**w01-01-SoftwareEngineering-2024**

0:00  
OK O software engineering General introduction to the module.

0:09  
The purpose of this module is to introduce to software engineering.

0:13  
There are lots of textbooks that you can read.

0:16  
They tell you part of the story.

0:17  
The other part of the story is really from experience and a lot of textbooks give you ideas which don't work or maybe they work with a bit of modification.

0:27  
So anyway, what we're going to do in this is we're going to discuss the concepts and they will be somewhat textbook based and practical applications.

0:36  
Some of that will be in the lab.

0:38  
So the lab session hopefully is a lot of fun.

0:41  
We will be dealing with some realistic problems where we're trying to solve those problems and by doing so learn something.

0:48  
We're going to be describing and delivering some sort of project output.

0:53  
That's the idea of software engineering anyway.

0:57  
The other point I've already mentioned is the real life issues that we could encounter.

1:01  
So what do the problems look like that you actually see when you're working in the real world?

1:07  
And part of this we will see from our guest lecturers.

1:12  
Now there will be six of these, hopefully, and the first one is on Thursday this week.

1:17  
So they're all people who are currently working in a company and you'll be able to see their role, their background.

1:25  
And what will happen is during that session, we'll ask them a bunch of questions, try to find out more about what they're actually doing and how software engineering is working in their projects.

1:36  
And also you get to meet them, which is helpful because you might find a job which could be something you need after you've finished your MSC.

1:45  
In terms of the assessment for this, it's 40% group project, 60% exam.

1:51  
It's like this because the group project is fairly realistic to what you might see in the real world where you are working in a group and being able to do that is actually quite important.

2:03  
The exam is just a test your understanding of some of the ideas and the concepts that are discussed or introduced either in the lecture material or the labs.

2:12  
So there's a bit of multiple choice and a bit of short answer written questions.

2:19  
Right.

2:19  
So that is the general introduction.

2:21  
Does anybody have any questions or comments before I dive into the content for today?

2:28  
No.

2:29  
OK, let's go.

2:30  
Yeah.

2:33  
You mean the for 995?

2:35  
Yes, SO995 has been fully marked.

2:37  
It has also been moderated.

2:39  
There hasn't been the exam board yet, so I'm waiting for the exam board before I release them.

2:45  
The exam board is this month, so they will be released as soon as that exam board takes place.

2:52  
That's normal process because the exam board is like the last check of the marking, right, to make sure everything is fair.

2:59  
So don't worry about that.

3:00  
We'll just carry on and they'll be released as soon as I can, essentially.

3:08  
Yeah.

3:11  
Yeah.

3:11  
OK.

3:12  
So if you want to specify you as a group, want to specify to me another time, that would be very helpful.

3:19  
So what's happened is the person who sets up the projects has just told me here, put a time.

3:25  
I'm like, OK, but he hasn't looked at your timetable, which is rather silly.

3:30  
So if between you want to suggest a time, that would be great.

3:34  
I'll just put another time in my timetable.

3:36  
OK, great.

3:38  
Anything else?

3:40  
We'll sort that out afterwards.

3:43  
Right.

3:43  
Good software engineering.

3:48  
So where did this start?

3:50  
One of the key points in history where software engineering was used was around the Apollo missions to the moon.

3:59  
Now there was a problem here, which is that they had to write a computer programme in a very small system.

4:06  
This is the interface to the system.

4:09  
It's the desk key user interface.

4:12  
And this was on the lunar Lander.

4:16  
And so they had a very small amount of memory and they had to put all their software into this small computer.

4:23  
Now, if you can imagine, it wasn't written in the way that you'd write to a disc.

4:27  
It was actually manually wound onto the memory.

4:32  
Now, the problem they faced was money wasn't an issue.

4:36  
That wasn't a problem.

4:37  
They had to get to the moon.

4:39  
Limited computational power was a big issue and you'll see this today sometimes when you're dealing with an embedded system where you've got a smaller processing footprint.

4:50  
They had a very small one.

4:52  
They also had limited memory, that is memory to work in and limited storage for their programme to stored in and they had limited software development time and they needed it to work.

5:05  
Now if it didn't work, the astronauts would die potentially.

5:09  
So it's a safety critical system.

5:11  
And so out of this scenario, before it, we had the Wild West of software engineering where a bunch of people had their own keyboards and they were bashing away their code, throw it together.

5:22  
But during this, what came out of it was software engineering, where people had very carefully captured what needed to be implemented and then implemented it and tested it and made sure it was absolutely working as normal.

5:38  
Without that it would probably have failed.

5:42  
So you can look up on the Internet or read around what happened at that time, but it was very important that they started to work in a structured way.

5:51  
Now it's evolved since this time and there are many ideas that have become fashionable or less fashionable anyway, the way that you normally work, OK, you're not normally in an Apollo Guidance system team, sadly.

6:07  
You normally work in one of these 4 configurations that I've got here on the slide.

6:12  
So on the left hand side, you might be in a software team that's delivering software internally to your own company.

6:20  
So that could be, for example, if you're working at JP Morgan, somebody internal will ask you for something, you're building it for them.

6:27  
And then the product owner, that's the person who is following the features.

6:31  
They sit inside your company.

6:34  
It might be that you are building some software.

6:37  
Again, the product owner is inside your company, they are collecting the features that need to be implemented, but the customer is outside.

6:45  
You're selling software to somebody else and you have to really talk to this person, the customer, make sure they're happy, make sure they understand what it is they are being delivered.

6:59  
You may have this set U where the product owner is actually inside the company that you're selling into.

7:05  
So this company here hopefully has got some intelligence.

7:10  
I'd say hopefully, I've dealt with a lot of companies which don't have intelligence and they think they do.

7:15  
And then things go badly wrong because the development team over here hasn't hasn't got this sort of stubbornness or the wilfulness to tell them they have a bad idea.

7:25  
So you've got development team here, customer, product owner.

7:29  
And then on the right hand side, this is the most complicated system which I have seen as well and can go badly wrong where the customer is somebody else in between, you have a middle company and then you are in the development company, one step away.

