Lab 4
Testing

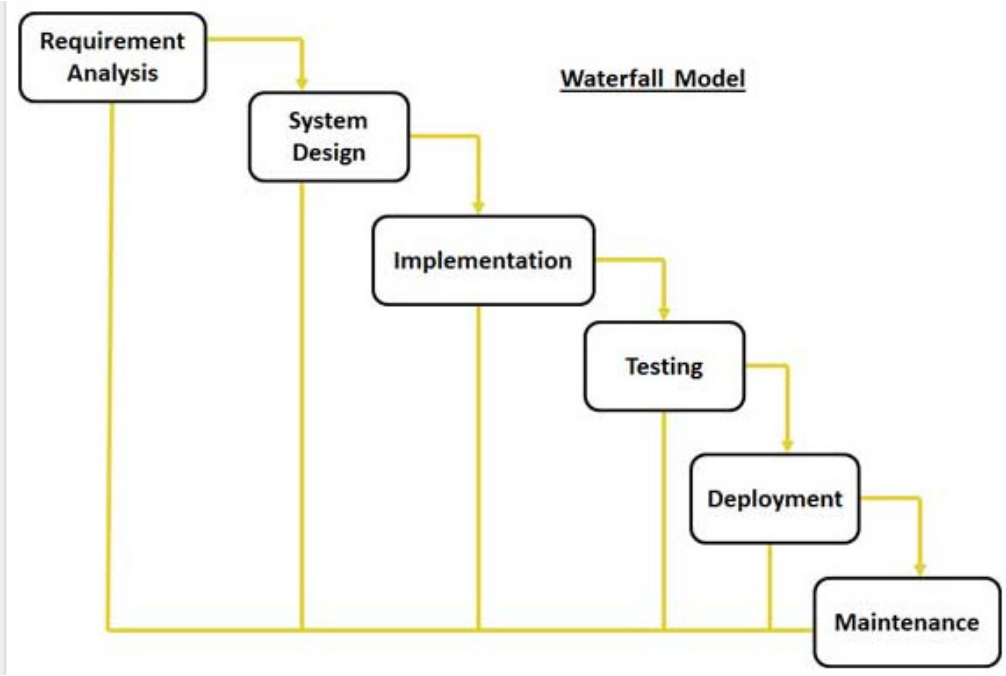
SDLC Models:

There are various software development life cycle models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development.

Following are the most important and popular SDLC models followed in the industry:

1-Waterfall Model design

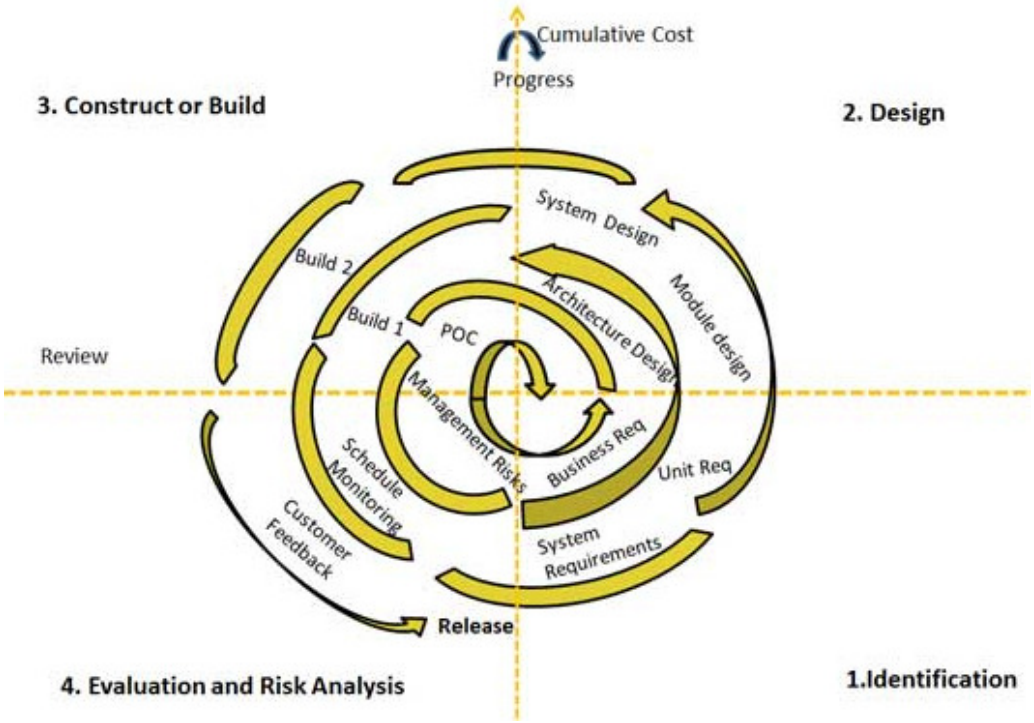
- Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project.
- In "The Waterfall" approach, the whole process of software development is divided into separate phases. Typically, the outcome of one phase acts as the input for the next phase sequentially.
- Waterfall development has distinct goals for each phase of development. Each phase of development proceeds in strict order, without any overlapping or iterative steps.



