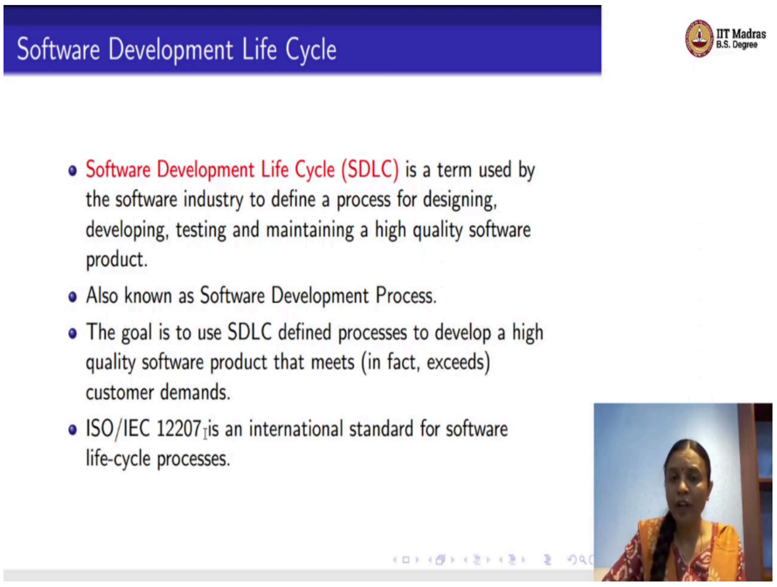
Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Technology, Bangalore
Software Development Life Cycle

Welcome back, this is the second lecture of week one. Before we move on to learning about software testing, it is important to get an idea of the entire software development life cycle, testing is only a part of it. The paired course on software engineering will teach you a lot more about the software development life cycle, but for completeness's sake, I thought, I would start giving a brief introduction on the software development life cycle with, a little bit of focus on where testing might come into picture. So, this module is on a brief introduction to the software development lifecycle for you. So, let us understand what software development lifecycle is about.

(Refer Slide Time: 1:02)



Software Development Life Cycle

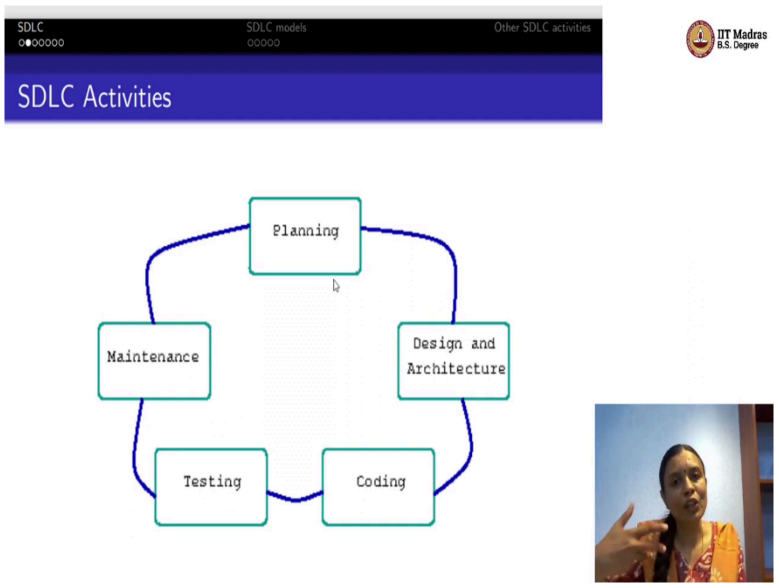
- Software Development Life Cycle (SDLC) is a term used by the software industry to define a process for designing, developing, testing and maintaining a high quality software product.
- Also known as Software Development Process.
- The goal is to use SDLC defined processes to develop a high quality software product that meets (in fact, exceeds) customer demands.
- ISO/IEC 12207 is an international standard for software life-cycle processes.

So, Software Development Life Cycle abbreviated as SDLC, is a broad umbrella term used by the software industry. To define a process a way by which they would design a piece of software, decide on what to build a software for, develop it, test it, and release it to the market or give it back to the customer who ordered the development of software for them. So, it is a set of processes or it is a way that tells you how a team in a company would go about creating a software end-to-end. It is synonymous with the term software development process and abbreviated as SDLC.

The goal of an SDLC is to define processes or ways by which one would strive to develop a high-quality software that will meet all the expectations from the customer and, in fact not only meet but exceed all the expectations from the customer. Like in all other industry, the software development lifecycle also has an ISO standard, and the ISO IEC standard 12207 is an international standard that defines software lifecycle processes.

