Module 1

Software Engineering

Introduction to Software Engineering: What is Software Engineering, Characteristics of Software Life cycle of a software system: software design, development, testing, deployment, Maintenance. Project planning phase: project objectives, scope of the software system, empirical estimation models, COCOMO, staffing and personnel planning. Software Engineering models: Predictive software engineering models, model approaches, prerequisites, predictive and adaptive waterfall, waterfall with feedback (Sashimi), incremental waterfall, V model; Prototyping and prototyping models. Software requirements specification, Eliciting Software requirements, Requirement specifications, Software requirements engineering concepts, Requirements modelling, Requirements documentation. Use cases and User stories.

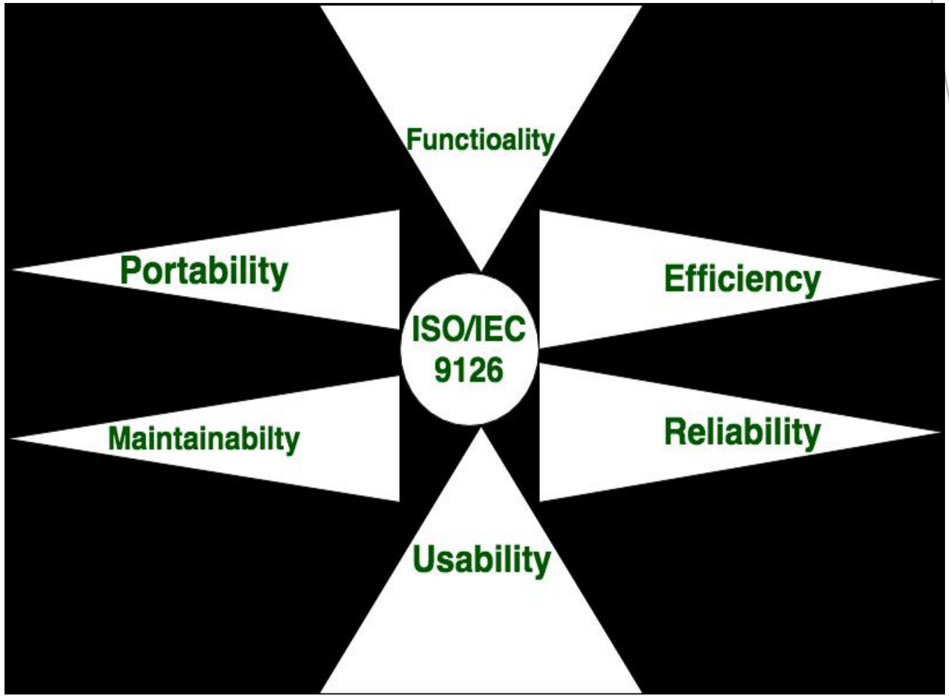
What is Software?

The product that software professionals build and then support over the long term.

Software encompasses: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately store and manipulate information and (3) documentation that describes the operation and use of the programs.

Software is defined as collection of computer programs, procedures, rules and data. Software Characteristics are classified into six major