2-Spiral Model design

- The spiral model combines the idea of iterative\incremental development with the systematic, controlled aspects of the waterfall model.
- In the first quadrant of the spiral model each cycle begins with the identification of objectives for that cycle, the alternatives possible for achieving objectives and the constraints.
- The next step is to evaluate these different alternatives based on objectives and constraints. The evaluation in this step is based on the risk perception of the project.
- The next step is to develop strategies that resolve the risks; this step involves activities like bench marking, simulation etc.
- After this the software is developed keeping in mind the risks and finally next stages are planned.
- The spiral model has four phases:

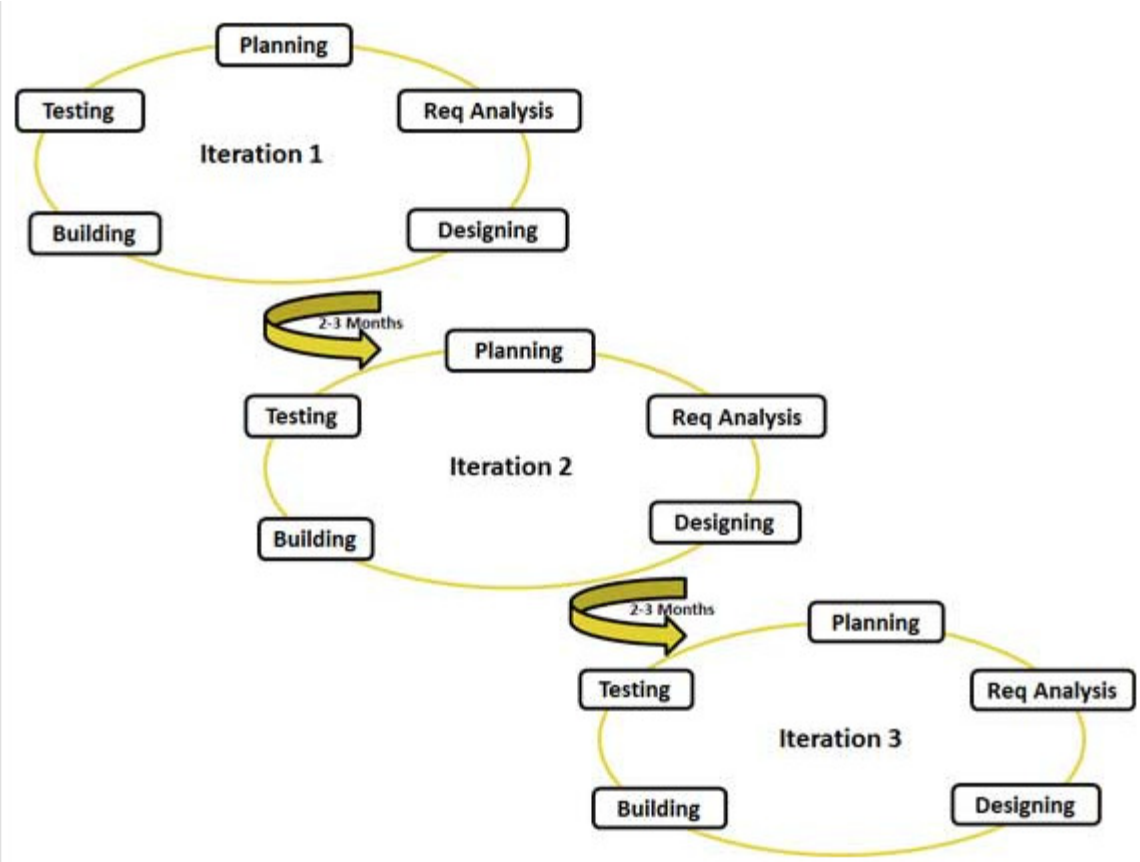
  - **Identification:** This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase. This also includes understanding the system requirements by continuous communication between the customer and the system analyst.
  - **Design:** Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.
  - **Construct or Build:** Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.
  - **Evaluation and Risk Analysis:** Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. At the end of first iteration, the customer evaluates the software and provides feedback.



