**w02-01-RequirementsCapture-2024**

0:06  
Now you have seen from our last set of slides, I had this slide in last week and in these next few weeks we're going to be walking down these series of steps where today we're looking at requirements definition or requirements capture or as it's sometimes called, requirement solicitation.

0:28  
Now the idea of capturing requirements is to basically determine what we're going to build.

0:35  
Now, if you remember in the discussion last time, if you don't have requirements, there can be a disagreement between you and a client where you assume one thing and they assume another.

0:46  
So we want to remove ambiguity, meaning a misunderstanding that implies that the requirements need to be written in plain English, clear English, English that the stakeholder and you both understand concerning what should be built.

1:04  
So the purpose of this is, as already said, to avoid conflict between the developer and the customer and to make sure that you're building the thing that the customer actually wants.

1:16  
Now normally requirements or they could be user stories as we'll discuss later, are part of the agreement that the software is finished.

1:26  
So it's very important that you make sure they're written correctly.

1:31  
If they are vague, as we'll discuss later in the lecture, that will cause you problems because you could say I have fulfilled it and somebody else says no you haven't.

1:41  
So we need to bear in mind that we're going to deliver against these.

1:45  
Imagine if you're in a court of law.

1:47  
If the law is very obvious and clear, it's clear what the decision is.

1:52  
And the same is true with requirements or user stories.

1:55  
So we want to minimise the risk of conflict.

1:57  
Now you can't speak to your customer and then disappear into a broom cupboard and then reappear with the software.

2:04  
You keep needing to speak to the customer to double check that they understand what you understand.

2:10  
There needs to be an ongoing trust.

2:12  
Nevertheless, if the requirements are written carefully, it should be clear to both parties what is going to be built now.

2:22  
Requirements capture, or is it sometimes called requirements Solicitation, is the process of formally capturing these requirements.

2:30  
So we start off by identifying who these stakeholders are.

2:34  
These are people who have something to do with the project.

2:37  
It might be the end users, it might be the IT sys admin who are responsible for installing the software.

2:44  
He might be somebody in the company who's affected by the software.

2:48  
So these are the stakeholders.

2:51  
Then we perform some kind of requirements analysis.

2:54  
The idea here is when we're starting a piece of work, imagine somebody's brought us in and they've asked us to build an application, we may start by reading the technical documents.

3:06  
If we don't, then we enter the discussion with the client and having no knowledge at all.

3:11  
Now, in some cases there aren't any technical documents, but in a lot there are.

3:16  
So there might be documents about an existing system, there might be documents about a process.

3:21  
It's not about a piece of software, but it's about an ongoing process.

3:25  
And you can read those documents before you talk to the stakeholders.

3:28  
Remember, the problem with stakeholders or customers is they tend not to be aware of software engineering as in how it works.

3:37  
So it helps a lot if you start to understand what it is you're building even before you talk to them.

3:45  
Then you talk to the stakeholders and over one or more or many discussions, depending on the complexity of the project, you are hopefully going to, well, collect a series of user accounts or user stories.

4:00  
Now, it might be to start off with your confirming or your assumptions in the documents you've already read.

4:07  
By the way, don't assume things.

4:10  
You may think, yeah, yeah, this is obvious.

4:12  
But if you're in even a tiny bit unsure, ask the customer.

4:17  
It's much better that you say, oh, yes, it's written here.

4:20  
Does this imply that?

4:22  
And the customer, the customer could say yes or no, That's not actually right.

4:26  
So it's very important to just share the assumptions with the stakeholders.

4:33  
And what you normally do is you collect these user requirements and then you discuss them with the customer.

4:41  
So if you go for a piece of work, it could be that the customer has already got a set of user requirements.

4:48  
I've seen that before.

4:49  
And you read those, you read the technical documents, you go back to them, and then you start work by adding requirements or maybe updating the ones they've written.

4:58  
You basically are starting at a given point anyway, and you go round in loops talking to the stakeholders until you feel that you've got most of them.

5:08  
Now you can't assume things or hide things because they will actually 'cause you trouble later.

5:15  
In terms of, let's say you assumed you understood the data model inside and you didn't really.

5:20  
You remember that was one of the big problems of Police Scotland, misunderstanding the complexity of the system.

5:27  
Now one other thing that you need to consider is allowing updates.

5:31  
What tends to happen with software builds is you start working on the build and then somebody suddenly wakes up and thinks, oh, oh, actually we need this feature.

5:41  
And so it's really useful to be able to bring in extra requirements during the software build.

5:48  
And that's where an agile or agile like software life cycle is helpful, where you haven't frozen everything at the beginning.

5:58  
OK, so how do you identify the stakeholders now?

6:02  
It is effectively a group of people who are affected by your software directly or in or you know, relatively directly, IE the people who are actually going to use it.

6:15  
They're the users.

6:16  
They're really important to please.

6:18  
Now, they might not be the most a powerful group of people in terms of authority or in terms of who pays you.

6:28  
But if they're not happy, then eventually it will be the the client, whoever's paying for it.

6:33  
If it's not, the users won't be happy.

6:36  
And then who's paying the bills?

6:38  
That could be the end user.

6:39  
It could be somebody else.

6:41  
So you need to understand who are the users.

6:44  
It could be another company.

6:45  
If you remember the discussion last week, you can have several layers of companies where one is delivering into another, which is delivering to another.

6:53  
So you need to understand who is the important person in this or the important people.

6:59  
The other people who you could consider the project manager.

7:02  
Now they might not be a user, this is perhaps somebody in a different company, but you need to make sure that they understand what you're doing.

7:10  
They're an important stakeholder, A-Team leader.

7:13  
This again, could be a technical head in the other company that you're delivering into an IT manager or group of IT people.

7:22  
They may well be responsible for the software infrastructure where your application's being deployed.

7:28  
If they aren't happy, your application may not be used at all.

7:32  
And the CEO, well, we don't talk to them normally, but if things go really badly wrong or if their wishes aren't fulfilled, that could affect us.

7:42  
So as you think about these people, you then are trying to think about how important they are in terms of their power and their influence on a project, and you end up with a little graph like this.