https://www.geeksforgeeks.org/software-engineering-software-characteristics/

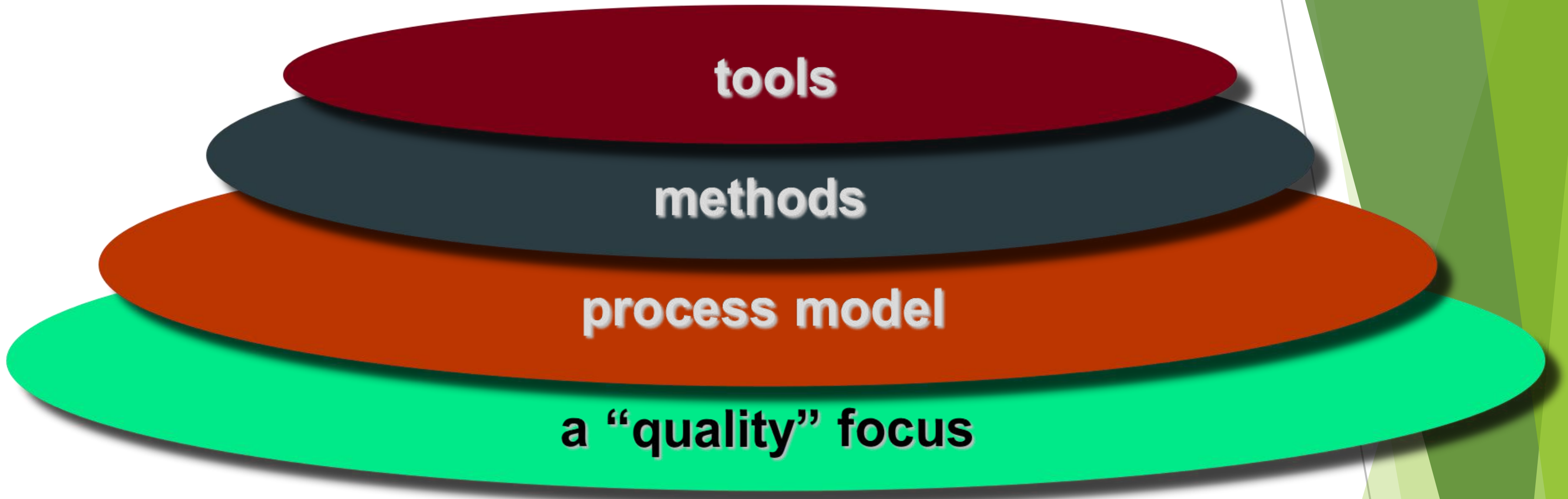


Software engineering is “a systematic approach to the analysis, design, assessment, implementation, test, maintenance and reengineering of software, that is, the application of engineering to software.

Software engineering is a layered technology.

Software Engineering

A Layered Technology



Any engineering approach must rest on organizational commitment to **quality** which fosters a continuous process improvement culture.

Process layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.

Method provides technical how-to’s for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.

Tools provide automated or semi-automated support for the process and methods.

Software Process

- A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is **not a rigid prescription** for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the **appropriate set of work actions** and tasks.
- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by **a specific customer** to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs **more to maintain** than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Features of Software?

- Its characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs.

Software Applications

- 1. System software: such as compilers, editors, file management utilities
- 2. Application software: stand-alone programs for specific needs.
- 3. Engineering/scientific software: Characterized by "number crunching"algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc
- 4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- 5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- 6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- 7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

Software Engineering Definition

The seminal definition:

[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

The IEEE definition:

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).

Importance of Software Engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Five Activities of a Generic Process framework

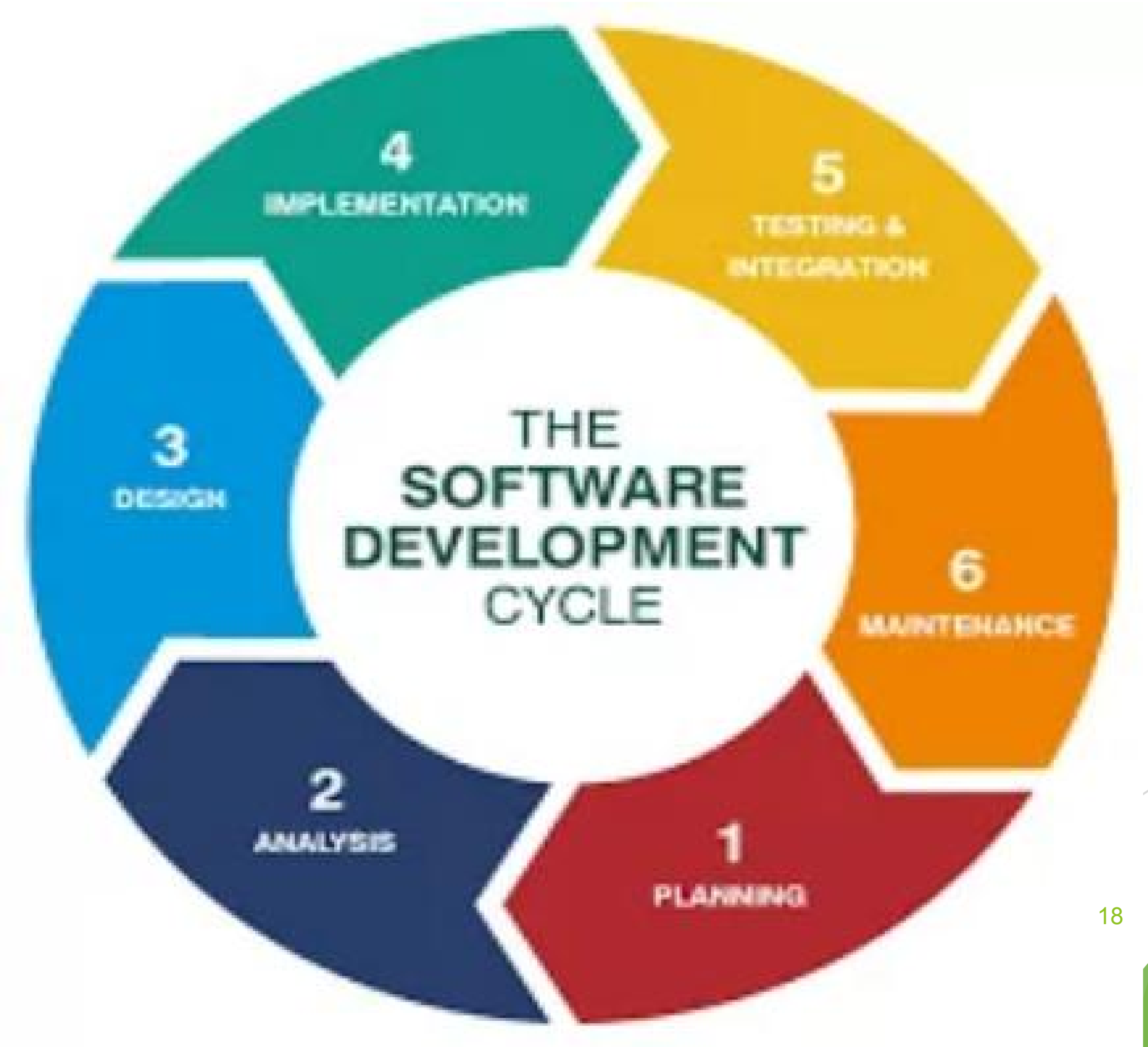
- Communication: communicate with customer to understand objectives and gather requirements
- Planning: creates a “map” defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a “sketch”, what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.

- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.
- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

SDLC

- Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying and maintaining the software.
- SDLC defines the complete cycle of development i.e. all the tasks involved in planning, creating, testing, and deploying a Software Product.

Characteristics of Software
Life cycle of a software system



- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

Project Planning Phase

- The **Project Planning Phase** is the second phase in the *project life cycle*. It involves creating of a set of plans to help guide your team through the execution and closure phases of the project.
- The plans created during this phase will help you to manage time, cost, quality, change, risk and issues. They will also help you manage staff and external suppliers, to ensure that you deliver the project on time and within budget.
- Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

- It defines the roles and responsibilities of the project management team members.
- It ensures that the project management team works according to the business objectives.
- It checks feasibility of the schedule and user requirements.
- It determines project constraints.

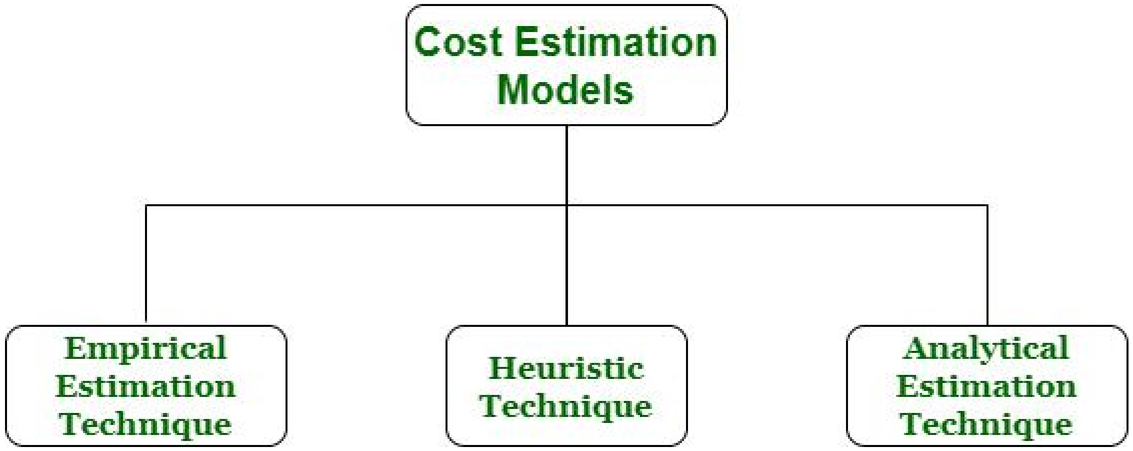
Project Objective

- A project objective describes the desired results of a project, which often includes a tangible item.
- A project may have one objective, many parallel objectives, or several objectives that must be achieved sequentially. To produce the most benefit, objectives must be defined early in the project life cycle, in phase two, the planning phase.
- The objective of software project planning is to provide a framework that enables the project manager to make some reasonable estimates of resources, cost and schedule
- These estimates are made at the beginning of a software project and should be updated regularly as the project progresses towards completion
- The planning objective is achieved through a process of information discovery that leads to the formulation of reasonable estimates

Scope of the Software System

- First activity in the planning of the software project is the determination of the scope of the software
- Scope of the software describes the data and control to be processed, function, performance, constraints, interfaces and reliability
- Feasibility

Cost Estimation Models


Cost Estimation Models
Empirical Estimation Technique
Heuristic Technique
Analytical Estimation Technique

Empirical estimation models

- Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step.

Cocomo (Constructive Cost Model)

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**.

It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule.

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

- COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

- Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
- Determine a set of 15 multiplying factors from various attributes of the project.
- Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.
- The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below.
- $\mathbf{E_i=a*(KDLOC)b}$

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types

- Organic
- Semidetached
- Embedded

Organic - A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

Semi-detached - A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity.

Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type

- **Embedded -** A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

According to Boehm, software cost estimation should be done through three stages:

- Basic Model
- Intermediate Model
- Detailed Model

Any of the three forms can be adopted according to our requirements.

