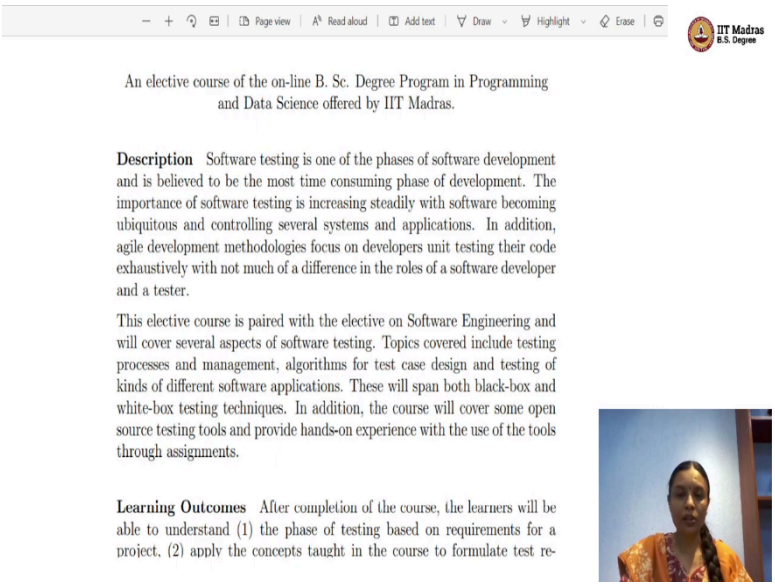
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Information Technology, Bangalore
Software Testing Motivation

Hello dear learners, to introduce myself, I am Meenakshi, a faculty member at IIIT at Bangalore, where I have been for the past 12 years. I work in the area of software testing and formal verification and teach courses on software testing, discrete maths, algorithms, graph theory, and formal verification back in IIIT Bangalore.

I will be teaching you this course on software testing, which is paired with the course on software engineering as a part of the online degree program, or the B.Sc. third-year course. So, to start with, I will talk to you about what the course has to offer, what will be the learning outcomes, and the syllabus of the course. And as a part of this video module, we will also see a brief motivation software testing. So, let us get started.

(Refer Slide Time: 01:09)

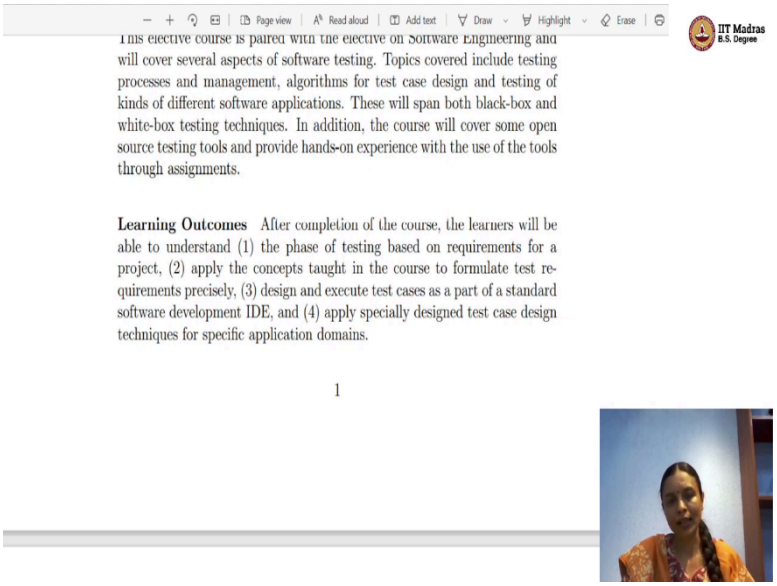


So, what is this course going to teach you? So, as I told you, software testing is paired with software engineering and is a vital part of the entire process of developing software that we all use. So, in addition, the agile methodologies which many companies follow these days put a lot of stress on developers' unit testing and integration testing of their code.

So, Software Testing cannot now be considered as a job that will be done only by testers and not by developers, everybody has to do software testing of their code themselves. So, this course will teach you all the aspects of software testing from the process-oriented aspects,