7:54  
So the idea is as a sketch, you're trying to work out who you should be talking to and at what level.

8:01  
So on the X axis here we've got level of interest, on the Y axis we have power.

8:06  
So you might imagine the CEO is somewhere up here, lots of power, but probably doesn't care about your software unless it goes badly wrong.

8:14  
Whereas the users may well be down here, low, lower power, high level of interest, the IT sys admin, they're probably somewhere up here.

8:26  
They have got a lot of power.

8:27  
They care more than the CEO, but maybe not that much depending on the software that you're building.

8:33  
So when you are brainstorming stakeholders, you probably want to write them down and then put them in a chart like this.

8:39  
It's a guess as to where they go, but that will inform when you talk to them and how you talk to them.

8:45  
For example, if it's IT sys admin, you might have one or two meetings with them just to collect particular requirements.

8:52  
Make sure that you've understood the software framework that you're deploying into.

8:56  
Their concerns have been heard, they feel happy.

8:59  
Whereas users, you may well talk to them a lot because you want to make sure that the software is what they want now.

9:08  
Yeah, users, as I said already, maybe outside the organisation, they may not be in the company you're delivering into, maybe some other company.

9:17  
What's often done to try to talk to the users is you could have a survey, you could create a prototype as well and you could share that prototype with the users.

9:28  
They could play with it and test it out.

9:31  
It could be an alpha version, some of the features in there and you quickly understand more about what the users want.

9:38  
Now, as I've said already, powerful stakeholders can cause failures, meaning if they're not happy, even if they've got a low level interest, if they're not happy, your project doesn't succeed because it hasn't been deployed.

9:53  
OK, here's another way of looking at stakeholders.

9:56  
You can order them in terms of their degree as it were.

10:00  
So the inner layer here is the direct users.

10:04  
So this could be the operator or the customer.

10:06  
And then around that we have the management or the analysts.

10:11  
So they're people who are going to use the software, but they're not directly using it.

10:15  
And then the IT management who are responsible for deploying it and they are important as well.

10:22  
I say they are important as well because as I mentioned last week, if you ignore them, yes, there are real cases where this has been done and software has not been deployed.

10:32  
Now, user requirements, there are essentially two ways of writing these things down.

10:37  
You've got user requirements and user stories.

10:39  
User stories are to do with agile.

10:41  
User requirements are to do with originally waterfall, but have been used in other life cycles which aren't agile now often in textbooks.

10:52  
So for example if you look at the textbooks that I've referenced, they won't include the user type.

10:58  
Now the user type has been included for many years in requirements that the Ministry of Defence uses and some safety critical projects.

11:08  
Personally I think that's much more useful than not including the user.

11:13  
So if we don't include the user, we have no idea who wanted it.

11:17  
So here the user could be something like the IT sysadmin requires the ability to remove all the files or something like that.

11:26  
So we're going to put in here the actual user, whoever the.

11:29  
So it could be an IT sysadmin, could be a user, could be a, could be some kind of system controller, some person, it's a real person and what they want to do.

11:41  
And so we write this down in as simple English as we can.

11:44  
Now occasionally you have a table of user requirements.

11:47  
And what happens is that you need to use special words.

11:52  
And so if you read the requirement itself on its own, it might be confusing.

11:56  
So what you may do is in such a case, have a paragraph of text that explains what the particular words mean.

12:04  
So somebody's read that and then they come to the user requirements and go, Oh yeah, I know exactly what they mean because they've defined what that word means above.

12:11  
That's a easier way of making the sentences shorter.

12:15  
So we want single sentences, not long, super long sentences.

12:19  
Keep them shorter.

12:21  
Only one sentence.

12:23  
And yes, they should relate to a person.

12:25  
User requirements are not about a system.

12:28  
If we're writing about a system, that is a system requirement or software requirement.

12:33  
Now, as I'm going to mention later on in the lecture, you can use Waterfall where the system or software requirement is the answer to the user requirement.

12:42  
So if you write down user requirements and it says the system must stop, that's not a user requirement.

12:48  
It's from the user's point of view.

12:50  
If you think about walking into a shop and you want something and you think, I wonder, I'd like something that does this, please.

12:58  
That's essentially what we're trying to say with the user requirement.

13:01  
Now when we when we write these down, we don't have in mind the exact technology.

13:07  
OK, don't go in there.

13:09  
I really want to in React or something you're trying to just capture what does the person actually want.

13:16  
Now it could be that they have a non functional requirement.

13:21  
We'll come to non functional in a minute, which is to use a particular piece of software which you need to write down because the IT sys admin want you to do that.

13:30  
However, you should try to think about your technology choices after you've captured the design because it might be that the design suggests you should use some other technology.

13:43  
Right now, bad user requirements are vague ones.

13:48  
Here's one I see every year students write down the design.

13:53  
Must be intuitive.

13:54  
OK, what does that mean?

13:57  
I quite like VI.

13:58  
This is VI down here.

14:00  
To me, this seems perfectly reasonable.

14:03  
It doesn't tell you where which characters to use to control it.

14:07  
To me it's OK though, because I've memorised them all.

14:09  
That's intuitive to me.

14:11  
So if you don't like it, tough.

14:14  
Now do you see what I mean?

14:15  
The intuitive design is relative to a person.

14:19  
And so we never, ever, ever say anything about the design quality in the user requirements.

14:26  
What we do instead is we actually create a, a representation of the user interface and we show that to the customer and say, is that what you want?

14:36  
And then we agree on it and we build it.

14:39  
Now that's what the customer has required or wanted, and we've got the user interface design to show to them.

14:45  
So never say intuitive design.

14:48  
The other silly thing I've seen before is must be fast, OK?

14:53  
What does that mean?

14:55  
Is that a picosecond or is that 2 seconds?

14:58  
I don't know.

14:59  
So you need to be more precise.

15:02  
You could say needs to load within 2 seconds.

15:05  
That might be OK, Or it could be needs to work in, I don't know, few nanoseconds, which is extreme.

15:12  
But you must be precise if you're vague.

15:16  
The problem is it isn't fulfilled.

15:18  
It's like a discussion between you and the customer.