The first level, **Basic COCOMO** can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Effort=a_1*(KLOC) a_2 PM

Tdev=b_1*(efforts)b_2 Months

Person Required = Effort/Tdev

Where

- **KLOC** is the estimated size of the software product indicate in Kilo Lines of Code,
- a_1,a_2,b_1,b_2 are constants for each group of software products,
- **Tdev** is the estimated time to develop the software, expressed in months,
- **Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a_1*(KLOC) a_2 PM
Tdev=b_1*(efforts)b_2 Months
Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM
D = 2.5 * (1295.31)0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400)1.12=2462.79 PM
D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM
D = 2.5 * (4772.8)0.32 = 38 PM

Personnel Planning

- Personnel Planning deals with staffing. Staffing deals with the appoint personnel for the position that is identified by the organizational structure.

It involves:

- Defining requirement for personnel
- Recruiting (identifying, interviewing, and selecting candidates)
- Compensating
- Developing and promoting agent
- At planning time, when the system method has not been completed, the planner can only think to know about the large subsystems in the system and possibly the major modules in these subsystems.

- Once the project plan is estimated, and the effort and schedule of various phases and functions are known, staff requirements can be achieved.
- From the cost and overall duration of the projects, the average staff size for the projects can be determined by dividing the total efforts (in person-months) by the whole project duration (in months).
- Typically the staff required for the project is small during requirement and design, the maximum during implementation and testing, and drops again during the last stage of integration and testing.
- Using the COCOMO model, average staff requirement for various phases can be calculated as the effort and schedule for each method are known.
- When the schedule and average staff level for every action are well-known, the overall personnel allocation for the project can be planned.

Software Engineering models:

Predictive software engineering models

Predictive Modeling is a mathematical approach to build **models** based on the existing data, which helps in finding the future value or trend of a variable. It involves very heavy mathematical and statistical analysis to create such **models**.

Predictive models are useful primarily because they give a lot of structure to a project. Some **predictive software** development **models** such as waterfall **model**, waterfall with feedback, sashimi **model**, incremental waterfall **model**, V-**model** and **software** development life cycle

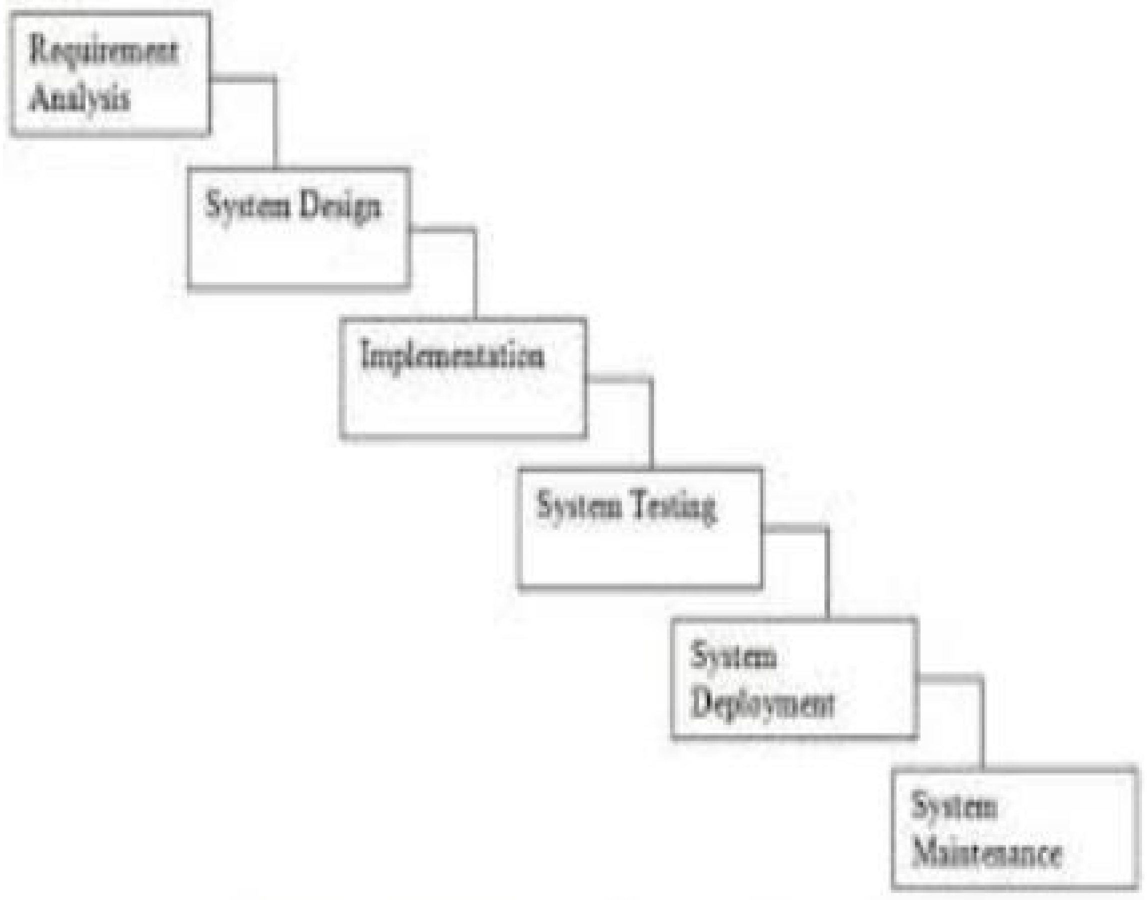
Hw:

Predictive vs Adaptive software models

Waterfall Model

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks

In this method, the whole process of software development is divided into various phases.