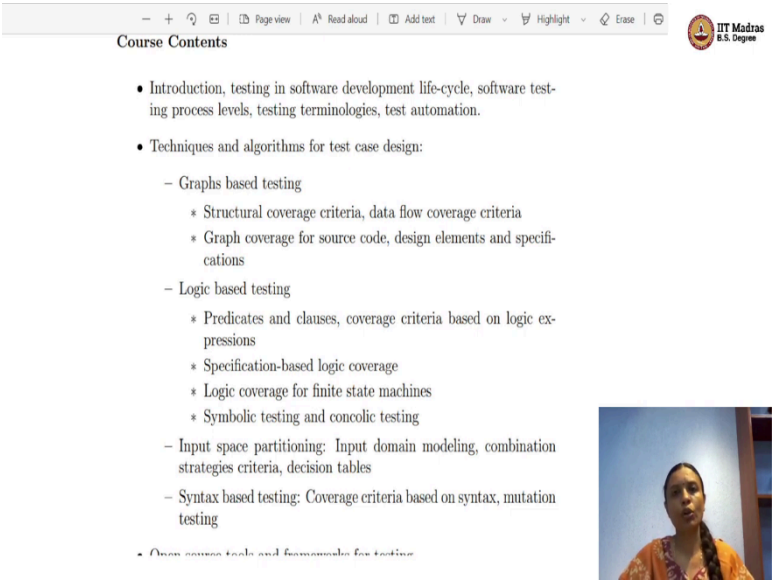
theoretical aspects, algorithmic aspects, and tool-oriented aspects where we will use open-source tools to learn how to test your code.

(Refer Slide Time: 02:06)



So, in terms of the learning outcomes, when you complete this course, you will be able to understand, which is the right phase of testing that you are in for your piece of code, how to phase out your testing into bits and pieces that go in phases. Apply the concepts taught in the course to design very effective test cases, and write your test requirements. To be able to design and execute test cases using an open-source integrated software development environment and an open-source test automation tool. And also learn how to extract design models from software artifacts that you can use to design your test cases.

(Refer Slide Time: 02:51)



So, what are the contents of the course? In the first week, like we are in now, I will introduce you to the concept of software testing. Talk about what role testing has to play in the software development lifecycle, motivate you to software testing, what are all the common terminologies, terms used in software testing?

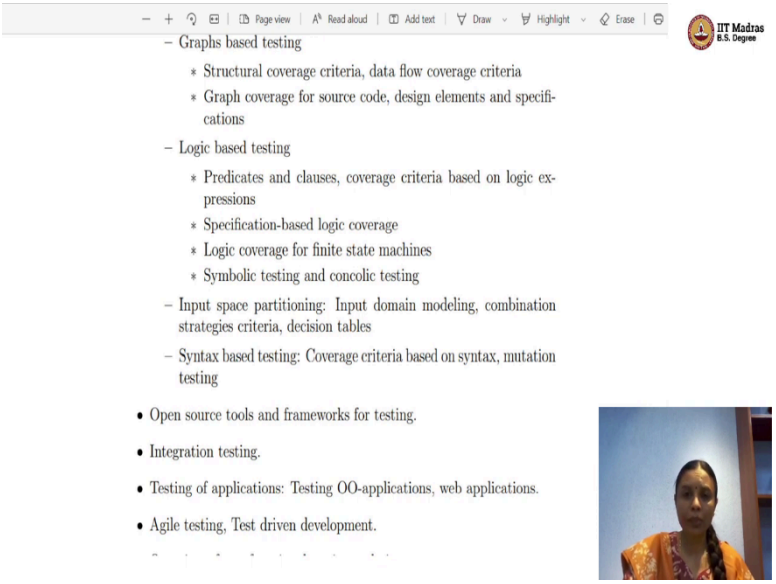
And what are all the process levels that various companies follow for doing software testing, and what is a set of good open-source tools that you could do to help you automate the testing process, that is what we are going to learn in the first week to start with and in the course. Later, a good part of the course is going to focus on test case design, where from your software artifact, which is predominantly code, you will learn how to extract what we call models, and then design test cases based on these models.

One of the first such models that we will be working with are graphs, you will learn how to extract graphs from source code from design elements and from specification entities and then we will work with what we call control flow graphs, data flow graphs and generate test cases from them. This will teach you how to design test cases by using graphs as the basis in this module.

I will also help you to revise the basic algorithms on graphs and the concepts on graphs that you might need from the point of view of software testing. After doing graphs, we will move on to logic, again I will teach you the fundamentals of logic, a revision from what you would have done in a course on discrete maths, but these fundamentals will cover whatever you need for the course on software testing, you will understand what a notion of a predicate is, what a clause is?

Where do these come in artifacts like code in requirements documents, how to extract them, how to design and execute test cases based on these predicates? A very prominent trending technique as far as logical-based testing is concerned is symbolic execution. So, this module will also teach you how to test based on symbolic execution techniques.