15:20  
We've fulfilled this.

15:21  
No, you haven't.

15:23  
Oh, this is intuitive design.

15:24  
No, it isn't.

15:25  
So then you're in trouble because you end up being billed.

15:29  
Don't be too restrictive as well.

15:32  
So what I'm trying to say here is you put in requirements that you think are there or needed, but they're going to make your life a lot harder.

15:40  
Like, oh, it must work in, I don't know, half a second.

15:44  
Whereas actually the customer doesn't mind if it's running for at least a second.

15:48  
So try to determine what is acceptable and do so in a precise way.

15:54  
If you're dealing with a browser, you want to say what version of browser?

15:59  
Don't say must work on all browsers because somebody, somewhere, I have a very old PC running Plan 9 and some very strange browser you've never seen before that doesn't work with your software.

16:10  
And they said, but you've told me it should work on this one, and then you're in trouble.

16:15  
OK, what do we need to do?

16:18  
We want to understand why the requirements needed.

16:22  
Now this is very important for user requirements, but it's also important as you'll see for user stories.

16:28  
Now, user requirements themselves don't have a way of saying what the benefit is.

16:33  
So normally you capture the user requirement and in a table you'd write down the benefit.

16:38  
So why is it needed?

16:39  
What's the point of it?

16:41  
Now the benefit is important because then you know how important that user requirement is.

16:47  
If it's obviously very useful to the user, then that's higher priority.

16:52  
So capturing benefit is a good idea.

16:55  
You may want to add tags, so, you know, hashtag, whatever.

16:59  
Yeah.

16:59  
Well, with user requirements, what we'd do is we'd say something like security user interface.

17:05  
And so you can search through your long list potentially of user requirements for ones that affect particular systems.

17:12  
So security could be some kind of security compliance or user interface could be some sort of user interface features.

17:19  
Now you haven't actually given them the user interface, but you're just flagging to you, the customer, the other developers.

17:25  
This one is to do with the user interface.

17:30  
OK, so when you're analysing requirements, what you do, we've already said analyse existing information.

17:36  
So if you've got existing process documentation, technical documentation, maybe an existing version of the software, other products on the market, there might be something a bit like what you need and you look at that for inspiration.

17:49  
It could be something that isn't exactly what you want but has features that you do want.

17:55  
So for example, looking at a Google Calendar when you're building an appointment piece of software, I'm not going to copy Google, but you're thinking about some of the features.

18:05  
Then what might happen is that if you are building software for somebody else, they give you a so-called request for quotation or RFQ and in there there may be some initial user requirements for you to bid against.

18:21  
So you say, yeah, yeah, we'll build this for you, for, I don't know, half £1,000,000 or whatever it is.

18:27  
You then end up often brainstorming ideas.

18:30  
So you're thinking about all the inputs you've been given from the stakeholder, the technical documents, you're writing them down and you end up with an initial set of requirements.

18:40  
We try not to delete user requirements when you're first brainstorming them.

18:46  
You might delete them, but as soon as the customer's seen them, you effectively score them out and say, we're not going to build this one.

18:54  
But you don't actually really delete it because occasionally what happens is people change their mind and then you can say no, no, we did have this, but we decided not to build it, didn't we?

19:04  
Because you remember that discussion anyway.

19:08  
So you want to consult the users once you've got your initial set of user requirements, and you probably want at least one discussion with them.

19:16  
Now, often this is done in workshops where you might have a collection of people representing the final users, the fight well, the stakeholders.

19:26  
You might have a separate one with the IT people.

19:29  
You may have surveys.

19:30  
If you want to cover a lot of users.

19:32  
You may also have focus groups.

19:34  
They're essentially many ways of trying to understand what a large group of people want.

19:41  
Yeah.

19:41  
So you invite interested stakeholders to the workshop.

19:45  
These are people who really care about the software, people who are going to give you some good feedback.

19:50  
Hopefully this is a team of three to five.

19:53  
Don't go very big in terms of numbers because it's much harder to hold a discussion with them.

19:59  
You may want to have, yes, some input from people who are very focused on what the customer wants.

20:05  
It could be some kind of product owner role, somebody who knows the customer very well, technical input.

20:12  
So there might be somebody from the IT team there, or maybe infrastructure.

20:16  
The project manager's going to be there, hopefully because they need to be happy.

20:21  
And with any meeting, we want to distribute information before the meeting so people are thinking about what we're going to talk about.

20:27  
Then we have the meeting.

20:28  
So give them the initial version of the requirements.

20:32  
Then you have the meeting.

20:33  
Now probably what happens is you have a, a list of comments, you update the requirements, you give them back to the client, you have another meeting.

20:40  
When I've done this commercially, we've had at least two meetings for a complex set of requirements and then off we went.

20:47  
Occasionally, very occasionally, for simpler projects, we maybe had one meeting and then in exchange by e-mail over some things.

20:55  
But you must be sure that you both agree who the stakeholders are, yes.

21:01  
Who do they agree with you about what the software is?

21:06  
OK, Surveys.

21:08  
How would you conduct a survey?

21:10  
So ideally you want to distribute these surveys to the main group of users.

21:16  
Now don't don't have very long surveys because what happens if you have 50 questions or so?

21:22  
The survey won't be completed.

21:24  
You essentially really want somewhere, I don't know, 11:50 questions.

21:29  
A very small number of questions won't tell you much, but if you go too long you'll just end up with surveys that aren't completed.

21:39  
You do want a selection of questions where there are options that will be more productive in terms of collecting answers.

21:48  
However, you probably want one or two open questions near the end where people can just tell you what they think.

21:54  
Now it's harder to collate those, but once they finish the survey, that is potentially useful.

22:01  
Now, when you're writing survey questions, be careful how you write them because with surveys you can be suggestive.

22:07  
There's a rather funny comedy sketch where that the survey order is reversed and at the end the answer is different because of the questions being suggested.

22:17  
So try to think about what your questions are doing.

22:20  
It can be with very large sets of people that you use AB testing.

22:26  
So this is where you essentially direct some part of the people who might be interested into a test environment.