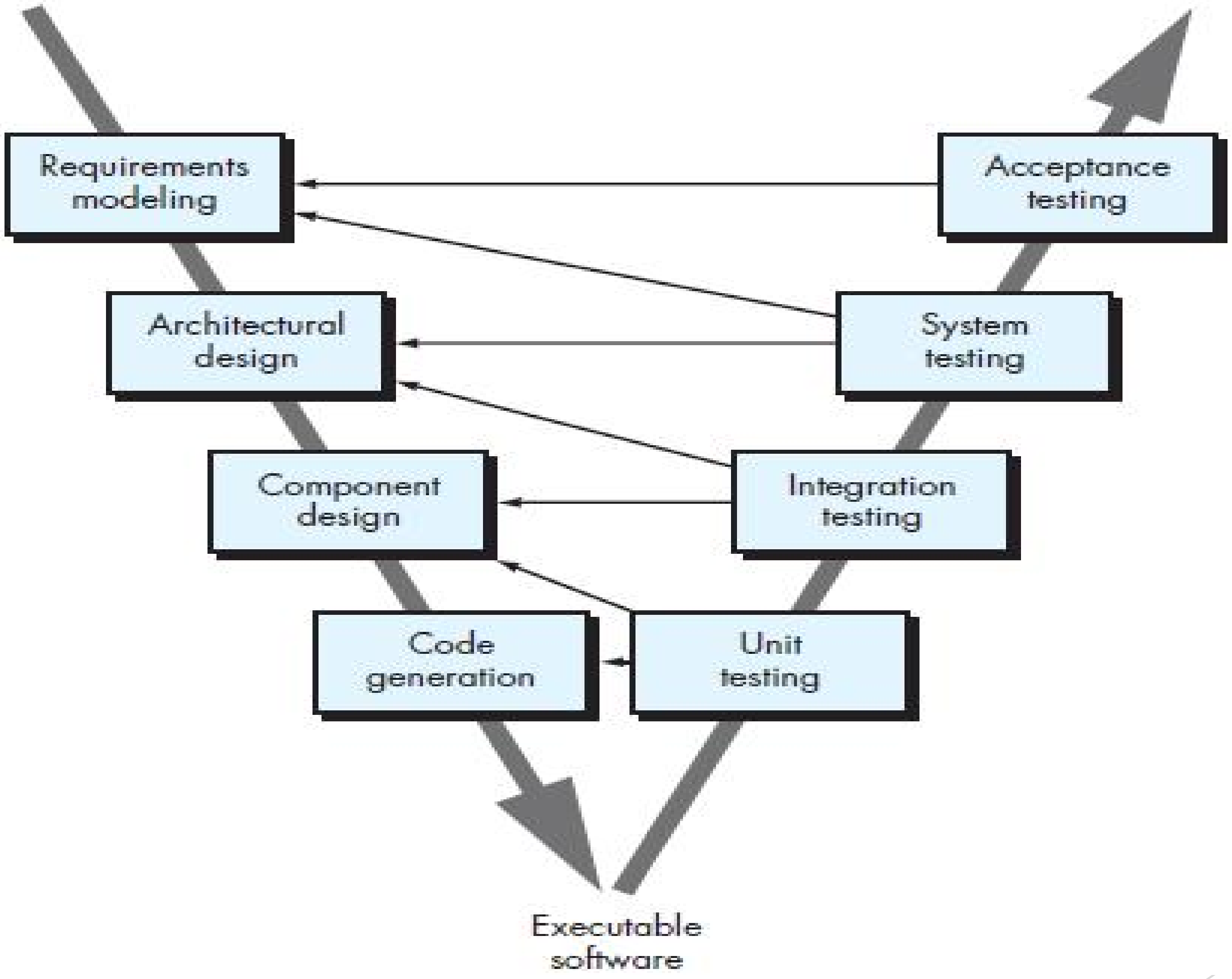


Waterfall Model - © www.SoftwareTestingHelp.com

When to use SDLC Waterfall Model

- Waterfall model can be used when Requirements are not changing frequently
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