Feel free to go through it a bit to the extent that you can get on open sources. You will get to know what the standards are and how they define these software development processes. Many companies especially, the large-scale companies tend to follow these kinds of standards to be able to define the software development lifecycle processes within their organizations, these standards help them to do that.

(Refer Slide Time: 2:53)



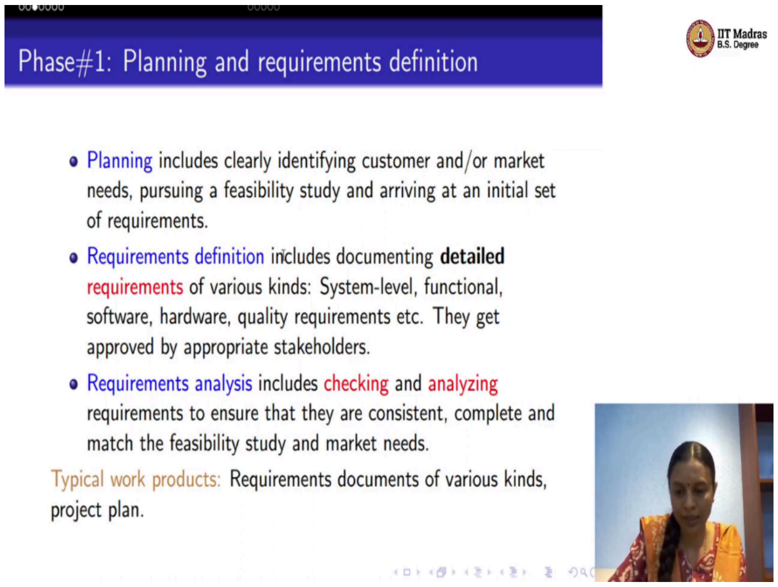
SDLC Activities

So, let us understand what the activities that are there in a software development life cycle are. To start with, right in the beginning, you do what is called planning. You need to know what are you going to write software for, and what are your development goals. What are the requirements that the software should satisfy? Who is going to give me the requirements? Who is going to check whether the software will meet the requirements? how much time do I have to finish developing this software and release it? How many people do I have? How do I delegate tasks? So, many different questions to be asked and answered in the planning phase.

After this, you also collect requirements which is an integral part of the planning phase, that is what is your software meant to do. Once you collect requirements you move into what we call designing and architecting software which comes up with the skeletal framework of how you are going to develop your software. Post which, you typically do coding which is the implementation phase along with that do testing which is to test if it meets all the requirements and if it is otherwise error-free, and after release your job does not stop there you cannot sit back, and say I am done with this cool software that I have written, handed it over to the customer I am done, maybe there will be errors that software will have post-release, maybe the customer will ask for new feature enhancements. So, most of the software goes through what we call a maintenance phase which comes post-deployment.

So, these things are put in a circular fashion overlapping with each other because they are heavily dependent on each other in terms of various SDLC activities. So, how do I know which one follows the other? How do I fix the order? Which would be the right kind of activity to do and when? Those are the kinds of things that the various models teach you.

(Refer Slide Time: 5:00)



Phase#1: Planning and requirements definition

- Planning includes clearly identifying customer and/or market needs, pursuing a feasibility study and arriving at an initial set of requirements.
- Requirements definition includes documenting **detailed requirements** of various kinds: System-level, functional, software, hardware, quality requirements etc. They get approved by appropriate stakeholders.
- Requirements analysis includes checking and analyzing requirements to ensure that they are consistent, complete and match the feasibility study and market needs.

Typical work products: Requirements documents of various kinds, project plan.

We will first understand little by little what each phase does. Typically, the first phase in the software development life lifecycle is always the planning and the requirements definition phase. As I just told you planning includes clearly identifying what the requirements of the customer are. what are the requirements that will identify me and distinguish me from what is already offered in the market? what are the market needs? Pursue a feasibility study, find out how much time you have, and come up with an initial set of requirements.

You take that initial set of requirements, elaborate it to a larger set of requirements of various different kinds. Typically, the output of this would be at least half a dozen requirements documents, so you will have what we call a system-level requirement, functional requirements, software requirement, you also call them a Software Requirement Specification SRS document for short, you will learn about these in the other course. Hardware requirements, requirements for quality performance, latency, reliability, maintainability, user interface, they all get approved by the appropriate stakeholders, you create lots and lots of requirements at this stage.

Typically, can take several weeks or even months to do this. Not only create requirements, you also begin to analyze the requirement, is my requirement feasible? Am I asking for something that is impossible to develop? Impossible to design? You ask yourself that question, try to arrive at an answer. Is a requirement correct? do two or three requirements contradict with each other? Does one requirement imply that the other requirement is impossible to achieve? Is there a cascading down dependency that is natural between the requirements?