7:46  
Now what can go wrong here is you are doing a fantastic job, right?

7:50  
You know your stuff.

7:52  
However, the person in the middle is a nompty to say the least.

7:56  
And they do not know their stuff, but they keep saying yes to the customer.

8:00  
And So what happens is that you're trying to tell this person in the middle help, help.

8:04  
Don't do that.

8:05  
We can't build that.

8:07  
Oh no, yes, yes.

8:08  
We, we want the customer to be happy.

8:11  
And then the whole project goes wrong and guess what?

8:14  
You're blamed.

8:16  
Now the way out of this is that you have to basically try to politely reach around the middle layer and and pull this customer into kind of three-way meetings.

8:25  
Or you tell the person in the middle, no, we're stopping and you walk away before it all goes badly wrong.

8:32  
Nevertheless, this is the hardest scenario to deal with, but it is possible to deal with it.

8:39  
OK, now I'm showing you that previous slide so that you start to realise the kinds of problems you're dealing with interfacing with whoever it is who's going to use your software.

8:52  
Now, normally with software engineering, we call these people stakeholders.

8:56  
These are people who care about what you're building, so it could be the end users, it could be the company, it can be all sorts of people.

9:04  
For example, the IT admin who manage the environment where your software is deployed are a stakeholder, so you have to make sure they're happy too.

9:14  
I did see one expensive project where the project was completely built, but they didn't bother to test.

9:21  
It worked on the client systems and so when it came to deployment, they couldn't deploy it because they hadn't considered the IT sys admin stakeholder.

9:31  
Now at the end of the day, you need to be paid.

9:35  
If it's an internal project, you may get told off.

9:38  
You may, you know, get in trouble.

9:40  
If it's an external project, if it goes badly wrong, you may not be paid properly.

9:45  
You may lose significant amount of money as we'll come to later on in this lecture.

9:50  
So you need some sort of way of agreeing what it is.

9:53  
You're going to build features that you are going to deliver.

9:59  
You also want to potentially deliver many times.

10:03  
It's not just here's your software, Thanks very much.

10:07  
I'm off now.

10:08  
You normally deliver and then you support the software for some period of time.

10:12  
Companies often don't want software that isn't supported now.

10:16  
It's better to release often.

10:20  
If you can release often and that will allow you to incorporate new features, fix bugs, you may not be able to do that.

10:29  
You may be stuck with releasing less often.

10:31  
That could be because it's some kind of safety critical system, but that's potentially more expensive and a different scenario that you're dealing with.

10:41  
OK, so teams and risks.

10:45  
Now, there are many ways of working.

10:48  
You can be like Linus who wrote the Linux kernel.

10:52  
You can go away and do most of it yourself.

10:55  
Now the benefit of that is you know what you're doing.

10:59  
However, if you're not careful, somebody else doesn't understand you, doesn't understand your code, and your code isn't useful anyway.

11:08  
So more likely than not, you have a lead developer and then you have say one or two, three, maybe 5 developers in a team.

11:17  
You normally have a project manager.

11:19  
This could be the scrum master as well.

11:22  
The project manager is sometimes a separate role.

11:25  
Their role is to interface with the stakeholders.

11:30  
They're the one who really bears the brunt of any customer relations issues.

11:36  
You often have one or many test engineers.

11:40  
Now these people are skilled in testing.

11:44  
They could be a developer who's not currently developing on your project, so they have experience of developing, or they could just be a professional test engineer.

11:53  
Now, normally these people have got the sort of inquisitive mind of thinking, I wonder if this works.

12:01  
Oh, it breaks.

12:02  
Great.

12:04  
They are normally given a test framework which is developed by the developers and they may be invited to think of other tests or give feedback or read the user guide.

12:15  
Essentially their job is to make sure it's built and runs properly.

12:20  
You may have some sort of release platform, so once your software has been fully built and then it's deployed onto the actual system.

12:29  
Imagine if you're running on an Apple Watch, the final Test is going to have to be done on an Apple Watch.

12:35  
It's no good testing it on a PC emulator, right?

12:41  
So we're now going to discuss how information flows through a project or how it doesn't flow.

12:50  
Now we need to describe what we're building, and this is very important.

12:54  
Don't describe things in such a way that somebody else doesn't know what you're talking about.

13:00  
You may think, oh, that's obvious, will.

13:03  
But if you use acronyms or words which are, should we say more computer science, then the stakeholder who may have no experience of computing won't know what you mean.

13:15  
You want a short sentences that describe the features.

13:19  
That's really crucial.

13:21  
You need to be able to also potentially describe what is inside your software, the internal structure.

13:28  
You need to be able to describe how the bits fit together.

13:30  
Software is often not just one programme, it's often many programmes stuck together.

13:36  
You have libraries you interface with or or other pieces of software that you've already deployed.

13:44  
You may need to describe the logic patterns that are inside your software.

13:47  
So OK, I'm going to press these three buttons and then that happens.

13:51  
Or maybe there's some state change.

13:55  
We will need to track process, sorry, progress and the issues.

14:00  
What do we mean here?

14:02  
We want to track how our software is being built.

14:05  
Like, OK, so we've built that bit now.

14:07  
Yeah, we're OK.

14:08  
We can move on to the next bit.

14:10  
If we don't have progress tracking, we won't know when we're finished.

14:14  
We also want to capture issues.

14:16  
Now these are normally bugs or software defects.

14:20  
That's the way of referring to them.

14:21  
But they're basically issues.

14:23  
So something you have to fix.

14:25  
And also you need to describe when it is finished.

14:29  
And you may say, well that's obvious, isn't it?

14:31  
It's when it works.

14:33  
But the problem you have is that your version of when it works isn't what the customer thinks of when it works, and so you need an agreement of when it's finished.

14:42  
And this is very important because otherwise you won't be paid, and that's bad.

14:49  
So here are the risks.

14:51  
If you are the 10X developer who's very fast and furious with the keyboard and you can't explain what you're doing to another person, then your knowledge is lost when the when you leave or when the 10X developer leaves.

15:07  
And sadly, your code probably goes in the bin.