22:34  
They don't know it's a test environment and they are then has to select one of two things.

22:39  
So it's one or the other, A stroke, B testing and they basically tell you which one they like.

22:45  
They may know they've told you that or you may hide it from them through an Internet portal.

22:50  
Anyway, that has been used for say, determining colours for search engines and what colours are more productive.

22:59  
OK, there are other requirements.

23:01  
Elicitations collect techniques so you can look at the interface and see how it connects with the other software.

23:11  
What do I mean here?

23:13  
If it's not a graphical user interface, it could be it's connecting over the network or it could be it's linked to another piece of software when it's compiled.

23:24  
So you'd look at the user interface design and you discuss that because that's where you're going to connect together.

23:30  
And then you may understand some more features.

23:33  
You may have a focus group.

23:34  
So these are group of users who come to you and tell you what they think in such a manner that you can discuss with them.

23:42  
You may have interviews.

23:43  
This is where you talk one-on-one to somebody and you discuss with them.

23:47  
You've got interview questions now, that's more work, but potentially it might be more useful in some cases.

23:55  
And then lastly, observation.

23:57  
So you may go into the company and see what they're actually doing.

24:02  
Now.

24:02  
This is actually quite helpful.

24:04  
I've done this before.

24:05  
I've wandered around in a company and I've seen what they're actually up to.

24:09  
Oh, can you show me that?

24:11  
And then you quickly understand the items or the problems that they're facing that formally you assumed you knew, but in fact you didn't.

24:22  
So if you do have the opportunity, you can watch how they interact with their existing software, their problems, their issues, or potentially you give them something to play with, a prototype and then you watch what they do with it.

24:37  
Now, once we've got the requirements, we have to track them with a unique identifier.

24:41  
We can't be confused about which requirements it is.

24:45  
Now, we haven't had any lectures particularly I think on primary keys yet, have we?

24:49  
No.

24:50  
OK, so with the relational database what you have is you have a unique key for a particular row.

24:57  
This is called a primary key in a relational database.

25:00  
We do the same thing with user requirements, so normally we'd call them 1234.

25:06  
If you're using an online system, the online system will probably generate the number for you.

25:11  
So for example, later on in one of the labs, we'll use Azure DevOps and you'll see it'll generate a number.

25:18  
If we're doing this in a document and we're not using one of these tools, I tend to use UR1UR2UR3.

25:26  
Now, the nice thing with this is that it's absolutely clear which one we're talking about.

25:31  
Oh, it's UR 22.

25:32  
Oh yeah, I remember that.

25:34  
And also when we're talking about in a technical document we can say that we have fulfilled UR22 rather than copy pasting the user requirement all over the technical document, which would make it more unpleasant to read.

25:47  
As I've already said, don't delete user requirements once you've shown them once to the customer.

25:54  
When you delete them, what you are going to effectively do is scroll them out and say they are deleted, but you don't really delete them because they may change their mind.

26:05  
And if you want to update them, unless the customer's absolutely certain that they want to update an existing one, I would suggest you just delete one.

26:16  
So delete it and create a new one.

26:19  
That way you can always go back to the original one.

26:21  
You're not losing any information now.

26:25  
Often students are confused what a functional user requirement is and a non functional user requirement is.

26:31  
So a functional user requirement is a feature that you build and you can test.

26:37  
A non functional 1 is a design choice or a technical constraint.

26:43  
So it's not a feature, it is an idea that we're fulfilling.

26:48  
So for example, it must have a graphical user interface.

26:52  
That is a non functional requirement.

26:55  
It must be compliant with cybersecurity Plus that is a nonfunctional requirement.

27:01  
We're not building a software feature to fulfil that.

27:04  
It's essentially telling us what the boxes that we have to fit into.

27:09  
So when you're writing down user requirements, think to yourself, is this a software feature that I'm going to test or is this something I have to meet as an overall requirement?

27:20  
If it is, it's a non functional requirement.

27:22  
It could be the hardware, so it has to run on a phone of some type or variant.

27:26  
That is nothing to do with a feature.

27:28  
It's just you've got to fit in that hardware.

27:32  
OK, here are examples of user requirements.

27:35  
So you can see I've got the user, user requirement number 132, whatever.

27:40  
The user requires the ability to create their own account.

27:44  
That's pretty clear.

27:45  
And that's a functional because that's a feature.

27:47  
The user requires the ability to reset their password.

27:51  
Again, that's a software feature.

27:53  
It's functional.

27:55  
The user requires functionality on an Android installation.

27:58  
That is a non functional requirement.

28:00  
We're saying that it has to run on Android.

28:03  
It's not a feature of software that I'm going to build.

28:06  
It's just we fit in the box that is Android.

28:10  
OK.

28:11  
Now as we said in the previous lecture, requirements must be traceable and tested.

28:17  
So the idea here is we have to fulfil our user requirements.

28:22  
So we want to map these user requirements to the software features.

28:26  
We want to say you wanted this.

28:28  
Here it is.

28:29  
We built it.

28:30  
And often with safety critical systems, you reverse map it as well.

28:34  
So you'd say, here's the software feature.

28:36  
Did I ask for this?

28:37  
No.

28:38  
Oh, can you get rid of it, please?

28:39  
Because that might cause my nuclear reactors to meltdown or something like that, or my plane to drop out the sky.

28:45  
So safety critical systems you reverse map.

28:48  
Normally with other systems you wouldn't.

28:50  
You'd just map in One Direction.

28:54  
Now we at the end, we use acceptance tests to verify the functionality.

28:59  
So these acceptance tests are where somebody will run through the features that they expect and they'll manually normally test them out.

29:06  
Now we do a lot of automatic testing, but when we're finished, the last set of tests are normally manual.

29:15  
So we can use user requirements within a waterfall or waterfall like life cycle.

29:21  
And if we do, what we end up with is we end up with a hierarchy.

29:25  
So we might have a high level user requirement like, oh, it must be able to calculate this value and you think, OK, then under this we may have one or two or three smaller user requirements that tell you more about what that means.

29:39  
And then from the user requirement, as you've already heard me say, the fulfilment of that is a software or system requirement.