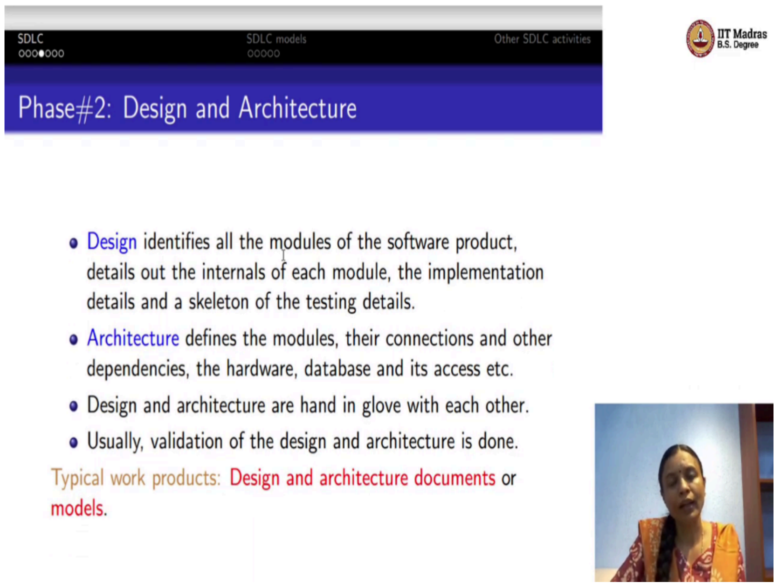
So, you also go through what we call a requirements analysis where you check for all these properties, is my requirement consistent? Have I left out anything important without specifying it? does one requirement contradict the other? is there a natural flow between the requirements? you analyze these kinds of things at this stage. So, typically a work product here would be requirements documents of various kinds, as I told you one for software, one for system level, one for hardware, one for functional, one for non-functional, so on, the list can be really long.

You also come up with what we call the project plan which typically identifies the duration that this entire development effort is going to take. By when will the product be ready? Do I give intermediate releases? How many people do I need? For how long are they going to work? that is what the project plan is going to talk to you about.

If you take an example and we sort of walk through that example down the SDLC, let us take something that we all use. Let us say I have, I use Swiggy to order food for my family once in a while, Swiggy what would be typical requirement for a software like Swiggy? Basic system level requirement must be that it should be available as an app on my phone, it should be available on the web browser also for somebody who is not familiar with the phone, maybe it should be available in different languages, it should correctly tell me which restaurants give what kind of food, it should have a good estimate on the time that it takes for the restaurant to be able to deliver, sitting here, can I order forlet us say my friend or my relative in another city. You know these are all what we call requirements.

Some of them can be exclusively about software, some of them can be about user interfaces, and some of them can be about the system-level requirements, you basically collect categorize, create these documents, and plan at this stage.

(Refer Slide Time: 9:01)



Phase#2: Design and Architecture

- Design identifies all the modules of the software product, details out the internals of each module, the implementation details and a skeleton of the testing details.
- Architecture defines the modules, their connections and other dependencies, the hardware, database and its access etc.
- Design and architecture are hand in glove with each other.
- Usually, validation of the design and architecture is done.

Typical work products: Design and architecture documents or models.

After this, comes the second phase where you design and also define the architectural elements of your software. What is the difference between design and architecture? Hopefully, the other course will tell you, but broadly speaking design identifies all the modules of the software product details out the internals of the module tells you that this is the high-level design.

Let us say, you use object-oriented design concepts these are going to be my classes, this is the class inheritance hierarchy, these are the methods, some of these methods are polymorphic, these are the constructors, so this is how you talk about design diagram, typically, written, maybe.

As a set of class diagrams, that talk about inheritance may be written as UML models, whatever you want to call it as. So, design talks about all the modules of the software product, tells you what the internals of each module is going to look like. The low-level design almost talks about the code and also gives you a skeleton to generate what we call integration tests later.