15:10  
I've had to deal with projects where people have given me their code.

15:14  
It's worked, but it's been horrible inside and it took a long time to fix.

15:19  
In some cases it was faster just to throw it away and write something else.

15:23  
So you're not popular.

15:25  
If you do this commercially, it doesn't work.

15:28  
You may be very clever, but in a real scenario you're only as good as the team.

15:35  
So if you know you're the star football player, you have to play in the team.

15:40  
Code with no comments is not a good idea.

15:42  
It was a sort of fashion that people came up with because when you update code, you have to update the comments.

15:49  
So if you make too much effort with comments, then you waste too much time on them when you're updating the code.

15:56  
That is a bad thing, but zero comments is not great.

16:00  
I can remember a project where I was given, I think it was 495 Fortran files.

16:05  
There were no comments and there were variables called sine Theta that held the value of kolst Theta.

16:11  
And I only knew that when I had actually done the maths through the entire code base and worked out what was in that variable.

16:18  
That was an excessive waste of time.

16:20  
Don't do that.

16:21  
Don't have variables like AAB or CCD or something like that.

16:26  
It should be obvious to people bad architecture design.

16:31  
If you set off and you're trying to build a very big programme and you don't know what it's going to look like before you start building it, it could well go wrong.

16:40  
The analogy is building a house.

16:43  
If you throw bricks into the ground and hope for the best, you and your friend who's building the other side of the house when you get to the top might not be right for the roof to go on because your design isn't there at all.

16:57  
I have seen the scenario where the customer was warned that the architecture that was being selected was wrong.

17:03  
Sadly, nobody listened.

17:05  
It was that three layer scenario.

17:08  
And yes, in the end this software didn't work and it was fundamentally flawed because of the architecture.

17:13  
Completely avoidable.

17:16  
Bad interface design.

17:17  
This is a problem where you have several developers, several teams of developers potentially, and the connection point between bits of software is called your application programming interface or API.

17:30  
Now if you haven't assigned this properly, you will potentially need more features and then the connection point becomes a real problem.

17:39  
It becomes a bottleneck.

17:41  
You do not want to change the interface very much if you can, if at all, because if you do, you break stuff that relies on it.

17:49  
So things like the Linux kernel are very careful that they don't break the user interface if at all possible.

18:00  
Bad issue tracking.

18:01  
The problem here is if you don't track them, you don't know when they're fixed or you don't know when they've broken again.

18:07  
And being able to test whether the issue's gone away again is normally associated with some kind of automated test, unit test.

18:17  
If you have no definition of completion, what happens?

18:20  
You continue to develop the software and then the customer doesn't accept it because as far as they're concerned, it's not finished because it doesn't have the feature in which you didn't decide on, or they didn't decide on, or it was in their head but you didn't write it down.

18:36  
You can get the idea.

18:37  
And then you don't have the money coming to you when you kick out the invoice.

18:43  
OK.

18:43  
Does anybody have any comments or questions about what I've discussed so far?

18:47  
Before I move on to development approaches, yes, yes, we are going to come to that.

18:58  
It's a project management role with inside the Agile life cycle, which is later on in the slides, yes.

19:18  
Yes, yes, I do know what you mean.

19:24  
In the best case scenario, the the project manager is a former developer who has become more involved in project management.

19:33  
Occasionally they are a specialist project manager, but then if they're a good one, they take interest in the software inside.

19:41  
It's not a good idea when you have a manager has who has no idea of what they're talking about with software, because then they will just beat the development team harder and harder.

19:53  
Over time they might learn, but they might lose their developers.

19:56  
So it depends on the company.

19:57  
You can ask who, who are they?

19:59  
What are they doing?

19:59  
But more often than not, the better ones have been developers.

20:03  
But they feel like they they like that.

20:06  
And as you see, when we do throughout the module, we'll discuss specialist roles like test engineer.

20:12  
So it might be that you like software development, but you want to do a bit more testing and you could sort of wander around, right.

20:17  
So you could be a project manager for a while.

20:19  
And they say, well, I've had enough of that just now.

20:22  
I want to go back to being a developer.

20:24  
So some companies will allow you to do that.

20:27  
Anybody else?

20:28  
Yeah.

20:40  
Yeah, good question.

20:41  
It depends on the company and the setup.

20:44  
You can have the lead developer and the project manager being the same person or it could be a separate person.

20:49  
I've seen both.

20:51  
If it's a bigger project, it's probably going to be a lead developer who's separate from the project manager.

20:59  
The lead developer may end up, as we've mentioned, the scrum master could be the scrum master too.

21:05  
So you the project manager's role is very specific in the sense of they have to talk to the clients, right?

21:13  
They're the point of connection with the client.

21:15  
And so some clients can require more talking to more work.

21:21  
And so that project manager role becomes more consuming and so you've got less time for developing.

21:27  
So that's where, as we said earlier, they could be just a project manager, as in that's their complete role, but they were formerly a software developer.

21:36  
Yeah, OK, right.

21:39  
Development approaches.

21:40  
Now, you can build software in different ways, and it depends on what you're doing as to how you approach the problem.

21:47  
You can start off by trying to understand what it is you're building from the hardware up.

21:55  
Now, when might this happen?

21:57  
This might happen when you've got a particular device you're going to deploy on.

22:01  
For example, imagine you've got an Apple Watch.

22:03  
You need to better interface to the sensors on the Apple Watch.

22:07  
That is a given.

22:08  
If you don't have that, you can't use them.

22:11  
So from this point we can understand the hardware.

22:14  
We can write little bits of programme where we're connecting to that hardware functionality.

22:21  
You can also understand the software development framework.

22:24  
If you're a developer that's not very used to that framework, you might just want to go and build a simple application and test it out.

22:31  
Now, why do we do this sort of thing?

22:33  
We're basically doing this so that we feel very confident we know what we're doing.

22:38  
And this is developer LED.

22:40  
So it could be that the company is giving you some time to build the infrastructure.

22:46  
You haven't promised anything to the client yet.

22:49  
You're doing this in house, so it's somewhat hidden from the client.