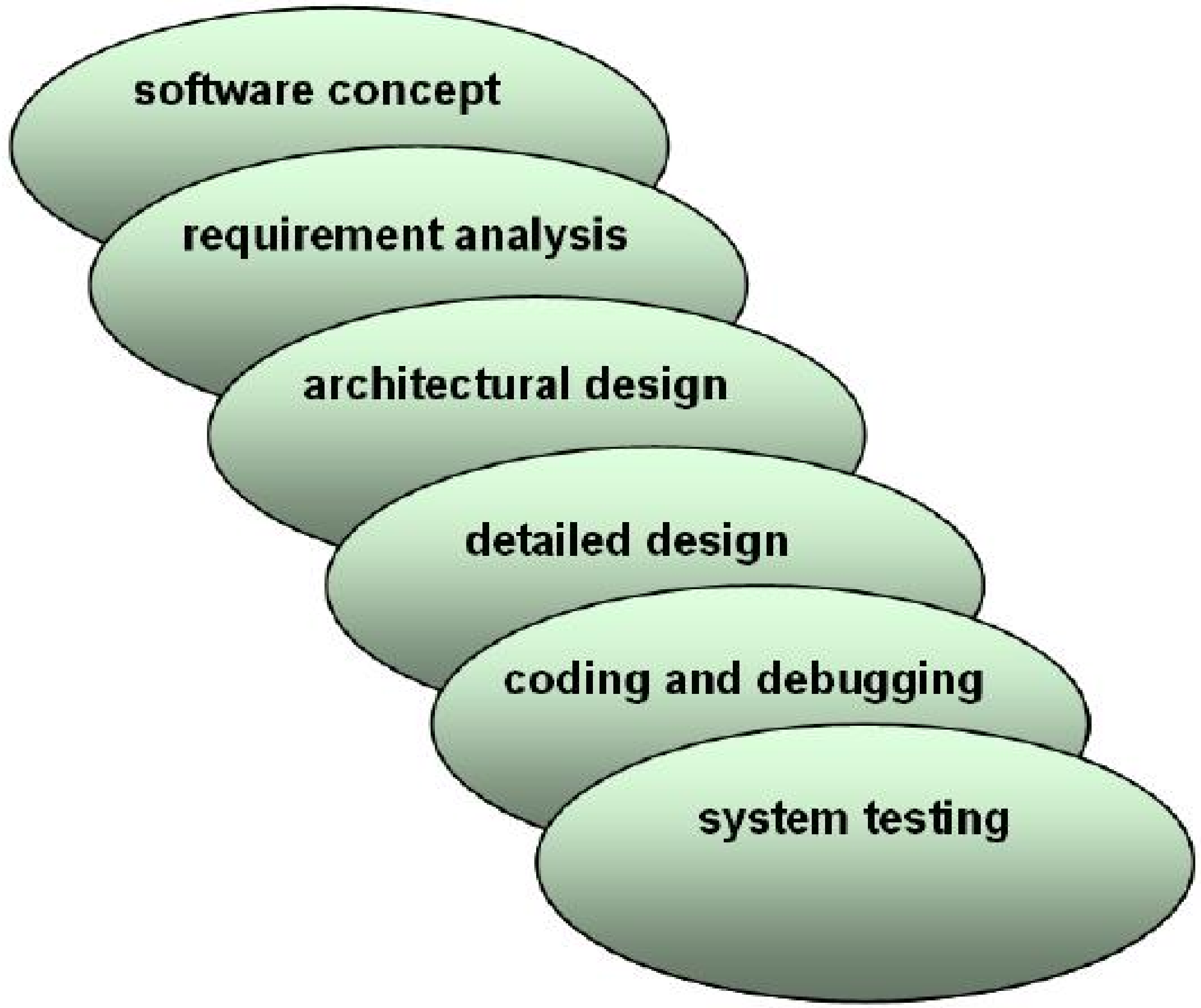
- The *waterfall model*, sometimes called the *classic life cycle* , suggests a systematic, sequential approach 2 to software development that begins with customer
- specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.
- A variation in the representation of the waterfall model is called the *V-model*.


Requirements modeling
Architectural design
Component design
Code generation
Acceptance testing
System testing
Integration testing
Unit testing
Executable software

- V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.
- As a software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution.
- Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moves down the left side

The Sashimi Model

It is sometimes referred to as the "waterfall model with overlapping phases" or "the waterfall model with feedback". Since phases in the sashimi model overlap, information of problem spots can be acted upon during phases that would typically, in the pure waterfall model


software concept
requirement analysis
architectural design
detailed design
coding and debugging
system testing

The Spiral Model

- The Spiral Life Cycle Model is a type of iterative software development model which is generally implemented in high risk projects.
- It was first proposed by Boehm. In this system development method, we combine the features of both, waterfall model and prototype model.
- In Spiral model we can arrange all the activities in the form of a spiral.
- Each loop in a spiral represents a development phase (and we can have any number of loops according to the project). Each loop has four sections or quadrants:
- 1. To determine the objectives, alternatives and constraints. We try to understand the product objectives, alternatives in design and constraints imposed because of cost, technology, schedule, etc.