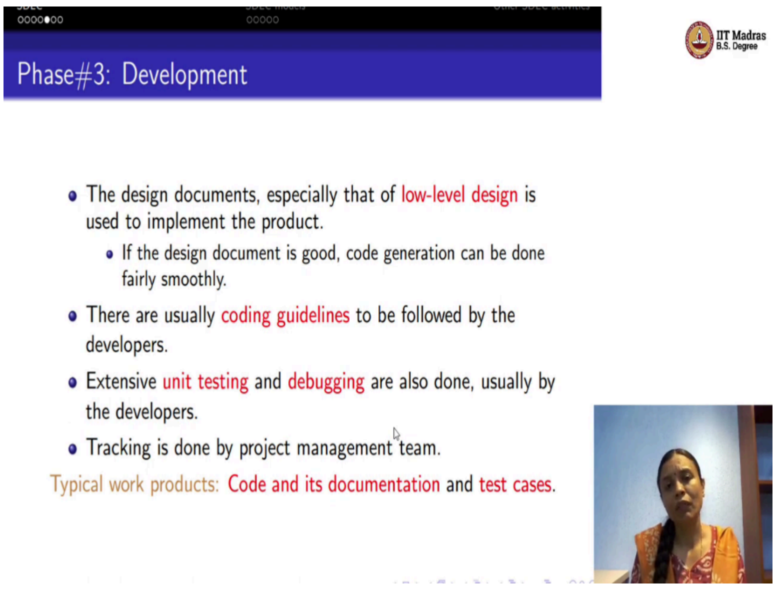
Along with design comes architecture, which talks about the modules, how they are connected, like for example suppose I take this Swiggy app, I want it to be available on a phone that has Android, I want it to be available on iPhone also. I also want it to be available from let us say the Edge browser, from Chrome browser, or from Firefox browser, this is what the architecture is going to focus on.

What does it mean? Various operating systems that it is going to run on, various databases that it is going to access, the browsers that it is going to run on, and other user interface aspects of it, is what the architecture is going to talk about. Typically, design and architecture always go together, they are hand in glove with each other.

We also validate the design and architecture. Suppose my architecture demands that I need the coolest of the operating systems which is yet under production, then it is not a feasible architecture, I need to be able to cater to the currently available products in the market on which I can architect my product on. So, we do feasibility study, we also do what we call system-level test cases from these documents.

So, the typical work products are, documents that have designed an architecture, they could be static documents like pdf files, or some other documents that you create in an editor or they could actually be models like you could create design and architecture models in dedicated languages, modeling languages that are present for them UML, SYSML, AADL, any of these could have these kinds of models.

(Refer Slide Time: 11:52)



Phase#3: Development

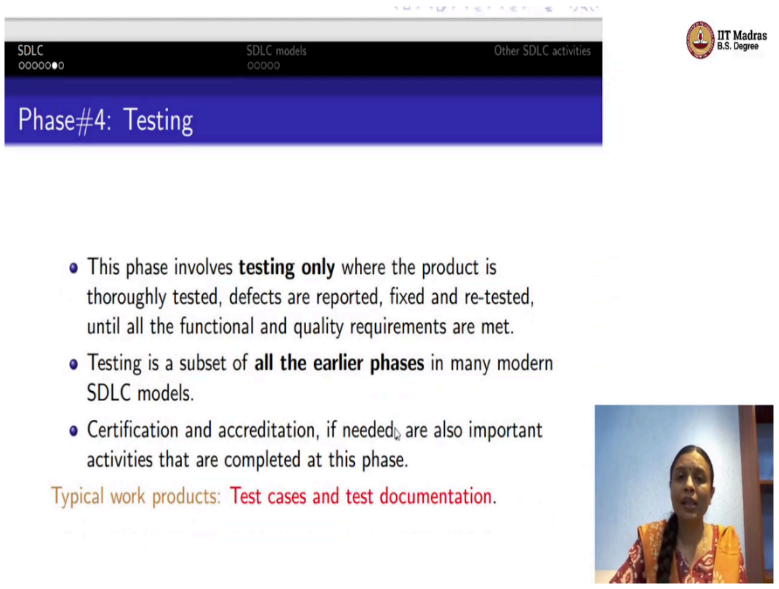
- The design documents, especially that of low-level design is used to implement the product.
  - If the design document is good, code generation can be done fairly smoothly.
- There are usually coding guidelines to be followed by the developers.
- Extensive unit testing and debugging are also done, usually by the developers.
- Tracking is done by project management team.

Typical work products: Code and its documentation and test cases.

The next phase after design and architecture is the coding or the development phase. Design documents typically contain what we call a low-level design that are used to write your code and if a design document is well done, modeling is well done, code generation is fairly smooth process. There are coding guidelines that the organization that you will work for will tell you and you have to follow those guidelines when you write code. Along with coding, agile methodologies especially insist on the developer doing debugging of the code along with unit testing himself or herself.

So, unit testing which is testing at the level of a method, testing at the level of a procedure or a function is done by the developer alongside coding. And in this development phase, a lot of the tracking is done by team leads and the project managers. Work products are fantastic documents here they are the actual executable code along with documentation and the test cases that you have used to unit test your code.