22:53  
If you're in a bigger company and they build it, been building with this framework for a while, probably somebody else has done the work to understand the framework, like maybeits.net or something.

23:04  
They have built up an infrastructure and you're working on top of that and you just have to talk to them about it.

23:09  
Occasionally you start from the beginning.

23:11  
You may have to test out some core algorithms.

23:14  
So for example, you might be dealing with some sort of allocation algorithm using, I don't know, reverse Vickery auction or something that you could find in a paper.

23:25  
And you've never seen that implemented before.

23:27  
So you're going to go ahead and put that into a bit of software test.

23:31  
Does it actually work?

23:32  
You may care about memory, CPU, disc, network performance as well.

23:38  
Why would you do this?

23:39  
Well, you do it because if you look at the performance at the small scale, you can guess how the hardware is going to behave at the big scale.

23:48  
Now this becomes a real problem if you've got a limited hardware footprint.

23:53  
So for example, if you have an embedded system, if you buy the computer that goes in the telly and the computer that goes in the telly is not big enough in terms of the processing power to renew, run your full scale programme, you are in trouble.

24:07  
So you're better off doing some testing on the small scale with some hardware and trying to project how much resources you will need.

24:19  
OK, you can do so-called user LED development or prototyping development where you are talking to the user, you're showing them things.

24:29  
Now you have to be careful with this because you are implicitly committing yourself to building it.

24:35  
You can show them, oh look, would you like this?

24:37  
Would you like this?

24:38  
Yes, yes, yes.

24:38  
We'd love that there is an understanding building in the clients head that you're then going to build that O.

24:45  
You need to be careful with what you show them, but nevertheless it's very useful to show them the user interface.

24:54  
So if you can build 1, even if the buttons don't work, you can show them how it's going to potentially work, talk them through how you think it should be working, and you can collect other features.

25:05  
What sometimes happens as well is that software is deployed, but it is not complete.

25:11  
And so then you can test it in the workplace and get some more feedback, which is a sort of enhanced user LED prototyping if you wish.

25:22  
All right.

25:23  
So once you've done this exploration phase, if you want from the ground up or if you're working with the user you're coming down, you then are going to think about all the other bits and pieces, right?

25:36  
So following the hardware selection, we may have to connect to the Apple Watch or maybe if it's a mobile phone or some other device.

25:45  
The thing in the telly we need to get our inputs and outputs.

25:48  
This is all low level programming, so it's the building up from the bottom.

25:52  
We may need to connect to a bus.

25:54  
Now a bus is just a way of sending data.

25:59  
So on computers you have lots of buses.

26:01  
You'll notice we have a USB port on laptops.

26:04  
Normally that's a bus you've got on the computer processor.

26:08  
You have an SPI bus, an I ^2 C bus.

26:11  
You may have a CAN bus to talk to electronics somewhere else.

26:14  
You may have GPI B bus that's an old style bus.

26:18  
Anyway, buses are a specific connection that you use to talk to something else, and there's a way of sending data depending on what you're connected to.

26:28  
So you need a driver.

26:31  
Then from this you may develop some kind of data structures.

26:34  
You need some data that are going to be sent to the device.

26:38  
For example, you're sending stuff to the screen.

26:40  
You may encapsulate those data in such a format that you then give to a screen driver, which then puts it on the screen building down.

26:51  
OK.

26:52  
Once you've understood that low level hardware, you get what the input should be from the user.

26:57  
They tell you how they think the thing should work, and then you build down from the outside into that middle part.

27:04  
And then you're going to fill the middle with the features which are coming from your stakeholder engagements.

27:11  
So it's whatever they want that you're trying to capture.

27:14  
Now, you as a developer don't necessarily give them exactly what they want.

27:19  
You need to think, is what they want actually what they want, or is it something else they need?

27:26  
Do they understand the cost of what they're asking for?

27:30  
So you need some input to haggle or discuss between you, The budget, the scope.

27:38  
If they have exactly what they want and it costs them a vast amount of money, they're not going to be happy.

27:44  
Sounds stupid, but they won't be, right?

27:48  
Let's move on.

27:49  
So development life cycles.

27:56  
Now we're going to discuss in the following lectures the steps of the Standard Life cycle approach.

28:03  
But here we're just introducing them.

28:05  
The normal steps are we start by defining the problem, what is it the customer wants?

28:11  
And for this we often use user requirements or user stories as you'll see them with Agile.

28:16  
We want to decide what the interface is.

28:19  
Now the interface could be some buttons on a screen, it could be on a computer screen, it could be on a watch, it could be a network interface.

28:32  
There are many different user interfaces.

28:34  
It could be a command line interface.

28:36  
Anyway, we have to decide what it is.

28:40  
We also need to decide what the high level design is.

28:42  
So how do the big pieces fit together?

28:45  
We want to decide how the smaller bits inside the components work together.

28:49  
Then we go ahead and build the programme.

28:51  
So we implement the thing, we test it and we check all the requirements, all there are they fulfilled and then we deploy and operate the programme.

29:00  
So there is sequential steps.

29:02  
Now what we're going to do in the following lectures is we're going to discuss this as requirements definition.

29:08  
So this is a broad term, software and system design, which is the next layer down, implementation, implementation and unit testing, which is another layer down, integration, system testing, and then finally operation and maintenance.

29:21  
So these are different stages in the process.

29:25  
All right.

29:26  
Now, if we've got features, if we've agreed what we want to build, we need to be able to trace those features to what was actually built.

29:37  
Imagine if you're building a house, you've got a plan.

29:40  
You go, where's my window?

29:42  
Oh, windows not there.

29:44  
You're going to look between your plan and what is built.

29:47  
In the same way with software, we want to capture the feature.

29:52  
We want to say who needs it and why do they need it?

29:56  
Now, that's important because a feature that's requested by some junior member of staff and it's like, Oh yeah, we might want that.

30:05  
Could be something that you put aside and you don't build it initially, maybe it's very much more important later.

30:11  
So you need to capture who needs it, why they need it, and you also need to decide how it's going to be tested.

