Session 2

SDLC: it’s a process used by software industry to design, develop and test sw.

Phases of SDLC:

1. Requirements analysis
2. Design analysis
3. Development
4. Testing
5. Maintenance

In Detail:

SDLC is a process used to provide high quality and maintainable product to the customer.

1. Requirement analysis:

Here a plan is made based on the customer requirements it forms a **basic building block**

Of the project. Here various requirements are analyzed functional, non-functional, Technical all this is fulfilled by utilizing **SRS** document, which defines and specifies things that need to be done during entire project.

1. Design analysis:

Using SRS software designers come up with best architecture of software which is specified in Design Document Specification (**DDS**).

Design like Low Level and High-Level Designs are made.

**High Level Design** is the general system design means it refers to the overall system design.

**Low Level Design** is like detailing HLD means it refers to component-level design process.

1. Development

Based on the DDS, Developer uses specific code.

1. Testing

The testing phase is required despite earlier testing because it ensures the **entire system works together** as intended, **validates all requirements**, identifies hidden issues, and allows for **end-user acceptance**. It also focuses on non-functional aspects like performance and security that earlier testing may not cover.

1. Deployment and Maintenance

At the end product is deployed and maintained with timely updates.

Today there are more than 50 SDLC models but none of ‘em is perfect.

The industry has three main components:

3Ps: people, process and product

Waterfall Model:

1. RA (here documentation is imp and prepares SRS [sw requirement specification] )
2. System Design (high n low level models)
3. Implementation (coding)
4. Testing (test cases, test results etc.)
5. Deployment
6. Maintenane

Here each phase depends on the previous phase.

Very old model and not used in industries.

Adv.

1. Quality of Product will be good
2. Since requirements changes aren’t allowed, Chances of finding bugs will be less
3. Initial investment is less cuz other team member are hired late
4. Preferred for small projects, where requirements are freeze.

Disadv.

1. Requirement changes are not allowed
2. An error in one phase could lead to defect in the end product.
3. More investment is required if any change occurs
4. Testing will start only after coding.

Spiral Model:

1. Requirement Analysis
2. Design
3. Coding
4. Testing

At First customers requirements are analyzed and then design is finalized after which development team performs coding and then testing. After doing all these once a cycle is completed a version of software would be delivered to customer then again, the cycle is executed and other version of software are delivered to the customer until the customer acknowledges.   
Using this model more requirements could be added to the software.

It is generally fruitful for Product base companies because of changing Market requirements.

Some notable points:

1. It’s an iterative model also called as **Version control model**
2. Overcomes drawback of Waterfall model
3. Suitable when there is dependency on modules.

Advantages:

1. Testing is done in every cycle
2. Customer has no need to wait long to have an entire product
3. Requirement changes are allowed.

Disadvantages:

1. Requirement changes are allowed but not allowed in between the cycle.
2. Still similar to Waterfall model so more are possible

Prototype Model is a type of blue print of software, It comes somewhere in between waterfall and spiral model.

Each part of a software is called as module i.e. each functionality that we observe is a module of the software

V Model:

Also called as VV model i.e. Verification and Validation model.

The unique specialty is, it provides testing at each phase of SDLC

It having flow like this:

1. BRS/CRS/URS (Business/Customer/User Requirement Specifications)

This will undergo UAT (User Acceptance Testing), such documents which is not understandable to technical people.

1. Then it’s converted to SRS document which consists of Venn diagrams and other graphs which are understandable to the technical team.
2. Based on SRS, designers will prepare HLD & LLD documents

At each phase a Review is carried out by the respective team members using techniques of **Review**, **Walkthrough** and **Inspection** as apart of testing.

This is called as **static testing techniques**

i.e. they are testing the project related documents

1. Coding begins, where multiple modules are built and at the end, they integrate all those modules.
2. Now Testing begins
   1. **Unit** **Testing**

It refers to testing of the **single modules** generally carried out by the developers

* 1. **Integration Testing**

It is the testing carried out by **integrating multiple** modules and this is also carried out by the developers.

These both are called as **White** **Box** Testing techniques.

* 1. **System Testing** it’s carried out by Testers to verify whether the product is working as per customer expectations or not
  2. **User Acceptance Testing** It is carried out by the user along with testers

These both are called as **Black** **Box** Testing techniques

These is called as **Dynamic Testing** which means testing the actual software

Verification v/s Validation

Verification

**Are you building it Right?**

Whatever activities that we are doing at each phase are correct or not.

Done by Developer

It looks, whether the steps/phases carried out are error free

FOCUSED on documentation

Static Testing

Validation

**Have you built the right thing?**

Done by Tester

Look at the end product whether its correct or not, user could validate whether product is built as per the SRS or not

Involves System Testing

FOCUSED on software product

Dynamic Testing

BEFORE SOFTWARE IS READY, WE DO VERIFICATION AND AFTER WE PERFORM VALIDATION

Advantages of V model:

Testing at each phase

Disadvantage:

More Documentation

Initial investment is more

Difference between Integration Testing and System testing:

Sure! Let me break it down further in simpler terms:

**Integration Testing in Software:**

Imagine you are building a **website** that has multiple parts:

* A **login page** for users to sign in.
* A **payment gateway** to process payments.
* A **product catalo** that shows the items users can buy.

These parts (modules) were built separately, so now you need to check if they work well together. This is what **Integration Testing** is all about.

* **Example**: Let's say the **login page** sends a username and password to the server to verify the user. After successful login, the **payment gateway** will handle the transaction. The **product catalog** shows products that users can buy after logging in.

**Integration Testing** makes sure:

* When a user logs in, their data flows correctly to the server.
* The payment gateway receives the correct transaction details.
* The product catalog can show the correct products based on the user’s information.

It's all about making sure **the parts work together** without errors when connected.

**System Testing in Software:**

Now, after all the parts (login, payment, catalog) are integrated into the website, you need to make sure **everything works properly as a whole system**. This is where **System Testing** comes in.

* **Example**: After integrating everything, you would test the entire website, checking:
  + **Does the login work properly**? Can a user successfully sign in?
  + **Can the user browse products** in the catalog smoothly after logging in?
  + **Is the payment gateway working**? Can a user make a successful purchase?
  + **Does the website load fast enough**? Is it secure from potential hackers?

**System Testing** ensures that the entire **website system** works correctly from start to finish, and all parts are **aligned with the requirements**, including:

* **Functionality**: Does everything work as expected?
* **Security**: Is the site safe for users?
* **Performance**: Does the site run fast without errors?
* **User Requirements**: Does the site meet what the users expect (easy to use, fast, functional)?

**In Simple Terms:**

* **Integration Testing** = Checking if the parts of the website (login, payment, catalog) communicate well and work together.
* **System Testing** = Checking if the **whole website** works correctly, smoothly, and securely for the user.

**Why Are These Important?**

* **Integration Testing** helps catch problems between the different parts of the system that might not be visible when testing them individually.
* **System Testing** ensures that, once everything is connected, the full system works as the end user would expect.