(Refer Slide Time: 13:02)



Phase#4: Testing

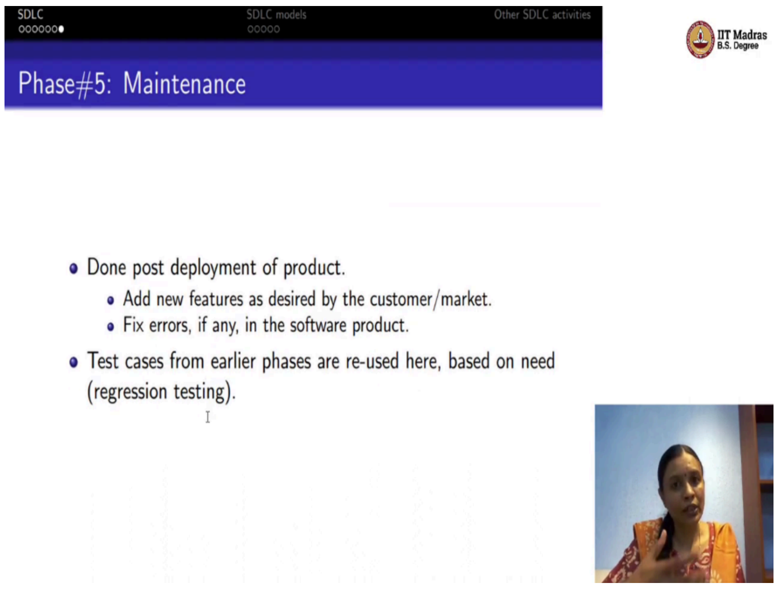
- This phase involves **testing only** where the product is thoroughly tested, defects are reported, fixed and re-tested, until all the functional and quality requirements are met.
- Testing is a subset of **all the earlier phases** in many modern SDLC models.
- Certification and accreditation, if needed, are also important activities that are completed at this phase.

Typical work products: Test cases and test documentation.

After this comes testing, obviously there was testing here itself in the development phase which is unit testing of the code done by the developer. The next phase is again testing but this testing is slightly different from the unit testing. So, this phase involves not writing any more code but focusing only on testing, where the product is thoroughly tested, defects are reported fixed, and retested until all the functionality is satisfied.

Typically, people do what we call integration testing at this phase where if two three modules are developed then they are put together or a software module is put together along with the database and the hardware and tested directly on the server. So, you could do software integration testing or system integration testing at this phase, and it is carries forward from all the earlier phases involved little bit of testing and validation, this phase focuses only on testing and validation. Along with this, wherever it is needed, if your particular software has to be certified by a third party, or accredited by a third party the kind of documentation and the testing that you would do for that is also done at this phase. Typical work products are test cases and test documentation.

(Refer Slide Time: 14:23)


SDLC　SDLC models　Other SDLC activities

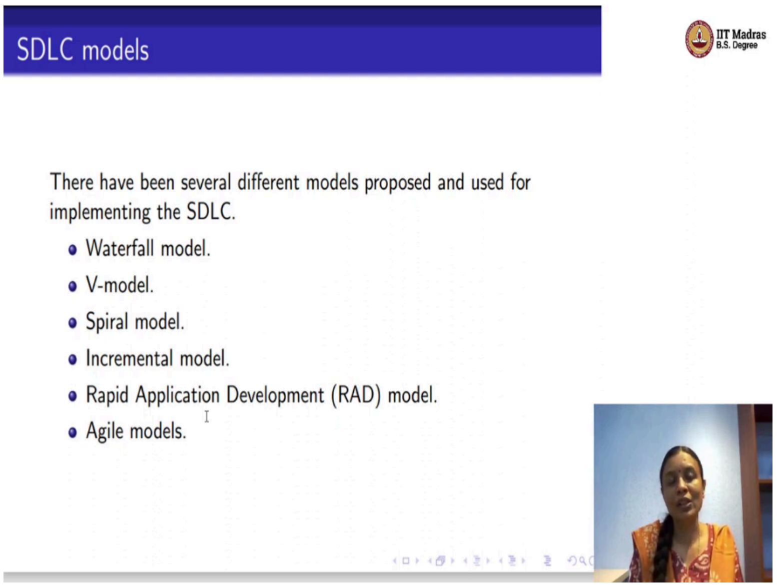
Phase#5: Maintenance

- Done post deployment of product.
  - Add new features as desired by the customer/market.
  - Fix errors, if any, in the software product.
- Test cases from earlier phases are re-used here, based on need (regression testing).

Post this, comes maintenance, you have finished developing a software, tested it thoroughly, system tested it, released it to the customer, there could be new errors that are found at that point in time or a customer could ask for feature enhancement of these errors. So, maintenance goes for longer sometimes, several years, till the lifetime of the software, post-deployment where typically any errors that are found are fixed and the software is retested.