2. Risk analysis and evaluation of alternatives.
3. Execution of that phase of development
4. Planning the next phase

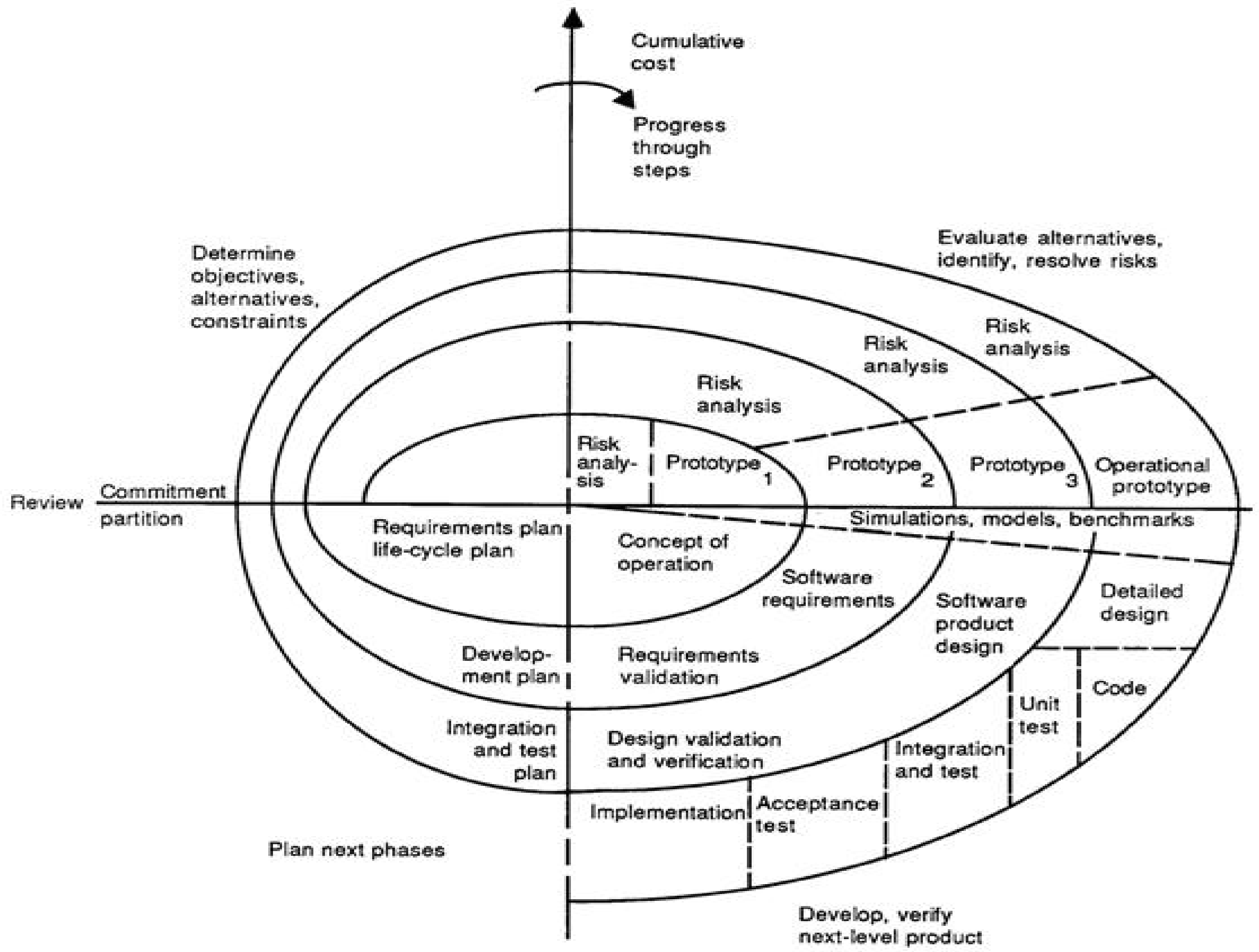
Why spiral model is called Meta model?

Spiral model is also called as meta-model because in a way it comprises of other models of SDLC. Both waterfall and prototype models are used in it. Here we do software development systematically over the loops (adhering to waterfall approach) and at the same time we make a prototype and show it to user after completion of various phases (just in case of prototype model). This way we are able to reduce risks as well as follow systematic approach.

- Advantages of Spiral Model
- 1) Spiral Life Cycle Model is one of the most flexible SDLC models in place. Development phases can be determined by the project manager, according to the complexity of the project.
- 2) Project monitoring is very easy and effective. Each phase, as well as each loop, requires a review from concerned people. This makes the model more transparent.
- 3) Risk management is one of the in-built features of the model, which makes it extra attractive compared to other models.
- 4) Changes can be introduced later in the life cycle as well. And coping with these changes isn't a very big headache for the project manager.
- 5) Project estimates in terms of schedule, cost etc become more and more realistic as the project moves forward and loops in spiral get completed.
- 6) It is suitable for high risk projects, where business needs may be unstable.
- 7) A highly customized product can be developed using this.

- Disadvantages of Spiral Model
- 1) Cost involved in this model is usually high.
- 2) It is a complicated approach especially for projects with a clear SRS.
- 3) Skills required, evaluating and reviewing project from time to time, need expertise.
- 4) Rules and protocols should be followed properly to effectively implement this model. Doing so, through-out the span of project is tough.

