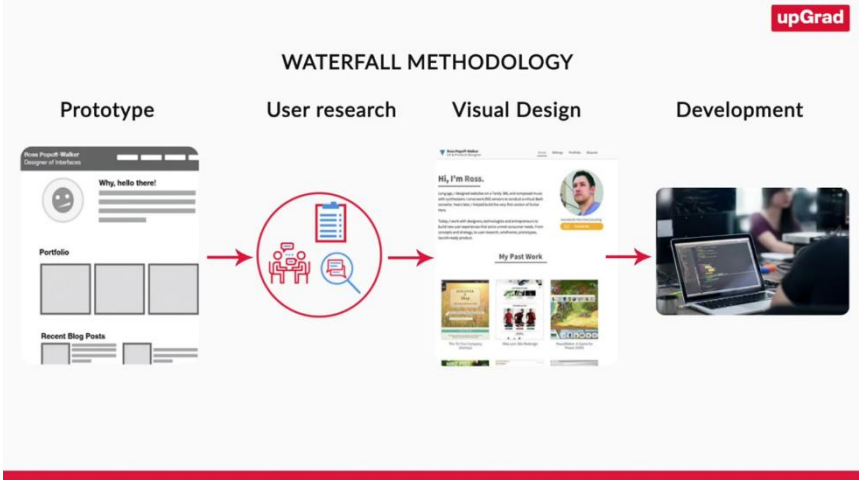


Later, you were introduced to the Stacey matrix, which categorizes problems into buckets by whether the requirements and technology are known or unknown.

Requirement known technology known problems can be solved by traditional method, while the problems where both requirements and technology are known are solved by experimenting or implementing strategies such as lean start-up or design thinking. In middle cases where either the requirements or technology is known and the other is unknown, are solved by agile methodology.

In the end, you saw how a team of novices might need more checks whereas a team of experts can deliver more efficiently due to their expertise. So, every team must figure out the right way of product development which suits their needs.

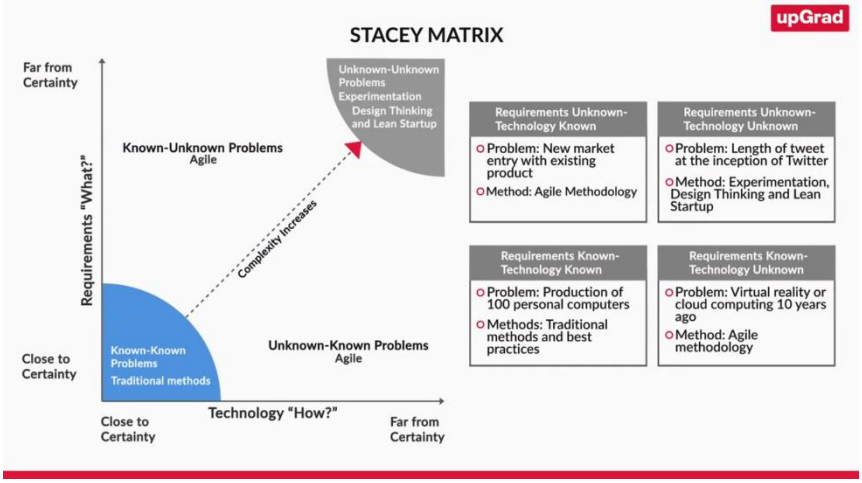
Now you've gone through the concepts covered in this session. Let's recap the examples you saw to understand these concepts better.



For traditional methodology, you saw how a project at Indy's is done with the design first approach, where they froze upon a design and visual elements, and then later develop them.

Next, you saw an example of wireframe and mock-up, which are nothing but iterations in an iterative development process. To understand agile as both incremental and iterative, you saw how the moving car feature in an Uber was developed in an incremental and iterative way.

To learn about big design upfront, you saw how web architecture is fixed. To understand technical terminology, you saw the example of a YouTube page for front end and see video section to understand offline storage.



To understand the Stacey matrix better, you went through two examples. The first one was for known-known problems where you saw the production of a hundred personal computers. The second one was an unknown-unknown case where you saw how the length of a tweet would have been decided at the inception of a Twitter.

You saw two more examples. One was virtual reality or cloud computing as it was 10 years ago, as a requirement unknown, technology unknown example. And the second one was an established product entering a new market as a technology known, requirement unknown problem.

With this, the session comes to an end. Next time, we will learn about scrum, which is one of the agile methodologies. Until then, keep learning and have fun.