Any new features or change requests that the customer wants are also added, and in this phase, test cases that you have created in all the earlier phases during development are reused partly and new test cases are created. So, there is a whole sub-area in testing dedicated to this phase and that is called regression testing, we will see regression testing in detail later.

(Refer Slide Time: 15:20)

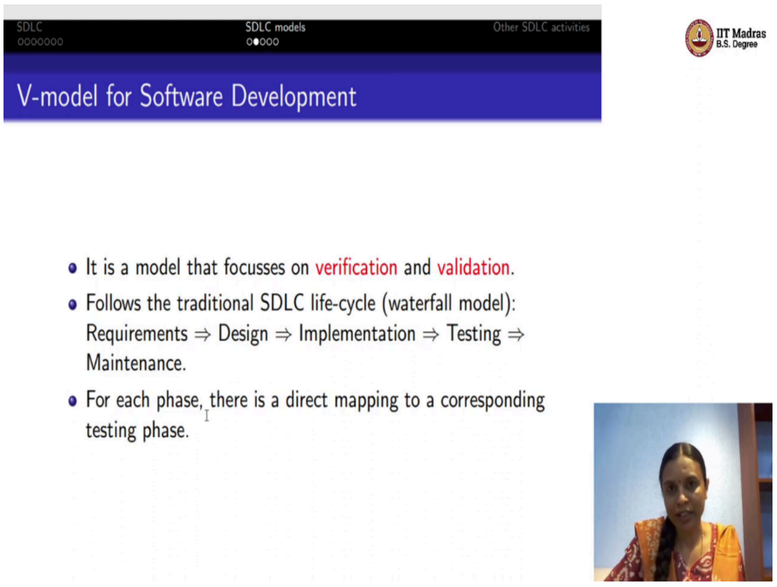

SDLC models

There have been several different models proposed and used for implementing the SDLC.

- Waterfall model.
- V-model.
- Spiral model.
- Incremental model.
- Rapid Application Development (RAD) model.
- Agile models.

So, that was about what each phase of the software development life cycle was. Historically, if you pick up any theoretical book on software testing, you would have learned about these software development life cycle models, waterfall model, V-model, spiral model, RAD model, incremental model, and big bang model, you know you can google for these terms, plenty of information on the internet and on software engineering textbook is available for these models. I am not going to walk you through all these models, but we will focus a few that is relevant for the purposes of moving on in this course.

(Refer Slide Time: 16:00)


V-model for Software Development

- It is a model that focusses on verification and validation.
- Follows the traditional SDLC life-cycle (waterfall model): Requirements ⇒ Design ⇒ Implementation ⇒ Testing ⇒ Maintenance.
- For each phase, there is a direct mapping to a corresponding testing phase.

So, V-model for software development is a model that extensively focuses on testing. So, it is a model that focuses on verification and validation, they are terms related to testing, they will be clarified in the next lecture. It follows the traditional SDLC life cycle, it is a lot like the water model, you first do the planning phase, write your requirements, then work on design and architecture, then do coding and unit testing, then do integration testing, deploy a software, and finally do maintenance related activities.

So, that flow is the same as the traditional waterfall model, waterfall model I did not tell you about, but it is just this. What is different between the V-model and the waterfall model, is for each of these phases, V-model does a direct mapping to the corresponding testing phase.

(Refer Slide Time: 16:54)