29:46  
So that's something you're going to do.

29:48  
And then from the thing you're going to do, there is a functional requirement which is a smaller thing.

29:54  
And then from the functional requirement, we have a matching user acceptance test.

29:58  
Now you may see this setup in engineering.

30:02  
So if you're using, let's say, civil engineering approaches, there may well be this set of tests in there.

30:11  
As we've already said, engineering and software engineering uses the same set of life cycle ideas, right, V life cycle, we mentioned it last time.

30:22  
The problem is we can't easily change direction.

30:25  
So we've got a hierarchy of requirements potentially with high level 1, lower level 1, functional requirement and then your feature.

30:35  
And if you're having to move everything around, you're changing this whole requirement stack, it becomes quite difficult.

30:41  
You might lose or emit some benefit as well because the benefits aren't baked in.

30:47  
A benefit remember, isn't part of requirement statement.

30:50  
You have to discipline yourself and keeping them in a table with the requirements.

30:55  
However, yes, requirements can be used for waterfall of the life cycle.

30:59  
Now, what is the problem with user requirements?

31:01  
Other than them being potentially cumbersome to use?

31:05  
The other problem is you end up with a behaviour like this.

31:09  
This is a so-called S curve.

31:11  
What am I talking about?

31:12  
So here's the time at the bottom.

31:14  
So developers start working.

31:16  
No user requirements have been fulfilled.

31:18  
Developers are still working.

31:19  
Three weeks later, still no user requirements fulfilled.

31:22  
A little bit later, still none.

31:24  
Now the client's sitting there thinking, but you people really lazy, you haven't fulfilled any user requirements yet.

31:30  
And also they can't comment on what we're building.

31:33  
And then we come closer to the deadline and we're now fulfilling a few and now we've fulfilled them all.

31:38  
Oops, we've finished.

31:40  
So that tends to be what happens because the user requirements might describe a feature where the software can pieces underneath sit in all the different layers.

31:50  
There could be a back end service, there could be a middle layer, there could be a top layer.

31:54  
And so it's hard to deliver that early in the project.

31:58  
You can only deliver it late.

32:00  
And so it's hard for the client to give you any feedback until it's basically too late because you delivered it.

32:07  
So this is bad.

32:08  
We want to try to move away from that as much as we can.

32:12  
We'll still have it as a problem, but we want to ideally deliver some things earlier so that the client can give us some feedback.

32:23  
We need to prioritise our user requirements.

32:25  
So that could be in terms of, yes, how long are we going to take to build it?

32:31  
How hard are they to build?

32:33  
How much does the person want them?

32:34  
That is the benefit and then we may want to think about a minimum viable product.

32:41  
This is the smallest set of features that could still be useful to the client.

32:46  
Getting that into their hands is potentially really useful because you learn how they use it and it might change the next user requirements that you actually implement.

32:56  
You need to think about the cost.

32:58  
These benefit.

32:59  
It's if you've got a project that is about 95% complete and it's going to cost a lot more to do.

33:06  
The very final thing you might tell the client actually we're nearly there, but do you really want this?

33:12  
It's going to cost you, I don't know, another half million or whatever that may cause scope decisions to be made.

33:18  
Scope.

33:19  
By scope, what we mean is we decide not to build some things.

33:25  
When you've delivered software, you often deliver again version 1.2, version 1.3, version 1.4.

33:33  
So you don't actually have to have absolutely everything in the first version that you've deployed anyway.

33:40  
Now, one way of prioritising user requirements is the so-called Moscow prioritisation.

33:45  
So this is the must have, should have, could have, will not have.

33:49  
So again, it's a brainstorming strategy where you've written down the user requirements and you just mark them up as MSC or W and yes, you show to the stakeholder and you agree between you.

34:01  
Is that about right?

34:03  
OK.

34:03  
I've also seen other ways of doing this like high priority or number system.

34:09  
Anyway, Moscow is one way of doing it, of documenting formally that you have decided a particular priority.

34:17  
Great.

34:17  
So that's user requirements and associated life cycles to do with them.

34:23  
Does anybody have any questions or comments before I move on to agile?

34:30  
No.

34:31  
OK, then I'll be on to agile.

34:33  
So with agile, what we do is we use user stories.

34:36  
Now whoever came up with this obviously thought to themselves, we don't have a way of connecting the benefit to a user requirement.

34:43  
We don't.

34:44  
So a user story is formally 3 things.

34:48  
So we've got as a some kind of user.

34:51  
Now this user, unlike a user requirement, can be a persona.

34:56  
What do I mean by a persona?

34:57  
It could be something that isn't a person.

35:00  
So for example, a fridge is not a person, but a fridge might want to connect to a service to report something.

35:07  
So you could literally have as a fridge.

35:10  
I want to interface this network so that I can offload my compressor data or something.

35:18  
So this could be a real person or it could be a persona.

35:22  
They want something and again, it needs to be written in a succinct way and they need to say why they want it.

35:30  
Occasionally I see students who don't give benefits.

35:32  
So there's no, the benefit that is not good because then there's no information as to how how to prioritise it, right?

35:42  
Remember, the benefit is telling you is it super important?

35:46  
Is it less important?

35:47  
How is it connected to the system?

35:49  
So you need to capture all three things in one sentence.

35:53  
Now, don't make these sentences very long because some of the systems you'll come up against won't let you.

36:00  
If you use Azure DevOps or some of the other requirements capturing pieces of software, there is actually a limit to the length of the user's story.

36:08  
It's also hard for somebody to understand a very, very long sentence.

36:12  
So lastly, these user stories have to be complicated, yes, potentially, but not too much.

36:21  
We need to be able to fulfil them within a Sprint, probably a few days.

36:26  
So if you make this user story very complicated, it becomes, in agile software terms, it becomes an epic.

36:33  
And an epic is sort of a top layer thing.

36:36  
It's not something there's a user story that we're going to implement inside a Sprint.

36:41  
Now user story should be vertical slices of functionality.

36:45  
What do I mean by this?

36:47  
We're going to add some features to our software.

36:49  
So if you imagine you had a user interface, you had a back end.