30:19  
So when will this work?

30:20  
What will it look like?

30:23  
You need to have the list of features that are in your software release.

30:27  
This is version 1.01, point five or two, whatever, and these are the list of features.

30:33  
You often see that with commercial software.

30:35  
Now there is a problem with software and this potentially is an issue when you're dealing with safety critical systems.

30:44  
So for example, if you're building software that goes in an aeroplane or a civil nuclear reactor, then you must be very careful with zero day exploits.

30:55  
So what I mean here is you've got some software that's in your code.

30:59  
It's not yours, it's in a library.

31:02  
You didn't know it was there.

31:03  
You were just using the library.

31:05  
But in that library there's a problem.

31:07  
You've never tested that bit of code, but it goes in the aeroplane and then something strange happens and that piece of code is executed and the plane drops out of the sky, or something bad happens, or the nuclear actor melts down.

31:20  
So you need to have a reverse mapping when you're dealing with safety critical systems, where you look at the features that have been built and you go back to what you asked for.

31:32  
And essentially you're saying, did I ask for that?

31:34  
No, get rid of it.

31:36  
But did I ask for that?

31:36  
No, get rid of it.

31:37  
Now, this is potentially a problem with these safety critical systems.

31:42  
So for example, nuclear reactors, they, the more modern ones, use what's called PLCS, programmable logic controllers.

31:51  
And in there, yeah, there could be libraries and you have to be very, very careful because yes, potentially you could exploit them.

31:59  
Here is the first software lifecycle that, well, really was established as being a software lifecycle, which is the waterfall.

32:07  
Now you can see why it's called waterfall.

32:10  
We have at the top a box and we're flowing down to the bottom.

32:16  
And the idea here is we start off by capturing the requirements, understanding what does the customer want.

32:22  
We then write it up as a design.

32:24  
So we go away and do the thinking, we then do the implementation, we then do the verification and finally we give it to the customer.

32:31  
Now this is great by in fixing things because you sit down carefully with a customer and you spent ages and ages and ages fixing this and then finally when you fixed it, you go on to the next thing.

32:44  
Now, most of our descriptions of life cycles are actually used in engineering as well as software.

32:50  
So they have used this sort of approach for fighter jets and that's why they started taking a long time to build, because by the time you're down here you can't change things at the top.

33:03  
It's a, it's fairly strict in the sense of you're locking yourself out of making changes.

33:11  
Now, what are we saying?

33:14  
Requirements are fixed so you agree them and then that's it.

33:18  
That's what you're building.

33:19  
Potentially there's a long gap between the requirements capture and the delivery.

33:24  
That could be a problem because during that long gap you've learned something else, but it's too late.

33:30  
You're locked in.

33:31  
You're just going to have to deliver.

33:33  
So when you come to delivery, the customer might not be happy.

33:37  
The stakeholders can't influence the development very easily.

33:40  
Imagine you have a meeting, you show them the version, you go, oh, there, there it is.

33:44  
And they say, oh, no, no, no, we didn't want it there.

33:47  
I know we said it in the design, but we, we actually want the buttons over here.

33:50  
And you've missed that feature.

33:52  
Oh no, we can't do that.

33:52  
Now we're on the implementation phase.

33:55  
So what happens is if you strictly follow it, typically the customers are unhappy.

34:00  
As a project manager it's great because you have constrained everything so you know how much it's going to cost you and so you've reduced the risk to use commercial entity.

34:11  
So what it looks like graphically is shown on this slide with the so-called iron triangle.

34:18  
So what we do here is we fix the requirements.

34:21  
We say right, that's what we're building and then somebody who's seen this before, they are OK, we've built a few houses before we know how much this is going to cost.

34:30  
It's got this, this number of bricks, whatever it is.

34:34  
They then go ahead and they can estimate the cost and the schedule.

34:40  
How long is it going to take to build, how much it's going to cost.

34:43  
Now, obviously these two things are related because the schedule will develop depend on the number of developers, the number of developers, the number of hours related to the cost.

34:54  
There are other versions of the Waterfall, so you may see the V life cycle written down and some projects still use it.

35:04  
It's somewhat like the Waterfall, but potentially you have a bit more flexibility.

35:10  
The idea here is you work down the V and then back up and at each stage you're verifying your previous design step.

35:19  
So you essentially you are verifying your detailed design or system integration.

35:24  
So you have interaction, formal interaction with the client to say that's signed off.

35:28  
Now, I have seen this used in some military and civil nuclear projects, but often it's not used.

35:38  
Now there is dwindling interest in this because it's also inflexible.

35:44  
It's good for constraining costs.

35:47  
But yeah, it's typically used for maybe the initial build where you've got nothing to start with, or maybe a smaller software project.

35:57  
Now there are many approaches.

35:58  
So what we're doing now is we're going from that waterfall and we're going through history and looking at different ideas people tried so that you can see there's a range of ideas you can use, and you don't have to use one of these ideas.

36:13  
You can use an idea and slightly change it.

36:16  
This one is called the spiral life cycle.

36:18  
Now what happens here is you start in the middle and you build a little prototype.

36:25  
So you're building, oh look, here it is.

36:27  
You show it to the customer, they go, OK, fair enough.

36:30  
You're a concept idea.

36:32  
You then go around and you think about it a bit more.

36:34  
You build another prototype and you go around, you capture some software requirements.

36:39  
You talked about it with the customer potentially you bit more new software design.

36:43  
And then when you're on the out outer loop here, you've essentially started to lock down things.

36:49  
You've built your unit tests, you integrate and then you your final product comes at the end.

36:55  
So it's an iterative, what do we call it, prototype LED design, where you are freezing things out as they become more stable.

37:06  
And this is still used a bit.

37:10  
So it's helpful because you're dealing with prototypes where you've got less risk.

37:15  
You can then use them to discuss, think about ideas, lock them down.

37:19  
As things become more rigid, it becomes more like the waterfall.

37:24  
Here's another instance.

37:25  
This one is a little bit later in time, 1991.