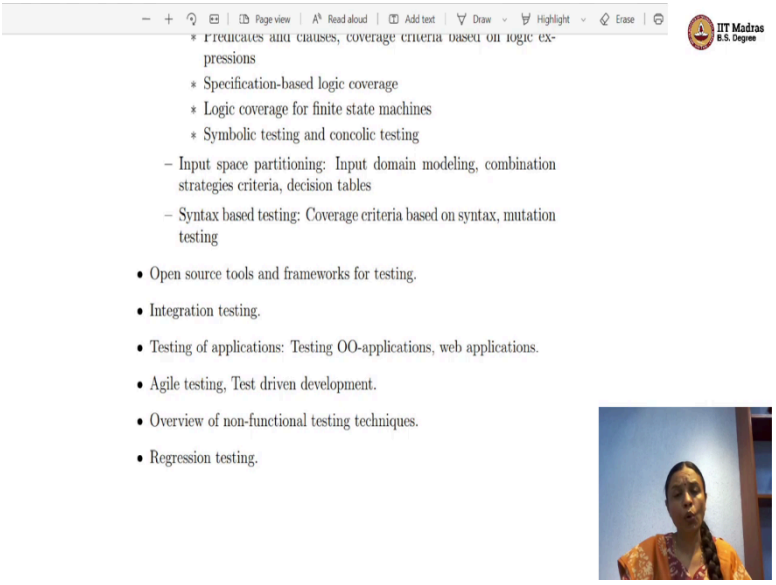
(Refer Slide Time: 05:24)



After this, we will look at a black box testing technique called input space partitioning. Where we look at the requirements document, study the inputs to your software, and learn to generate black box test cases based on these inputs and the requirements that dictate these inputs. So, we take the input space learn partitioning techniques that will partition the input space, and generate test cases based on this.

You will also learn a bit about decision tables and how to generate test cases based on them. Finally, one last design technique generic technique that we will learn is that of syntax-based testing, or what we call mutation testing, we will consider the program as a syntactic entity and learn how to generate low-level test cases that can be used to test and debug your program that will avoid and detect common programmer mistakes.

(Refer Slide Time: 06:26)

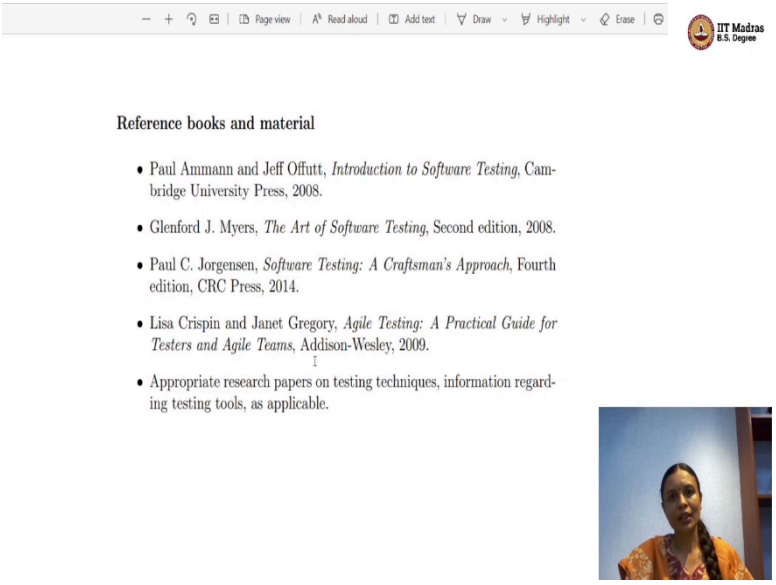


So, after these basics of test case design, you will also hand in glove learn to use open-source tools mainly a framework called J Unit, which we will use to generate the test cases and automate their execution. After this, you will learn a bit about integration testing, especially testing for design elements. And then the course will move on to focused testing techniques for various classes of applications.

Object-oriented applications that follow object-oriented design aspects are very popular, web applications are very popular. So, we will learn how to test object-oriented applications and how to test web applications. What are the testing techniques specific to the testing of such applications? Agile methodology is a popular software development methodology. So, we will learn a bit about agile testing and test-driven development.