36:52  
So that web service for user storage fulfilment, you may add a little something in the user's interface layer, you may add something in the web service.

37:01  
So hence it's a vertical slice that is from the top down.

37:05  
We want to deliver functional updates.

37:09  
So we don't want no software, no software.

37:11  
We're basically saying here it is, we've added this new feature and it needs to be small enough.

37:16  
The user's story that is that our development team can complete it within a few days to weeks or so.

37:23  
It should be completable with inside a Sprint.

37:26  
Remember, a Sprint is somewhere between 2:00 and 4:00 weeks.

37:29  
It needs to be clearly written so that we can estimate the time.

37:34  
If it's not clear enough, we can't estimate the time, and that's not good.

37:38  
Remember, it's our unit that we're using inside our sprints.

37:45  
We are going to treat them in the same way as we do with user requirements.

37:49  
So we have them as numbered lists here.

37:51  
I tend to use US1.

37:53  
US 2 epics are essentially a high level user story that's sit above user requirements, but other than that we avoid hierarchies.

38:04  
So we don't have high level requirements, low level requirements, system requirements, so on and so forth.

38:10  
We just have user stories.

38:13  
Our development team might have a few tasks that they fulfil to fulfil the user stories.

38:17  
So as you may see in some of the tools, you have a task under the user story, but we try to avoid hierarchies as much as possible.

38:25  
And we also allow potentially their creation during the development process.

38:30  
So while we're building, we're talking to the client.

38:32  
Oh, oh, you wanted that?

38:34  
OK, I'll write it down.

38:35  
Here's the new user story.

38:36  
Do you agree with that yet?

38:37  
Great, it's in the system now.

38:41  
We want to actually put in a bit more effort to avoid this S curve that I showed you before.

38:48  
And to do this, we may want to write down user stories which the client hasn't actually asked for, but they bridge the gap between what the end feature is and where you are if you're beginning.

39:00  
So for example, you might say, as a user, I want to be able to save data in the database so that I can restore the information, something like that.

39:08  
You can then say to the client, look, we've fulfilled it.

39:11  
We're now pulling the data out or saving the data into the database system.

39:17  
So we end up with a burn down chart.

39:19  
And again, we'll come down to burn down charts later in the lecture series where we're talking about project management.

39:25  
A burn down chart is essentially where we're trying to understand our rate of completion of user stories.

39:32  
There's a burn down and a burn up.

39:34  
One is completion, 1 is essentially completion, but the other way around, cumulative amount of work done.

39:42  
So we want to order these user stories as well, because some of them might need to be built first.

39:49  
We try as much as possible not to have them interfering with each other, but some of them will do.

39:57  
Here's one set of suggestions about user stories.

40:00  
Ideally, they're independent.

40:01  
Occasionally, as I've already said, you're stuck and they can't be.

40:04  
But ideally they are negotiable.

40:06  
That is, you can discuss how you may implement this, what the benefit is valuable the customer must.

40:13  
Value what you're actually doing.

40:15  
Estimatable, meaning that you can try and guess how long it's going to take you to build, small enough that it can fit inside a Sprint, and testable, meaning you can build a test that cheques have you fulfilled it.

40:27  
Again, thinking about vague ones.

40:29  
If it's vague, you can't build a test against it.

40:34  
Now, if you're brainstorming, if you're writing down user stories, you can look at them and think, hang on a minute, that's going to be too much work for a Sprint.

40:43  
And so you end up splitting user stories.

40:46  
So don't worry if you get them wrong the first time you write them down and you look at them and you try to understand, is this too complicated?

40:53  
If it is, split it so it could be steps in the workflow.

40:58  
It could be that there are some different pieces of a business rule.

41:02  
It could be that there's a simpler version that might work.

41:06  
It could be that you've got different types of data you're going to deal with, like you're going to list some users, or maybe you're going to list some features or something inside the software.

41:17  
It could be that you need to split them into different parts.

41:21  
Maybe create, read, update, delete.

41:23  
Features need to be separate user stories.

41:26  
Depending on the complexity of what the user story implies, it could be scenarios of some kind of use case.

41:32  
We'll come to use cases in a bit of how the user's going to implement.

41:36  
So use case diagram is potentially useful for thinking about user stories and maybe fragmenting them.

41:43  
And we all see one of these in the future, we might want to split them because we've got some kind of technical or functionality spike, meaning there's something very hard we have to do.

41:55  
And so we need a developer user story where we're trying to explain a feature that's needed, but it fills the gap.

42:02  
So we step forward.

42:06  
We want to group user stories in the same way we did with user requirements.

42:11  
So we give them tags.

42:12  
It might be security, it might UB user interface data processing.

42:15  
This will help the development team to understand what they're going to do with that user story.

42:23  
Now the epic is, as we said, at the top of the user story stack.

42:29  
So the way agile requirements or agile user stories work is that, yes, we have an investment theme where this is we want to invest in security or something.

42:39  
It's a vague theme that the customer wants.

42:42  
The epic then is an answer to that, a little bit like the system requirement is an answer to the user requirement.

42:48  
And then above the epic, yeah, after the epic, sorry, we have user stories, epic, a broad idea, then user stories.

42:55  
And then under user stories, we may have tasks they say may have a task is just used for parcelling out some of the work.

43:04  
You don't have to have a task.

43:05  
I've built quite a few bits of software where we started from user stories.

43:09  
We didn't have epics, so we had enough user stories and then we just got on with it and they weren't that hard, so we didn't have many tasks.

43:19  
Now the task is treated in the same way user story is by the developers in that they build it and they test it works and then they sign off that task and then there'll be a user acceptance test for the user story itself.

43:33  
So here we go.

43:34  
Here's an investment theme, security.

43:36  
The epic could be user authentication, so that's a feature that's needed.

43:41  
And then the user story, one of them under this epic will be as a user, I want to be able to create an account so that I can securely access the ServiceNow.

43:51  
Obviously, under authentication, you're gonna have to create an account, you're gonna have to have password authentication, you're gonna have other user stories.

43:58  
I've just added one that sits under here.

44:04  
OK, so as I've already said, you might not need tasks.