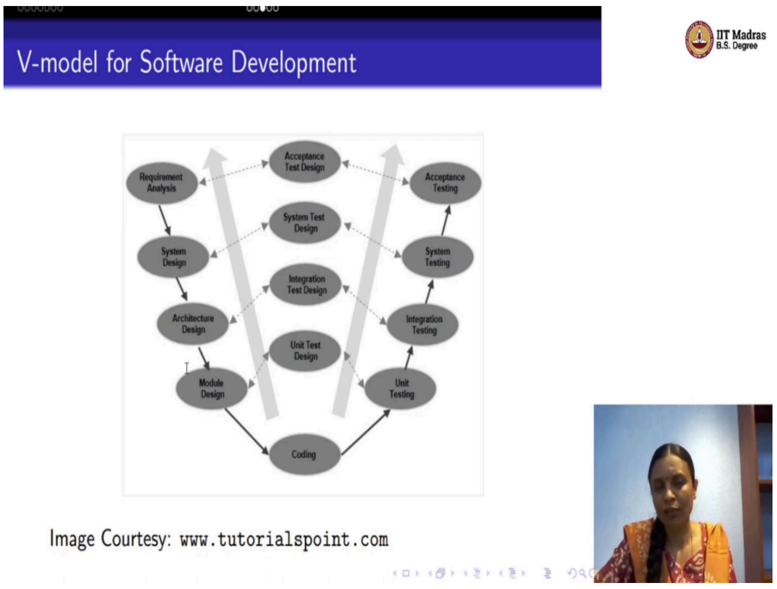

V-model for Software Development

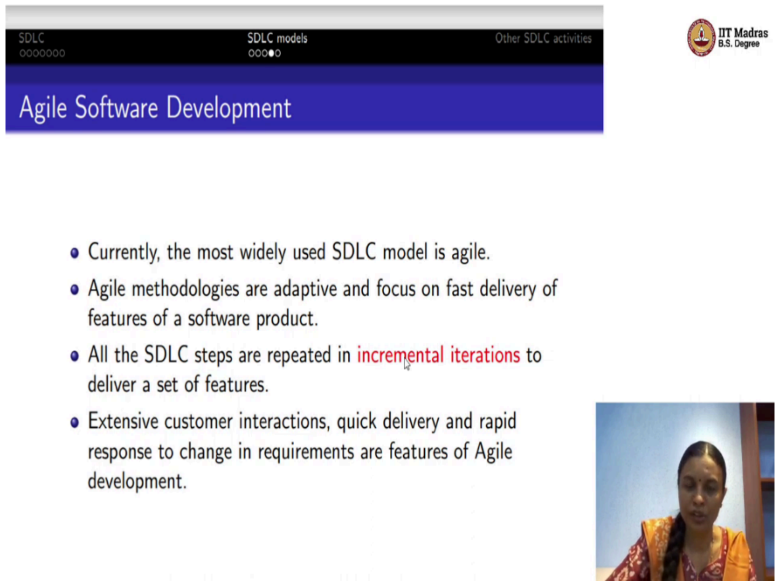
Image Courtesy: www.tutorialspoint.com

This figure that I took from www.tutorialspoint.com, which I extensively use is a pretty good website, will tell you what V-model is all about. So, if you look at this top left here, there is requirements analysis, the planning phase, followed by system design which we saw, followed by architecture which we again discussed, then low-level design, coding.

So, this is the left side of the V, I hope you can see the V here, and on the right side, along with coding, you have unit testing, then integration testing, where modules are put together, system testing, and finally, after deployment or just before deployment you do something called acceptance testing.

And what are these arrows that go across these layers? What is basically saying is the test cases for unit testing will come from the module design. The test cases for integration testing will come from architecture, the test cases for system testing will come from the design documents, the test cases that you use during the acceptance testing phase are likely to come from the requirements document. So, this V-model focuses a lot on testing, that is why, and it also fixes the dependencies on where the test cases will come from which faces.

(Refer Slide Time: 18:25)


SDLC | SDLC models | Other SDLC activities

Agile Software Development

- Currently, the most widely used SDLC model is agile.
- Agile methodologies are adaptive and focus on fast delivery of features of a software product.
- All the SDLC steps are repeated in incremental iterations to deliver a set of features.
- Extensive customer interactions, quick delivery and rapid response to change in requirements are features of Agile development.

Another popular development model which is practically ubiquitous these days is an agile software development methodology, it is not one particular model but it is a collection of methodologies, you will learn a lot about them in the other course, and again in the firm that you will be working on, currently, it is the most widely used SDLC model.

Agile as the word says means agility, be adaptive, customer needs are changing, market needs are changing, so be adaptive to it, do not wait for the entire requirements to be written out, for the entire design to be done, for the entire implementation to be done, then do the testing and maintenance, do it in small subsets is what we call features that are incremental in nature.

Release your software with first set of features, do everything across the SDLC in terms of faces for that release, add a new set of features as an increment, and do everything again release the new set of features, so that is what agility caters to. How do you know which is the first set of features which is to be added at every stage? So, agile models are extensively dependent on customer interactions, quick delivery, rapid response, and rapid development.

(Refer Slide Time: 19:44)