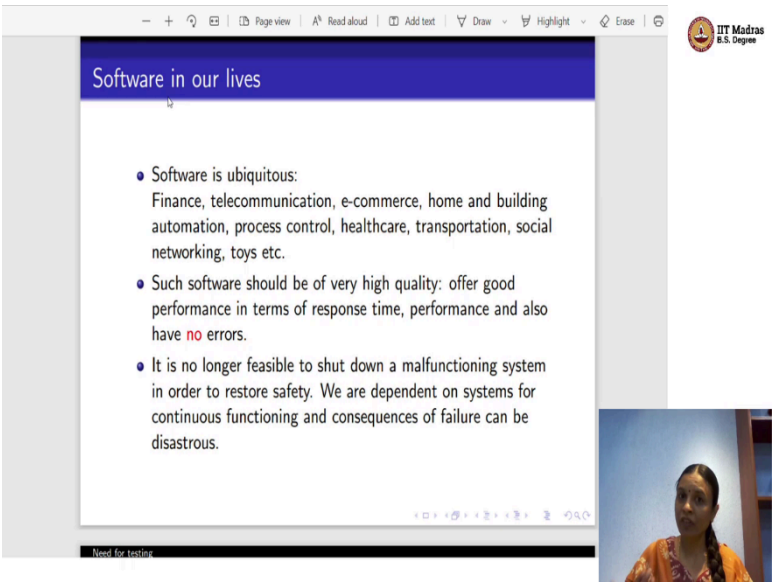
A part of testing is also devoted to what we call ensuring quality. So, we call this non-functional testing or quality focus testing. So, this course will give you an overview of non-functional testing techniques too. The course will also teach you about regression testing, which is done post-deployment of software.

(Refer Slide Time: 07:45)



So, that is the contents of the course for you. In terms of the reference books, and material, this gives you the predominant set of books that we will follow throughout the course, for most of these books, you will find soft copies online, I will also help you by providing appropriate papers, reading materials, slides and other information that you might need as we move on in the course. So, that was an overview of the syllabus and the contents of the course.

(Refer Slide Time: 08:31)



Let me continue in this first module by motivating software testing or helping you understand and appreciate why we need software testing as a subject itself. So, if you pause for a minute and focus and ask yourself the question, do I have software in my life? Is software present in

our everyday lives? The answer is very easy yes, software is ubiquitous, it is very much present in our everyday lives.

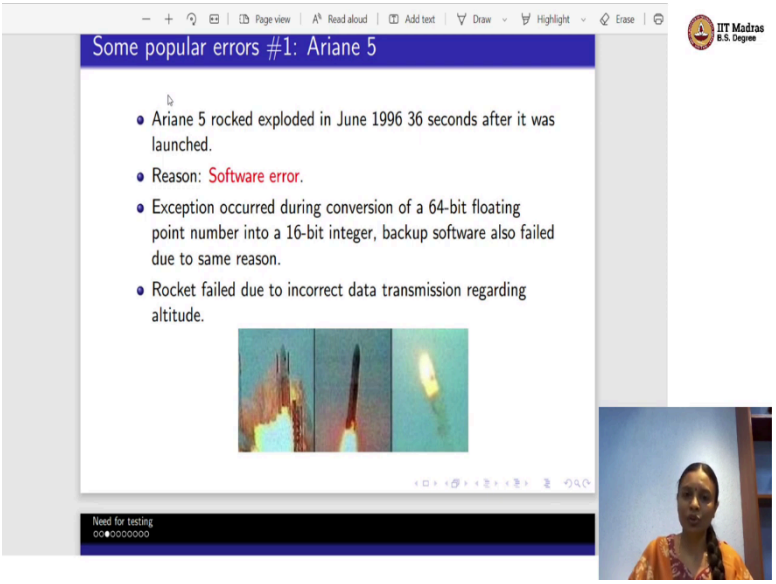
If I have to access my bank, I use a piece of software from my mobile app or the web browser. If I have to make a phone call to somebody or send a message to somebody using my phone or using some other communication. If I need access to the internet, I need software. If I have to buy things online, I use software. I have Alexa at my home, which is all the time helping me to make my life better. So, to say that is another piece of software.

Software in our daily lives exists very much, software also exists in homes, in building controls, in industries, in hospitals, in cars, planes, and railway networks, software exists for us to be entertained by social media and social networking, they are all software, toys that children's use, also have a lot of software in them. So, there is no doubt about the fact that software is omnipresent and ubiquitous in our lives.

Next, if you talk about it, you know we want the software to be of very high quality If we want to work fast, keep pace with our fast-paced lives, perform well have a very good user interface, and ideally, never fail. That is have no errors. The other important thing is that we need software to be working continuously up and running all the time.

The notion of "ohh!" is there a problem, let us shut it down, fix it, and then restart it to get it up and running, does not work for most of the software that we use in our lives. So, we want systems to be continuously functioning, and errors if any should be fixed as the system is up and running.