37:28  
The idea here is that you have several prototype cycles where essentially you build something, you show it to the customer, you build something, you show it to this customer, and you go around in a loop.

37:40  
And so, yeah, we think we've got it now.

37:43  
And then once you've done that, you go ahead and do your formal verification, final construction, whatever, and deploy.

37:50  
And that's quite useful.

37:52  
The problem with this is it's maybe not as formal, maybe not as easy to agree on when to stop.

37:59  
Anyway, you can see what's happened.

38:01  
People have gone from a very rigid way of thinking to become more fluid, more prototype lead.

38:09  
And then you end up with Agile, which is essentially another step down the can we build something that's a bit more flexible ladder.

38:19  
So the idea with agile, it came from the so-called Agile Manifesto and this was written down.

38:25  
You can find it using the link at the bottom.

38:27  
It was written down by a bunch of developers who are fed up with the status quo.

38:31  
They were fed up with people being unhappy, developers being unhappy, the customers being unhappy.

38:37  
So they came up with a bunch of ideas which are in the Agile Manifesto.

38:42  
Now, the Agile Manifesto has got lots of good things in it, but as you'll become aware during this module, potentially there are some other problems with Agile that you need to be careful with.

38:54  
So they really wanted to have more of a focus on individuals and interaction.

39:00  
So talking to people, actually discussing with the customer rather than processes and tools.

39:07  
They didn't want to get bogged down in vast amounts of paperwork which is associated with user requirements capture.

39:13  
They wanted to focus on building working software instead of writing enormous amounts of documentation.

39:20  
Now, if you are very strict with safety critical software in a waterfall approach, you end up writing lots of documentation before you build anything.

39:29  
I know because I had to do this for civil nuclear and it does take a lot of time.

39:35  
Customer collaboration, that is much better than you just sort of capturing things and leaving them alone.

39:44  
That is better than contract negotiation.

39:47  
Contract negotiation is where essentially you've built it and then you bring in the lawyers to argue that, yeah, it's finished.

39:54  
Nobody likes that you're wasting money on lawyers.

39:58  
Yeah, you may be paid in the end, but they hate you afterwards.

40:02  
It isn't actually what they wanted.

40:04  
So they wanted more customer collaboration.

40:07  
And then lastly, responding to change.

40:10  
It's very obvious when you've got a working bit of software what people actually want that you have missed.

40:17  
And so being able to respond to change different points in the development cycle, even when it's deployed is very important.

40:24  
Rather than following a plan, you follow a plan.

40:26  
You'll stay on cost, but potentially you won't deliver what the person wants.

40:32  
OK.

40:34  
So the goals of this make sure the customer's happy.

40:38  
So we end up with continuous delivery.

40:40  
We don't just give them version 1.0, they end up with version 1.0 dot whatever it is.

40:47  
So the software keeps rolling out.

40:49  
It could be every three months or or so.

40:52  
It depends on what it is as to how frequent it is.

40:55  
We are willing to accept late modifications, meaning somebody comes to you and say can you just get this in?

41:01  
Oh yeah, OK, we can put that in.

41:03  
We need working iterations of software.

41:06  
So we are developing the next version.

41:09  
It's released.

41:10  
We deliver frequently.

41:11  
Now that's going to have to mean you have to automate things like unit testing, which we will come back to later on in the module.

41:18  
We want the developers and the owners to talk to each other, to work together.

41:21  
We don't want to separate these people.

41:24  
This will motivate people.

41:26  
Face to face meetings are potentially a lot more efficient than e-mail.

41:30  
If you stare into somebody's eyes and look at them, you understand what it is they are feeling, rather than looking at an e-mail where you've got no idea.

41:40  
And yeah, you want to be able to produce iterations at a manageable pace.

41:46  
Now, this is the kind of tricky thing about Agile because if you don't manage it properly, you end up working too hard and you need a bit of experience to know what a manageable pace is.

41:57  
So compared.

41:58  
So I'm using the two extremes here.

42:00  
OK, on the left we have Waterfall, on the right we have Agile.

42:04  
So with Waterfall, what we did was we fixed the requirements and then we used our experience to say how much it's going to cost in the schedule.

42:12  
We estimate those.

42:13  
We might be wrong, but we try our best.

42:15  
With Agile, we basically say, OK, we're going to fix the amount of resources we're going to put in here so we know how big the development team is.

42:22  
We're going to be ploying to this.

42:24  
We fix the schedule, meaning we're going to deploy every month, every year or whatever it is, maybe every few weeks.

42:31  
And then given that we estimate what we can do in the time with Agile.

42:38  
There are many ways of doing a so-called agile life cycle.

42:43  
The one you'll see most, this comes from 2017, but it's still true.

42:48  
Agile Scrum is still the biggest chunk of the pie.

42:51  
It is one particular version of Agile.

42:54  
There are others here.

42:56  
You've got hybrid Scrum band, Scrum XP, K band, whatever else.

43:02  
Now, even with Scrum, you don't need to use all of it.

43:05  
It's written down.

43:06  
How does it work?

43:07  
It's kind of like an instruction manual.

43:09  
And you can actually be trained in Agile Scrum.

43:12  
You can have a certificate to say you're proficient, but you don't need to use all of it.

43:16  
And you might see in a particular company use parts of it, which is fine if it works.

43:22  
This is what agile scrum looks like within this normal illustration anyway.

43:28  
You have a team of developers.

43:31  
They are the the development team who are going to come together in a scrum.

43:36  
Now, a scrum, if you're a rugby player, you know what a scrum is.

43:39  
It's where you all huddle together, your shoulders interlocked.

43:43  
The idea here is that you are in close communication with each other.

43:47  
You're not sort of distant away.

43:49  
Oh yeah, I'll call you back later.

43:51  
No, you don't do that.

43:52  
You have a a stand up meeting normally and you talk to each other at the beginning of the day.

43:57  
What is it we're going to build today?

43:59  
OK, I'm going to build this.

44:00  
You're going to build that, right, Let's go on with it.

44:02  
1015 minutes, stand up meeting, then get on with it.

44:05  
So what happens is you have a product backlog.

44:08  
These things are features that you need to build for your product.