3-Agile Model design

- The purpose behind developing agile development methodology was to have agility which was missing in traditional waterfall models.
- Agile methodology uses continuous stakeholder feedback to produce high quality consumable code through use cases and a series of short time-boxed iterations.
- In agile, the tasks are divided to time boxes (small periods) to deliver specific features for a release. Each build is incremental in terms of features; the final build holds all the features required by the customer.
- **Agile methodology has four key features:**

  1. Stable code
  2. Continuous stakeholder feedback
  3. Cross functional and Self-directed teams
  4. Sustainable pace



Extreme Programming (XP):

- XP is one of the most popular agile methodologies. XP is a disciplined approach to delivering high-quality software quickly and continuously.
- It promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals, typically every 1-3 weeks.
- Extreme Programming emphasizes teamwork.
- XP is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.
- As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.
- The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels.
- As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed continuously, i.e. the practice of pair programming.

- The original XP recipe is based on five simple values – simplicity, communication, feedback, courage and respect –
- Twelve supporting XP practices (four categories):

1st Fine scale feedback

1) Test-Driven Development(TDD): is a software development process that relies on the repetition of a very short development cycle: **first** the developer writes an (initially failing) automated test case that defines a desired improvement or new function, **then** produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.
2) Planning Game: The main planning process within extreme programming is called the Planning Game. The game is a meeting that occurs once per iteration, typically once a week. The planning process is divided into two parts:
   1-Releasing Planning 2-Iteration Planning
3) Pair Programming: two engineers participate in one development effort at one workstation. Each member performs the action the other is not currently doing: While one types in Unit Tests the other thinks about the class that will satisfy the test, for example. A single, unsubstantiated, unscientific, undergraduate's survey has shown that, after training for the "People Skills" involved, two programmers are more than twice as productive as one for a given task.
4) Whole Team: Within XP, the "customer" is not the one who pays the bill, but the one who really uses the system. XP says that the customer should be on hand at all times and available for questions. For instance, the team developing a financial administration system should include a financial administrator.

2nd Continuous process

5) Continuous Integration: The development team should always be working on the latest version of the software. Since different team members may have versions saved locally with various changes and improvements, they should try to upload their current version to the code repository every few hours, or when a significant break presents itself. Continuous integration will avoid delays later on in the project cycle, caused by integration problems.
6) Design Improvement: functional changes start requiring changes to multiple copies of the same (or similar) code. Another symptom is that changes in one part of the code affect lots of other parts. XP doctrine says that when this occurs, the system is telling you to refactor your code by changing the architecture, making it simpler and more generic.
7) Small Releases: The delivery of the software is done via frequent releases of live functionality creating concrete value. The small releases help the customer to gain confidence in the progress of the project.

3rd Shared understanding

8) Coding Standards: Coding standard is an agreed upon set of rules that the entire development team agree to adhere to throughout the project. The standard specifies a consistent style and format for source code, within the chosen programming language, as well as various programming constructs and patterns that should be avoided in order to reduce the probability of defects.
9) Collective Code Ownership: Collective code ownership means that everyone is responsible for all the code; this, in turn, means that everybody is allowed to change any part of the code.
10) Simple Design: Programmers should take a "simple is best" approach to software design.