(Refer Slide Time: 10:41)

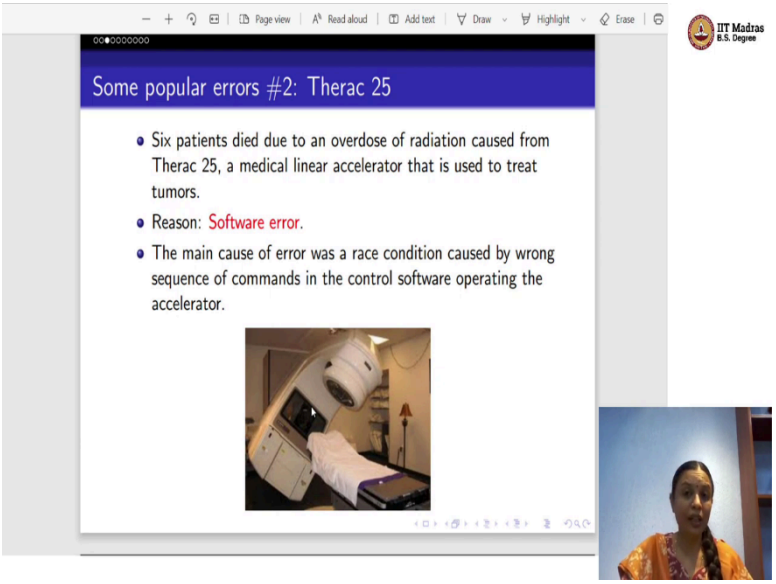


Before we understand any aspect of testing, one good motivation for testing comes by looking at what have been big software errors in the history, there have been numerous numbers of errors that are related to software, exemplar errors, but I will highlight a few of them that are big examples of failures because of software errors.

One of the early examples, in June 1996, was the failure of this rocket called Ariane 5, which was developed by the European Space Agency, this rocket was launched after 10 plus years of effort, millions of euros spent, and very unfortunately destroyed itself within 36 seconds after it was launched. And the reason why this big fault happened was because there was a software error, there was a piece of software, which was trying to convert a 64-bit floating-point number and write it into a memory space that was meant for a 16-bit integer.

Obviously, it did not have enough memory. So, it failed. These are what we call safety-critical software. So, there is always backup software available. But in this case, the issue was that the backup software also had exactly the same error. So, both the original software and the backup software failed, incorrect altitude information was passed to the rocket, the rocket ended up destroying itself.

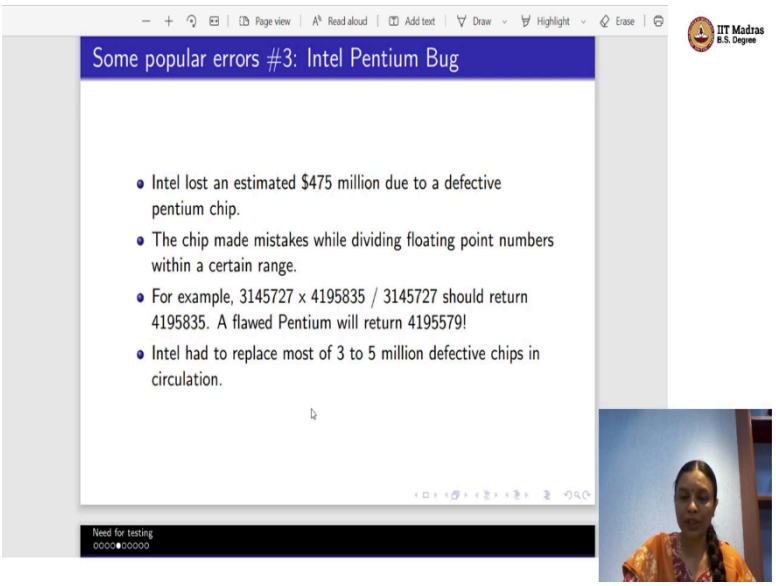
(Refer Slide Time: 12:19)



Another popular error that you find in history unfortunately resulted in the death of a few patients. Cancer patients receive radiation therapy by these machines that you can see in these pictures once that sample machine is given here. And these machines use a lot of software to calculate the amount and the duration of radiation that these patients have to receive as a part of the therapy.