44:13  
So they're collected and they're stuck put in here and they're normally expressed with user stories.

44:17  
We will come back to this, don't worry.

44:19  
You then decide what you're going to do inside a Sprint.

44:22  
Now, Sprint is just a set, a set of development steps if you like.

44:27  
It's normally a few weeks, it could be two weeks, it could be 4 weeks depending on who you're working with.

44:33  
So you're going to do 2 to four weeks of development on what's in here.

44:37  
So at the beginning, you have a meeting with a customer, you decide what you're going to put in this Sprint backlog.

44:43  
Then the development team goes away and in 24 hour jumps, they come back, they talk to each other in the scrum and they keep doing this until the sprint's finished and then they release a working iteration and that's how it works.

44:59  
Now we will come back to this in more detail in a later lecture, Agile Techniques.

45:07  
This was a survey done in 2017.

45:11  
What can you see here?

45:12  
You can see that people use parts of the agile life cycle ideas.

45:17  
90% of companies use stand up meetings even if they're not using other agile approaches.

45:23  
I've seen this used just on its own and it's really handy just checkpointing where everybody is.

45:29  
The good thing about this is if you're relaxed, you're honest, it's very, very quick to find problems and help each other.

45:37  
I'm stuck with this hardware.

45:38  
Oh, great.

45:38  
OK, I'll come round to your desk later rather than I'm not sure I should talk to them.

45:45  
And it goes on for weeks.

45:46  
And then you've got a problem.

45:48  
Yeah.

45:49  
88% use Sprint planning, 85% retrospectives.

45:53  
Retrospective is where you come to this Sprint meeting and you look back for what you previously did.

45:59  
Anyway, the point of this graphic is that people use parts of the Agile Scrum approach now.

46:08  
The fun stuff, development failures, software is not what the client wanted.

46:16  
You might think, oh this is just never happens.

46:18  
I've seen it happen.

46:20  
Why does this happen?

46:21  
It happens when you haven't spent enough time talking to the client capturing what the requirements are that they really wanted.

46:29  
Software doesn't work on the client's system.

46:32  
Oh oh I have seen this.

46:35  
Somebody spent, I can't remember, I think it was £100,000, a 150,000, something like that on software that couldn't be run because the person who built it didn't test it on the client's infrastructure.

46:48  
Oops.

46:50  
So badly designed user acceptance tests.

46:52  
Remember, we were talking about earlier, If you're building stuff for the Apple Watch, you've got to lastly test on the Apple Watch.

46:58  
Yeah, you really must do that.

47:00  
You can't guess because if it doesn't work on the final system, it's useless.

47:04  
We'll actually talk about this as the last mile.

47:07  
So things are nearly finished, but unless they work on the actual final hardware, it's not finished.

47:15  
Software contains defects, only appear when used.

47:18  
Oh dear.

47:20  
So you've done all your testing and then somebody clever presses a few buttons and your system reboots and you think, oh, oh, that's bad, we don't want that.

47:28  
Now going back to the Apollo story, it was quite amusing that the one of the engineers was, I think at the time, she had a small child and the child would play with the user interface and the child had managed to press a few buttons on the user's interface and it then did something bad and they had to reboot it.

47:49  
And after this, she reported it to the development team and they all said no, no, no, that won't happen.

47:55  
And, and then as the Apollo was coming to land Apollo 11, it happened and they had to reboot the computer.

48:03  
So basically, you want to be careful with your testing.

48:08  
And if it's safety critical, you want to test crazy stuff that no, nobody will ever do this.

48:13  
And that's where the test engineer is really useful because they've got the one of these keen brains to think.

48:20  
I wonder what happens if I do this.

48:22  
It's not in the feature set.

48:24  
It's not something in the user manual, but I could try it, see what happens.

48:32  
Yeah.

48:32  
You may fail because of your software life cycle.

48:37  
You may lack focus on the development goals.

48:39  
What is it you're building?

48:42  
Agile could help.

48:44  
I say could because you can screw up with Agile as well.

48:48  
You can focus on working implements with Agile.

48:52  
What kind of what what can happen with Agile is the customer keeps asking for more, but you don't actually really deliver.

49:01  
Now if you're not careful, you've you've sort of delivered that feature and then it gets ticked off and then there's sort of another one being developed and it's ticked off too.

49:11  
And the clients, Oh yeah, can I have the final version now?

49:15  
But in fact, most of the features don't fully work.

49:18  
So avoid \*\*\*\*\*\*\*\*\*\*\*.

49:20  
You must must develop something that works, sign it off and carry on.

49:27  
Clients can be really tricky sometimes and you need to be honest with them.

49:32  
And that could be that.

49:33  
You just have to say we need to stop, we need to fix this.

49:37  
If you allow the problem to become bigger, it can get out of hand.

49:42  
Here's one that got out of hand.

49:44  
I can tell you about some more problems that came out of hand if you like.

49:49  
So what happened here was a big problem.

49:52  
There was a bit of software that was being developed for Police Scotland.

49:55  
You can find about the technical stuff if you follow the link at the bottom.

50:00  
So it looked good to start with, good approach.

50:03  
But the contractor, the person who's building the software, didn't fully understand the complexity.

50:09  
So they thought they had control over what it was they're building, but they didn't.

50:13  
So I have seen this before, also in a different project, wasn't Police Scotland.

50:17  
I wasn't involved in this one.

50:19  
But when you don't spend enough time to understand the complexity of what the customer wants, that is really nasty as it grows and grows because there's a potential misunderstanding.

50:32  
You assume one thing, the client assumes another one, and when that gap starts growing, that's when things go wrong.

50:39  
What happened here was they didn't trust each other after a while and then there were disputes about the project's scope so they couldn't agree on that.

50:48  
And at the end there was a 24,000,000 LB settlement where the company had to pay 11,000,000 back and then in the further 13 million bad.

51:00  
Don't do that.

51:01  
And then of course, Police Scotland was screwed up because they didn't have what they needed.

51:05  
Now what is very obvious from this sort of thing, and it comes up time and time again, is public clients tend to be rubbish.