11) System Metaphor: The system metaphor is a story that everyone - customers, programmers, and managers - can tell about how the system works. For example a library system may create loan_records(class) for borrowers(class), and if the item were to become overdue it may perform a make_overdue operation on a catalogue (class). For each class or operation the functionality is obvious to the entire team.

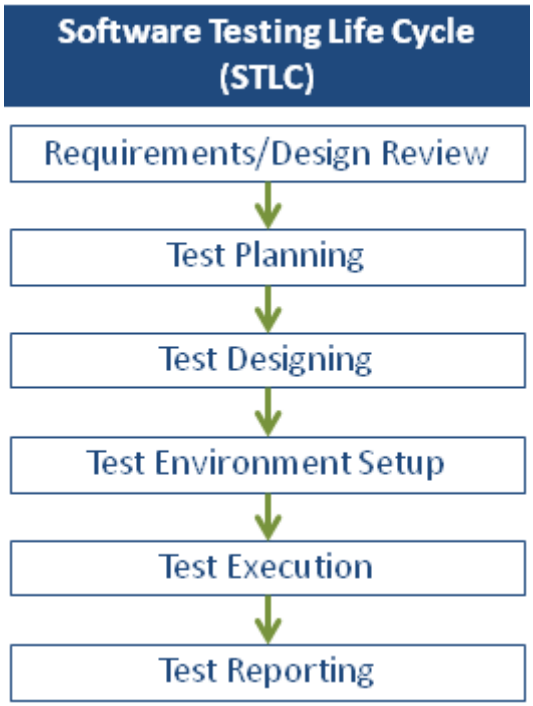
4th Programmer welfare

12) Sustainable Pace: The concept is that programmers or software developers should not work more than 40 hour weeks, and if there is overtime one week, that the next week should not include more overtime. Since the development cycles are short cycles of continuous integration, and full development (release) cycles are more frequent, the projects in XP do not follow the typical crunch time that other projects require (requiring overtime).

All the previous design models are in need to be tested some times per each phase or as a whole system.
The following section will focus in how to test a software and its life cycle.

Software Testing Life Cycle (STLC):

STLC defines the steps/stages/phases in testing of software. Testing software is an integral part of building a system.



1) Requirements Analysis:

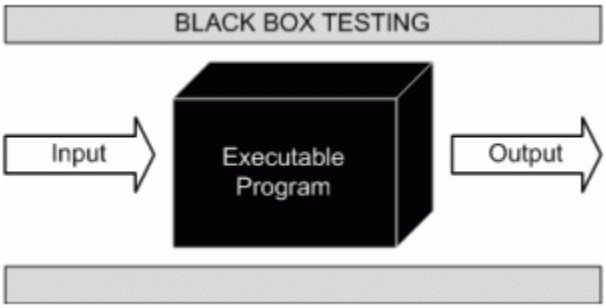
- In this phase, testers analyze the customer requirements and work with developers during the design phase to see which requirements are testable and how they are going to test those requirements.
- It is very important to start testing activities from the requirements phase itself because the cost of fixing defect is very less if it is found in requirements phase rather than in future phases.
- If the software is based on inaccurate requirements, then despite well written code, the software will be unsatisfactory. Most of the defects in a system can be traced back to wrong, missing, vague or incomplete requirements.
- Requirements testing: is testing the requirements whether they are feasible or not. Because a project depends on a number of factors like time, resources, budget etc. Before start working on a project it's important to test the requirement.
  - ✓ Does each requirement have a quality measure that can be used to test whether any solution meets the requirement?
  - ✓ Does the specification contain a definition of the meaning of every essential subject matter term within the specification?
  - ✓ Is every reference to a defined term consistent with its definition?
  - ✓ Is the context of the requirements wide enough to cover everything we need to understand?
  - ✓ Have we asked the stakeholders about conscious, unconscious and undreamed of requirements?
  - ✓ Have we asked the stakeholders about conscious, unconscious and undreamed of requirements?
  - ✓ Is every requirement in the specification relevant to this system?
- **When in SDLC?** In Analysis Phase

2- White box Testing:

- White box testing is the detailed investigation of internal logic and structure of the code.
- White box testing is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing).
- In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.
- The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