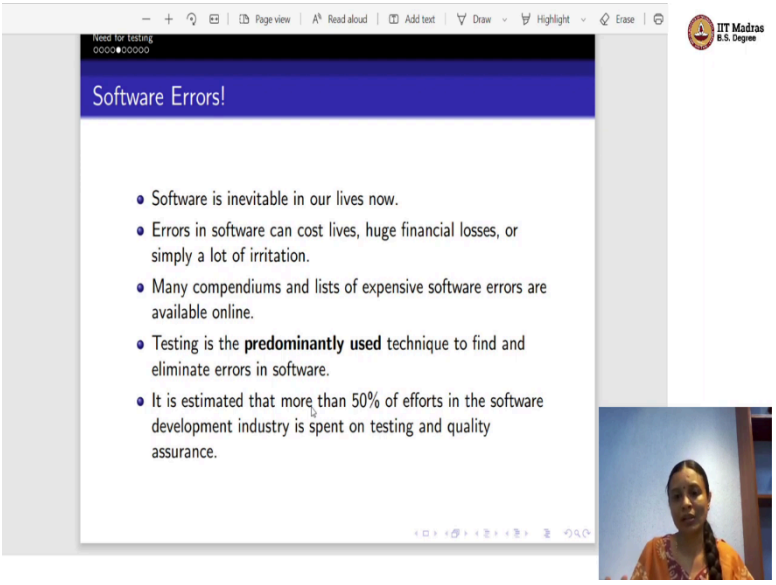
One such machine gave an overdose of radiation because it calculated an overdose of radiation, and that resulted in a few patients losing their lives. The reason for this is software error. There was a race condition in the software, which led to a wrong interleaving of commands of several threads, which ended up in the software calculating an overdose of radiation and delivering the same to the patient, resulting in death. So, these are very unfortunate software errors that have happened.

(Refer Slide Time: 13:19)



Another famous error, which resulted in a lot of loss to a big firm is this famous Intel Pentium 4 bug, Intel is estimated to have lost about 475 million dollars. Because at that point, it is P4 bug was doing floating point division in a wrong way. Such an error was found by a mathematician as a process as a part of his calculations and because of this, Intel had to recall all the P4 chips that were under circulation in the market and replace them, resulting in huge cost to the company. And the reason is another software error.

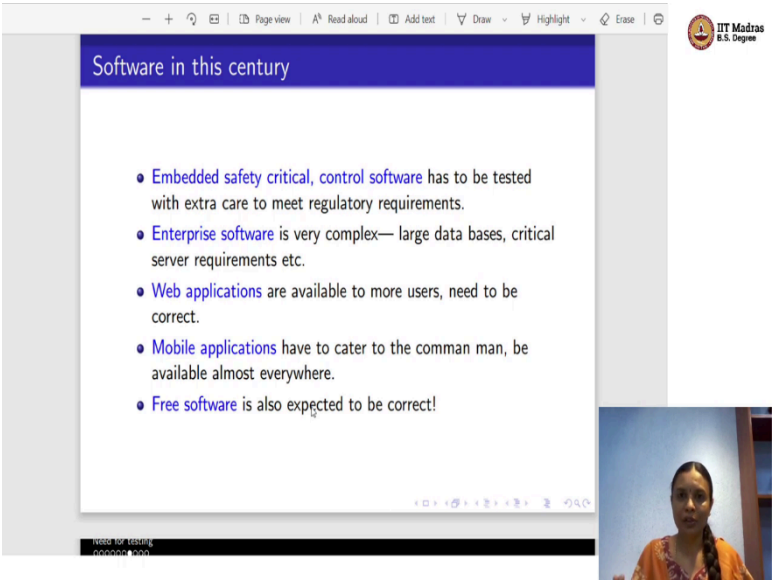
(Refer Slide Time: 14:00)



So, by now you get a good idea of how bad software errors can be, right? We know on one side that software is inevitable in our lives. It is present everywhere. But an error in this software can be very expensive as these examples illustrate it can be expensive because it can cost lives. It can be expensive, because it can cause huge financial losses or it can just cause us a lot of irritation by being slow and not working. In all of the cases, errors are unacceptable.

If you quickly take a break, and Google you will find many compendiums and lists of software errors available online. I encourage you to go through them to understand the impact that these errors have in the working of the software and resulting in an impact in our everyday lives. How do we eliminate them, how do we systematically find them and ensure that they are not there? Testing is the only predominantly used and time-tested activity to find and eliminate software errors. It is estimated by the software industry stalwarts that more than 50 percent of the total industry's efforts is spent on testing and quality assurance.

(Refer Slide Time: 15:23)



So, now that we have seen errors, and we have seen the testing can solve them, let us spend a few minutes trying to understand the kinds of software that we work in our everyday lives. So, the parts that are highlighted in blue in this slide are one classification, not exhaustive of the kinds of software that you will see.

The first kind, which is the kind of software that I do research on, especially when it comes to testing is what we call embedded software, the software that you would find operating the power window in your car, operating the parking assistant in your car, operating the GPS in your car, or maybe operating rail network, the software that helps with autopilot or landing in an aircraft, a software that operates our lifts, they are embedded as a part of a system that is tightly coupled to the hardware, and almost always safety critical.