44:08  
It depends on the complexity of the user story, how much time the development team is going to take to complete it.

44:13  
You may not need epics, depends on the complexity of the project.

44:16  
And also you may have a hybrid system where you have user requirements to start with and then you go to user stories and you use your user requirements for high level discussion with the client because they might find that's easy to work with.

44:29  
I've seen this several times.

44:32  
Yes, our user, sorry, our tasks should be smart.

44:37  
So these are the things that the developer's going to work on the tasks.

44:41  
So rather similar to the user stories, they have to be specific.

44:44  
What is the developer or developer's going to do?

44:47  
Measurable as in how are they fulfilled?

44:50  
Achievable.

44:51  
We've got to fix them up in a couple of days.

44:53  
Relevant.

44:54  
That is, they are relevant to the user story.

44:57  
If they're not, that's no good.

44:59  
Time.

44:59  
Boxed as in they're limited in the amount of work that's needed.

45:05  
We can use mascara to prioritise user stories if we want, and we can look at the benefit.

45:13  
It could be that we use a numbering system.

45:16  
Another thing we do with user stories is we often associate story points with them.

45:21  
Now a story point is some sort of indication of how hard is it to fulfil the user story.

45:29  
What I suggest in this course, and I've just taken it from somewhere else, is to use this Cones modified Fibonacci series.

45:35  
What we're doing is we're trying to guess how hard is it.

45:39  
Now it might be that the user requirement is hard and you know it's hard and therefore you give it a big number.

45:46  
It might be that you are not sure if it's hard or not and you give it a bigger number.

45:52  
So the bigger number is a combination of difficulty and misunderstanding risk.

45:57  
Effectively.

45:58  
If you're sure that it's easy, then give it a low number.

46:02  
Now during the builds you may end up giving well during the Sprint meetings.

46:07  
You may end up giving different story points.

46:10  
The more experienced you become, the better you are estimating story points.

46:15  
Now story points are somehow connected to ours worked, but they aren't the same.

46:21  
Sometimes people do map them directly and that maybe isn't a good idea because depending on your development team you may have, I don't know, 5 or 10 developers or maybe 2.

46:31  
And so story points don't necessarily map to ours, although there is some sort of relationship.

46:38  
Again, that's an important discussion point.

46:41  
When you come to a burn down chart or burn up chart, you're adding up the story points that you've completed.

46:47  
And so you try to look at your development team and say, are we outputting a consistent number of story points depending on the number of people working on this project?

46:57  
Yes, we can use hybrid methods, meaning we start off with user requirements capture and then we go to user stories.

47:05  
This is often used with some commercial teams.

47:10  
They can be built more disciplined in news epics.

47:12  
What I've found with epics is that potentially customers don't know how to use them, whereas they're happy enough with high level user stories.

47:23  
All right.

47:24  
So the conclusions of this lecture, it's vital to identify stakeholders.

47:30  
If we don't, then potentially we end up with software that doesn't work or isn't what they want in terms of the feature set that's bad and then we're not paid.

47:41  
We want to understand what the stakeholders want from the software, meaning we could build a prototype version and give it to them.

47:49  
We could give them a user an interface, user interface that's interactive and give it to them and ask them what they think.

47:56  
We need to describe these features carefully, otherwise we're not sure what we're building and the client's not sure what we thought they want.

48:07  
We need to deliver software that matches these deliverables, and that means that what we write down has to be written down carefully, otherwise we won't know what we're building in terms of fulfilling them.

48:23  
OK, so that is the end of my slides I believe.

48:28  
Does anybody have any questions or comments you can ask me?

48:30  
Questions about real problems or any other ideas I've discussed?

48:34  
Yes, yes, yes, yes, that's right.

48:45  
So yes, the idea here is an investment theme has many epics underneath it and the epic itself has many user stories underneath it.

48:55  
Yeah, they they are used.

48:59  
I personally don't find it too useful when dealing with customers because it they feel kind of artificial to the customer.

49:06  
But yeah, it's a it's a one to many relationship.

49:14  
Yeah.

49:26  
Right, yes, yes, good, good question.

49:28  
Often what you do is you will collect things together, right?

49:34  
So imagine you're talking to a customer and you might start by writing down, yeah, you might start with the user stories, or you might start start with user requirements and you write them down.

49:45  
And now these might be high level user stories, so they're actually too hard to build within a Sprint, but you write them down anyway because you're trying to talk to the customer.

49:54  
And then you probably then if you're going to use epics, you'd write down the epics and you'd order them and then you'd start to think about these.

50:03  
If they're too hard, then you'd split them up and you'd make this clear to the customer.

50:08  
And so you discuss this.

50:10  
What I've done myself commercially is I've found that the epics for a lot of customers who are not familiar with Agile software development, they aren't too useful.

50:18  
So I've actually started with high level users requirements.

50:23  
We greet those with the customer and then from there produce user stories.

50:27  
And having user stories, I can then carry on with the discussion with the client about what we're building.

50:33  
But yeah, you don't want to confine yourself when you're talking to the users, the stakeholders just capture the information.

50:39  
You can then break it up into smaller pieces, order it, come back to them with your understanding, and you essentially go around the loop again.

50:52  
Yes, that's right.

50:53  
You probably give this back, this, these things back to them and you'd echo it back to them to say, have I understood you right?

51:00  
Is this, does this epic sit under this investment theme?

51:04  
Is that you know what you're interested in?

51:07  
So yeah, you, you try not to assume things.

51:11  
So you listen to people, you write things down, you ask them if that's what they want.

51:15  
You create a document, you show them the document, you ask them for comments and you go around the loop a couple of times to make sure you have collected them all.

51:24  
Now instead of physical documents or electronic documents, you can use requirements capture tools where the it's essentially a web service and you can give the web page to the client.

51:37  
You can say here, they all are in this web service.

51:40  
Have a look at them.

51:42  
Mostly because of security and various other problems.

51:46  
I haven't done that.

51:46  
So mostly although they're in an online system, we export them into a document.

51:51  
We give the document like it could be APDF or whatever it is, Excel or whatever the clients wanted.

51:57  
We give it to the client, right?