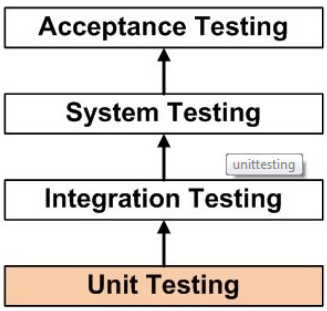
- The tester chooses inputs to exercise paths through the code and determines the appropriate outputs.
- Internal structure/ design/ implementation of the item being tested is known to the tester.
- **EXAMPLE**: A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.
- **When in SDLC?** In Implementation Phase

3- Black box Testing:



- **Black Box Testing** is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.
- Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester
- This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.
- **EXAMPLE**
- A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

4-Unit Testing:



- **Unit Testing** is a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
- A unit is the smallest testable part of software. It usually has one or a few inputs and usually a single output.
- The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use.
- **Method: Unit Testing follows the methodology of while box testing**
- **When:** Unit testing is the first level of testing and is performed prior to Integration Testing.(Implementation Phase)
- **Who?** Unit Testing is normally performed by software developers themselves or their peers. In rare cases it may also be performed by independent software testers.

5-Integration Testing:

- **Integration Testing** is a level of the software testing process where individual units are combined and tested as a group
- It is a different form of testing, in which the interaction between two or more "units" is explicitly tested. Integration tests verify that the components of your application work together. You might make sure that an email was actually sent in an integration test.
- It is the phase in software testing in which individual software modules are combined and tested as a group.
- It occurs after unit testing and before validation testing.
- Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.
- **When?** Integration Testing is performed after Unit Testing and before System Testing.(Implementation Phase)
- **Who?** Either Developers themselves or independent Testers perform Integration Testing.

6- System Testing:

- **System Testing** is a level of the software testing process where a complete, integrated system/software is tested.
- It is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.
- Ensures that application programs written and tested in isolation work properly when integrated into the total system.

- System testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s).
- The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware.
- System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.
- **When?** System Testing is performed after Integration Testing and before Acceptance Testing.(Implementation Phase)
- **Who performs it?** Normally, independent Testers perform System Testing.

7-Systems acceptance test

- **Acceptance Testing** is a level of the software testing process where a system is tested for acceptability.
- A test performed on the final system wherein users conduct a verification, validation, and audit test.
- Uses real data over an extended time period
- Extensive test that addresses: verification testing, validation testing, and audit testing.
- User acceptance is a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This is beta testing of the product & evaluated by the actual end users. The main purpose of this testing is to validate the end-to-end business flow.
- **Method:** use black box testing methodology

8-Functional Testing:

- Testing of the functions of component or system is done. It refers to activities that verify a specific action or function of the code. Functional test tends to answer the questions like "can the user do this" or "does this particular feature work".
- This is a type of black box testing that is based on the specifications of the software that is to be tested.
- It is a way of checking software to ensure that it has all the required functionality that's specified within its functional requirements.
- It is used to verify that a piece of software is providing the same output as required by the end-user or business.
- Functional testing involves evaluating and comparing each software function with the business requirements.
- Some functional testing techniques include smoke testing, white box testing, black box testing, unit testing and user acceptance testing.
- **When in SDLC?** In Implementation Phase

9- Validation Testing:

- The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
- Are we building the right product?

10- Verification Testing:

- The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.
- Are we building the product right?
- The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation."
- Runs the system in a simulated environment using simulated data.
- Alpha testing
- Simulated environment using simulated data
- Checks for errors and omissions regarding end-use and design specifications

11- Web Accessibility Testing:

- Web accessibility testing is a subset of usability testing where the users under consideration have disabilities that affect how they use the web. The end goal, in both usability and accessibility, is to discover how easily people can use a web site and feed that information back into improving future designs and implementations.
- Accessibility testing is the technique of making sure that your product is accessibility compliant. There could be many reasons why your product needs to be accessibility compliant.
- Accessibility testing is a type of systems testing designed to determine whether individuals with disabilities will be able to use the system in question, which could be software, hardware, or some other type of system. Disabilities encompass a wide range of physical problems, including learning disabilities as well as difficulties with sight, hearing and movement.