If there is a problem in the software, it will immediately result in accidents. So, we want them to be error-free. Other kinds of software is what we call enterprise software, software that runs in the banking and financial sectors, software that operates at the backend of all these e-commerce, companies, and so on. They operate with large databases, they have critical server requirements, and several clients accessing them online, they all have to be correct, they cannot fail.

The next kind of software is web application, software that runs through our browsers that we use, they also have to be correct, they have to be available to several different users who are not necessarily proficient software developers, because they are meant to be available for the

public to use for the public to consume several different languages. And they have different kinds of correctness requirements and different kinds of errors that come in them.

Another popular category of software is the mobile applications that again, most of us are dependent on, they also cater to the common man, and have to be available almost everywhere, and errors. And these can be expensive. Ironically, we also use a lot of free software for which we do not pay for. And we expect all that free software to you that we use to also be free of errors. So, the kinds of errors that come in each of these kinds of software are very different. And the kinds of techniques that we might need, can also be different to find and to fix these kinds of errors.

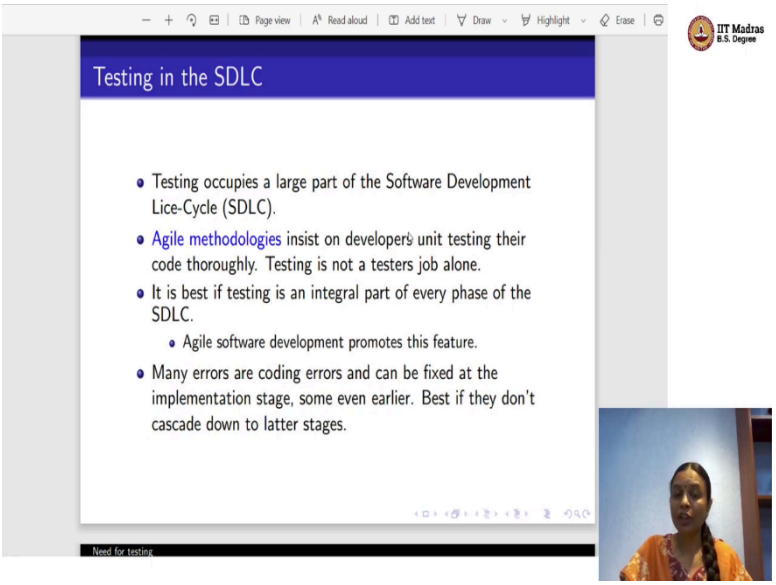
(Refer Slide Time: 18:00)

So, the next important thing that motivates why we need to test early and why is testing important is software typically is developed using a well-defined process is what we call a lifecycle. I will tell you in the next module, what a software development lifecycle is, you will learn more about it in the software engineering course. So, initially, you write what we call requirements, then you do what we call design, then you write coding, and then you do testing and deploy software.

Suppose you find an error early on. That is when you are writing the requirements or design of your software, then the cost of fixing that error is cheaper. That is what this graph says, look at the red bars that depict the cost in this picture. Suppose you find them at the level of unit testing, that is while coding, or at the level of integration testing while putting together your code, then also, it is not too expensive to fix these errors.

Suppose you find them later. Specifically, after you deploy the software, look at how tall this red bar goes. The cost of finding an error and fixing it gets very expensive, the kind of errors that I told you about earlier, be it in the rocket or be it in the radiation machine, or be it in the Intel Pentium chips. They were all errors found post-deployment and hence very expensive when they actually come in terms of finding them. And the cost that it has in terms of loss of lives or loss of finance is very expensive. So, the earlier that we find errors, preferably, the better it is for us. So, it is important to test early as you are writing your code.

(Refer Slide Time: 19:48)