51:59  
Because often allowing them to directly access something the developers use is not good security wise.

52:06  
But nevertheless, they're getting in their list.

52:09  
You could be Userstory 22, whatever, and that's the number that's in our other system.

52:14  
So we have a mapping, yes.

52:24  
So user requirement is something that was originally used within waterfall model.

52:32  
You can use it as a concept to prime your understanding and then carry on with user stories, which is what I've done commercially that's allowed you, you must think to yourself, software engineer, sorry, software engineering is a set of ideas and approaches.

52:51  
So if, if you're gonna, if one particular set works better with a particular customer, change your approach.

52:57  
You, you tell people upfront, normally we're going to capture user requirements, then user stories, then we're going to order them.

53:04  
So you tell the customer what the approach is.

53:07  
You might have an agreement with the customer that your first version is going to be the V life cycle, after which you're then going to carry on with Agile.

53:15  
It depends on the project as to what you agree, but you agree the life cycle and then you match that.

53:23  
And like I say, you don't have to be exactly the same as a textbook approach.

53:27  
As I said, with these user requirements, textbooks don't require you to put in the user.

53:34  
This tendency has come from actual practical use.

53:41  
So where are they?

53:42  
Yeah.

53:44  
So practically, it's really useful to have the user in the sentence.

53:48  
And so a lot of people who've done engineering and software engineering in commercial sector keep the user in there.

53:54  
It's not in the textbook, but this is, you know, how does it actually work?

53:58  
It's like learning how to be a brick layer.

54:00  
Actually laying bricks may not be the same.

54:04  
Anybody else?

54:06  
Yeah, go for it.

54:11  
Yes, Right.

54:14  
OK, I'll mention what I mean again.

54:16  
Which slide is that?

54:20  
Yeah, yeah.

54:21  
All right.

54:22  
So what do we mean here by vertical slice of functionality is that if somebody wants a feature and then that feature probably touches several layers of the application.

54:34  
So you could have a user interface layer at the top.

54:38  
Underneath that there might be something that's calculating something or collecting data from somewhere that needs to reply to the user, sorry, reply to the user interface.

54:48  
There might be a web service below that.

54:50  
So by vertical functionality we mean the top layer, which is where the user plugs in, and the bottom layer is essentially where the data is stored.

55:00  
Yeah.

55:01  
So you may have to touch all the different layers between the user input and the bottom to fulfil the user's story.

55:08  
Now it could be that you only touch the top 2 because you left a feature in the middle layer, but you don't want the software to be sort of partially broken.

55:17  
The purpose of saying it may be a vertical slice is that you, you have to think to yourself, right?

55:25  
My architecture needs to allow me to add a feature in the user user interface, the sort of calculation algorithm layer and the database and then deliver that as an incremental update.

55:37  
If we build software so that we have to build the entire bottom layer first, then the problem with that is the customer doesn't see anything, so they can't give us any feedback.

55:49  
So we need, if possible, to have an end feature they can see connected to something underneath.

55:57  
Now, when we're going from absolutely nothing, it might be that the top level feature is very simple, like some data appears on the screen, but that shows that our low level features of reading and writing to the database now works.

56:11  
Nevertheless, it's still a vertical slice.

56:13  
Yeah, All right.

56:18  
Anybody else?

56:23  
Yeah, yeah, yeah, I can talk about that.

56:35  
Does anybody have any questions about the rest of the lecture or anything else everybody wants to stop?

56:42  
OK, the container stand scanning idea.

56:48  
So containers are often used to deploy software to the cloud and what they are is essentially layers of software.

57:00  
So at the bottom layer you have the basics parts of an operating system.

57:03  
It's not a proper operating system, but you've got the basic parts and then on top of that you may have other libraries or things people added.

57:11  
And then the top layer has got your actual programming.

57:14  
Now, when you put these services onto the cloud or somewhere else, there could be a problem in one of those layers because you started with somebody else's container and then you added some layers on top.

57:28  
So you need to pay attention to what's in the other layers below, which could have a vulnerability in like it could be a particular version of Linux has got an error or problem in it which affects your final product or not.

57:42  
Now there are pieces of software that will do this commercially, so I think Jfrog X-ray will do it for you.

57:51  
But anyway, you can unpack containers, meaning the layers can actually be taken apart and you can look inside them.

57:59  
And then you can combine that with essentially a list of issues which can be collected from the Internet.

58:06  
There are various places where they publish issues to do with software.

58:10  
And then you can use that to scan the container.

58:14  
And then graphically you could flag this as being serious or not.

58:18  
So it's potentially interesting as a project because containers are used a lot for software development and deployment.

58:27  
It's also interesting because they are a problem in terms of security and yes, we do have to worry about what's stored in them.

58:35  
I have got some supporting material in that I can show you how to unpack a container on all of its layers.

58:41  
And I've had previous experience with searching for issues in software.

58:46  
So it's essentially collecting together those pieces and building a tool.

58:52  
With any of these project ideas, you can put more or less into a particular place.

58:59  
So you could say, oh, I'm scanning it, I'm scanning a container.

59:02  
But now I'm going to focus on the user interface features.

59:05  
I want to be able to see this in a such a way.

59:08  
So you end up putting more work into the user interface, but you fulfilled the brief.

59:11  
Or you could say, oh, I really, really, really want to be very focused on the scanning.

59:16  
So I'm going to scan this version of DB and this other version I'm going to scan for all these different vulnerabilities.

59:23  
So when we come to look at a project and we market, we're looking at the overall amount of software that you've made.

59:31  
Yeah.

59:32  
So if you end up with more in the user's face, we're looking at the total.

59:37  
As long as you fulfil the brief, it's fine.

59:40  
It's the same with all the project ideas.

59:42  
There's a kind of a lot of leeway to put moral work in different parts of the project.

59:49  
OK.

59:53  
It's Jfrog X-ray.

59:54  
Yeah, it's what it's a commercial package for scanning for software vulnerabilities with any of these project topics that I'm supervising.

1:00:02  
Anyway, I normally provide a bunch of resources, so you don't have to start from zero.

1:00:08  
I'll give you some hints or places to read.