51:15  
All right, So one of the big problems here is Police Scotland as an organisation is not very, should we say, skilled with software development.

51:26  
If you have people who know what they're doing on the customer side, they will ask you for something that's reasonable.

51:32  
But if it's a public sector organisation, they typically ask you for something that's unreasonable and you have to try to explain to them what it is they really want or what it is they're asking for and what it will cost.

51:46  
You may need to walk away.

51:48  
So generally speaking, I have seen most problems with public sector organisations.

51:52  
If it is a private sector organisation, they will typically have more intelligence about what it is they're wanting.

52:01  
This doesn't just happen with software, this happens with engineering.

52:04  
If you are aware of the boats, the two boats for Cal Mac that are being built, it's all the same stuff.

52:11  
The software problems, same problems, right?

52:17  
What happened with this one?

52:18  
There were critical error errors in the technical coding.

52:21  
You did also have problems inside here where it wasn't clear that things have actually been fixed.

52:28  
So a defect was resolved, but then it broke something else.

52:32  
So this tells you they didn't have a proper automated testing framework, which is bad because if you don't have that, you wrap it in development, you potentially break lots of other things you've previously tested and manual testing is expensive.

52:47  
There was a lack of compliance.

52:49  
This is also bad.

52:51  
So compliance is where your software needs to fulfil a certain set of standards, could be security standards or something else, data protection standards.

53:01  
There was an error in the search and audit modules and yeah, there was limited functionality in this administrative module, so it looked good to start with, but some of the process and approach that the company was using was wrong.

53:13  
I'd argue probably the public sector organisation typically speaking, doesn't know what they're doing either.

53:18  
And so you have this bad combination all ending up in a big mess.

53:23  
Nobody's happy, right?

53:26  
The end of this lecture.

53:29  
Software engineering is vital for success.

53:32  
If you just code away yourself in your own little cupboard, it might work.

53:39  
For example, Git as a piece of software, if you don't know we will be using it.

53:45  
Git was written, I think over a weekend by Linus.

53:47  
Again, he was fed up with the other version control systems, went in his garage wherever he's coding, came out at the end of the weekend with the software, and people still use it today.

53:58  
Great.

53:59  
That doesn't happen very often.

54:01  
So you do need tracking what it is you're building and what you're going to test.

54:08  
Mostly these days people use sort of agile in the way they approach things.

54:13  
I say sort of because there are many ways of using these life cycle approaches.

54:18  
You don't have to use absolutely agile Scrum.

54:21  
Sometimes people will say we require it to be built with this life cycle, but mostly people allow you flexibility about how you approach things.

54:29  
Scrum is still the most popular within the agile set of life cycles.

54:35  
If we don't follow a disciplined approach, we can end up with a very costly failure.

54:42  
I have seen this several times personally.

54:46  
It's very frustrating when you try very hard to avoid it, but it's a learning experience.

54:53  
Yeah.

54:53  
So ideally you need to so-called fail fast where you discover the problems early.

54:59  
If you can discover the problems early, potentially you can stop there and walk away.

55:03  
It could be that you've designed something.

55:05  
The customer thinks they know what they want.

55:07  
You stop there.

55:09  
You say, OK, we'll stop in there.

55:10  
Great.

55:11  
Here's the bill for that bit.

55:12  
You can ask somebody else to bill it.

55:14  
Have a nice day.

55:15  
All right, So you don't look bad if you have to exit at that point, better that than failing badly.

55:22  
Yeah.

55:22  
And it's also important to work with working increments that you are delivering to the customer.

55:29  
OK, so that is all I had to say this morning.

55:32  
Does anybody have any comments or questions?

55:36  
Yes.

55:44  
Yes.

55:45  
So I'm recording it.

55:46  
Yes.

55:48  
So I'm recording it basically.

55:49  
So you can go back and you can listen to it if you want.

55:52  
And yeah, if you're sick or something there, you can always catch up.

55:57  
Anybody else?

56:02  
All right.

56:04  
OK well, we can stop early if you like.

56:06  
So these lecture sessions are two hours.

56:09  
I don't normally use the full 2 hours.

56:11  
It depends on the subject.

56:12  
We end up going further sometimes, but we'd normally have breaks in two hours.

56:17  
But today I've cleared the slides.

56:19  
There seem to be no more questions.

56:21  
You can ask me questions about interest in commercial projects.

56:24  
I won't tell you the names, but I can tell you all sorts of fun stories in the lab.

56:30  
You will come up against some realistic scenarios and you have to try and work out what you're going to do with them, which could be rather fun anyway.

56:38  
So that.

56:38  
Yeah, you've got a question.

56:42  
Yes, Good, good point.

56:50  
Yeah.

56:50  
So the group project is essentially it's going from a high level idea to providing what would look like a realistic design document.

57:02  
So it's not a lot of text that you have to write, but you come in, you you want to describe the features that are in there.

57:09  
You've got a few user interface pictures and essentially that would be something that you'd have to give to a customer.

57:17  
So it's as realistic as I can do within the time.

57:20  
And yeah, it's in a group because that's normally how you work even on a design project.

57:27  
All right, So you don't need to write lots and you don't need to cite a lot of documentation.

57:31  
In fact, you don't need to cite anything for design project.

57:34  
OK, It's more like a thinking process.

57:37  
But when we get to that point, you'll you know, you'll have the brief and you'll be able to ask questions.

57:42  
There are two weeks for that project, but in a group of four or so, it's pretty easy to clear it.

57:48  
It won't take you very long.

57:52  
It won't.

57:53  
It's quite easy.

57:54  
You just get into a, you can book yourself a little meeting room like the G rooms and whiteboard and, you know, scribble around, come brainstorm your ideas, then write that up, come back, brainstorm a bit more.

58:08  
Somebody does a bit of user interface designs, then a bit of text around it to say how it's going to work.

58:13  
That's it.

58:15  
OK, great.

58:17  
All right then.

58:18  
Well, I'm going to stick around in case anybody's shy.

58:21  
Otherwise, hope you have a nice coffee break before your next session.