Agile model for Software Development

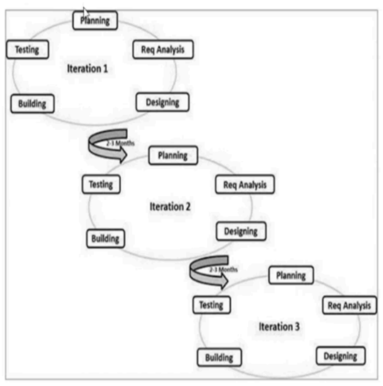
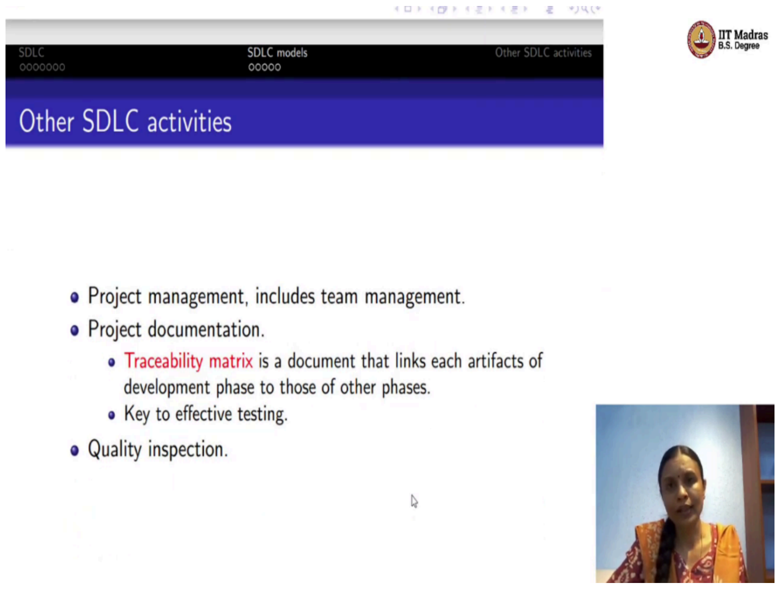


Image Courtesy: www.tutorialspoint.com

So, this is another picture that I got from tutorial point, which actually tells you nicely, how agile models for software development work in a generic way. If you see this blob here, there is iteration 1, iteration 2, iteration 3, and so on and each iteration, all the faces of the SDLC are encountered. In the first iteration, I plan, do requirements analysis, design an architecture, write code, test releases, release a set of features of my software, maybe the first version.

Second iteration, I repeat the whole thing again, plan, write requirements for this iteration, add design and architecture for this iteration, add code for this iteration, test, release again, iteration 3, and so on. Not exactly three iterations all the time, this is just meant for illustration, but I hope you get the essence of what it is. You essentially are agile, adaptive, and your software is developed in quick phases, what we call sprints in an incremental fashion.

(Refer Slide Time: 20:48)



SDLC | SDLC models | Other SDLC activities

Other SDLC activities

- Project management, includes team management.
- Project documentation.
  - Traceability matrix is a document that links each artifacts of development phase to those of other phases.
  - Key to effective testing.
- Quality inspection.

Along with these models, I have discussed a few popular ones, there are a whole set of other ones, big bang, rapid application development, incremental, and the good old waterfall model, which I have not talked to you about in detail. Along with these, there are a whole set of other umbrella activities that are performed as a part of the software development initiative. We discussed about project management, which is initiated in the planning and requirement phase.

So, you manage a set of teams, you know how many teams are there? who is going to report what? who is developing what? how do I put together all this code? how long is it going to take? what is the billing rate? you know lots and lots of project management activities happen, and it is a very important part of the software development life cycle another important part of the software development life cycle.

Another important part of the software development life cycle is documentation, the artifacts that you create as you write code. Code is, of course, the most useful artifact, followed by test cases which you are going to validate your code with, but along with that you also create several other documents. One important such document is what we call the Requirements Traceability Matrix RTM, you can look for it.

What it means is, so we talked about the various phases of software development, planning, requirements, design and architecture, coding, testing, how do you know what is how these faces are related? What do I plan for which part of the design caters to which requirement?

Which part of the code caters to which part of the design, which in turn caters to which part of the requirement? how do I trace these things?

So, that at the end of it, I can say see my code, put together as a piece of software, meets all these requirements, and maybe does not meet a few of them. How do I achieve that? I keep track of what I call a traceability matrix, which is a document, could be even be an excel sheet, that links each of the artifacts that are developed, work products that are developed, as we go through the software development life cycle.

This requirements traceability matrix or document is the key document for making testing, especially in the latter stages a very effective way to find and fix the errors. Along with these, as I told you, there are software quality auditors, quality inspection teams, certification teams, and accreditation teams that you work with, who will certify a software, and say it is ready to go to the market, it is safe and it has been well tested.

So, that also is an integral part of the software development life cycle and software development activity. I mentioned this in the passing as a summary slide because we will not be looking at all of these except maybe for one set on quality inspection, but remember that these are very important part of a software engineering team, alongside software development activities, and they are as integral as software development itself to make sure that the software is successful.

So, the goal of this module was to give you an overview of SDLC, what are the faces of SDLC? and introduce you to some of the models. In the next module, I will introduce you to the testing terminologies that we will be using throughout the course. Thank you.