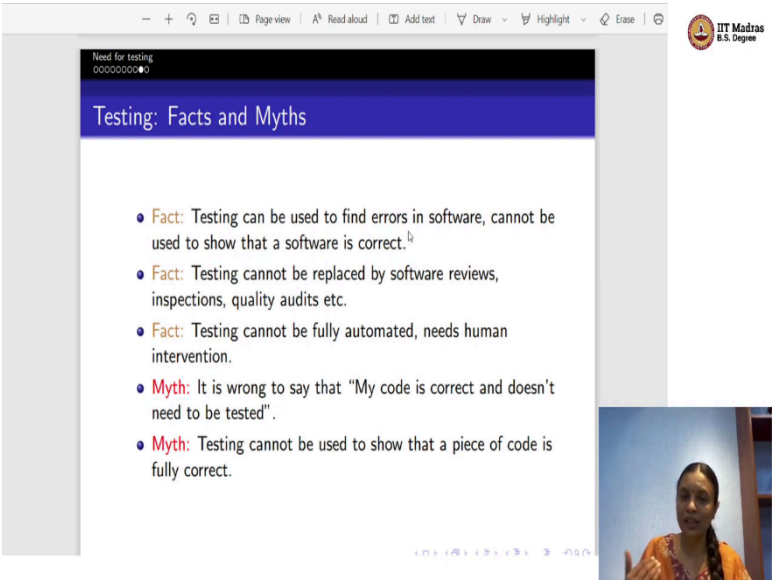


So, for testing, we will walk through the software development lifecycle. I will tell you what, SDLC is briefly in the next module, and it occupies a lot large part of the software development lifecycle. These days most companies follow agile development methodologies and agile methodologies insist that developers unit test their code themselves. You cannot say I have written this cool code, I think it is working fine.

Here you test, take my code, test it, and let me know where there are errors. That does not work that way. In agile, as a developer, the onus is on you to unit test your code also. So, testing is should be an integral part of every phase of SDLC. And agile methodologies which are being followed these days do make this a reality.

The other truth that you should know is many errors are basic coding errors, which are mistakes made by programmers. And they are best found, then in there, as a programmer or a developer unit tests his or her code, it is best to find errors right there and fix them. It is best if they do not move on to later stages of the software development lifecycle.

(Refer Slide Time: 20:59)



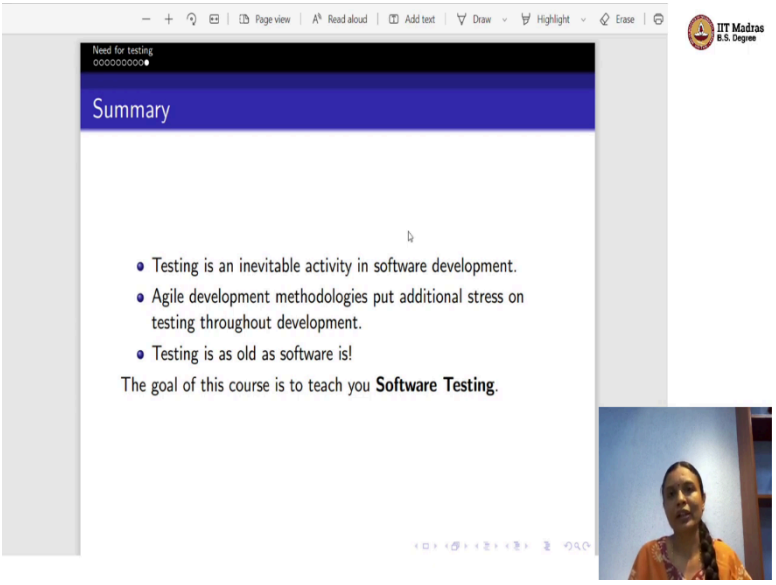
So, I would also like to use this motivation module to tell you what are the popular facts and what are the popular myths about software testing. Let us start with the facts. The facts are testing is a well-known technique to find errors in software, every error that testing finds is a real error that can be fixed.

But using testing is wrong to say that my software is fully correct, that I have tested my software well and my code is fully correct testing cannot be used to make such a statement. Testing can only be used to find errors and fix them. At no point in time will testing give you the confidence that your software is fully correct. I hope you understand the difference between these two terms.

The other thing is that testing also has to be done alongside other quality review processes, which are reviews done by a quality inspection team. Reviews done by a dedicated software quality auditor and so on. These two together are very effective in finding all the errors later in the course, we will do one module on this quality, what is inspection, and things like that. The third important fact that you need to know about testing is that testing is heavily automated.

There are lots of tools that will help you with testing. But it cannot be fully automated. You need a human being you need to know the domain of the software and you need to know certain aspects of software testing, to be able to design effective test cases, which will be effective in terms of finding bugs, the execution part can be automated test case design needs algorithm acknowledge needs human intervention.

(Refer Slide Time: 22:56)



So, that was a summary. I hope I motivated Software Testing enough for you. Remember that it is an inevitable activity of software development. And agile methodologies put a lot of stress on testing. If you are going to take up a job as a software developer, you are going to be in a company that is agile in some way. So, you will have to do testing a lot.

And testing is as old as software is and tell you why we will take some examples from real historical code and talk about errors that people found in that code. So, the goal of this course is to teach you all the software testing that you need to know as a developer. Welcome to this course.