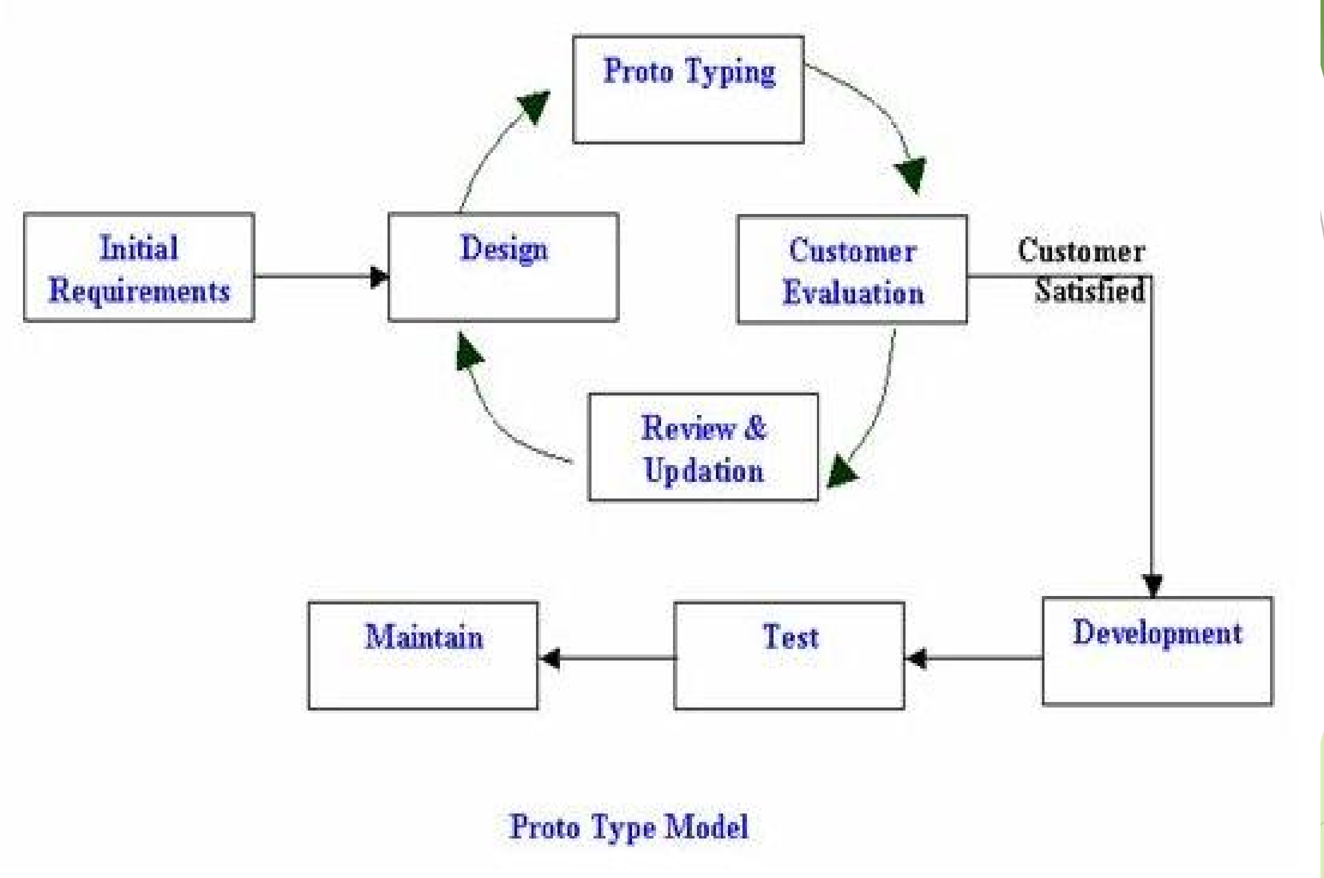
- 5) Due to various customizations allowed from the client, using the same prototype in other projects, in future, is difficult.
- 6) It is not suitable for low risk projects.
- 7) Meeting budgetary and scheduling requirements is tough if this development process is followed.
- 8) Amount of documentation required in intermediate stages makes management of project very complex affair.


Cumulative cost
Progress through steps
Determine objectives, alternatives, constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analy- sis
Prototype 1
Prototype 2
Prototype 3
Operational prototype
Review
Commitment partition
Simulations, models, benchmarks
Requirements plan life-cycle plan
Concept of operation
Software requirements
Software product design
Detailed design
Development plan
Requirements validation
Code
Unit test
Integration and test plan
Design validation and verification
Integration and test
Acceptance test
Implementation
Plan next phases
Develop, verify next-level product

Prototyping Model

- Prototyping model was developed to counter the limitations of waterfall model. The basic idea behind prototyping model is that instead of freezing the requirements before any design or coding can begin, a throwaway prototype is built to understand the requirements. This prototype is build based on currently known requirements.
- The throwaway prototype undergoes design, coding, testing but each of these phases are not formal. The prototype is developed and delivered to client; client uses this prototype and gets the actual feel of the system.
- Client interacts with the system and gets better understanding of the requirements of the desired system; this at last results in more stable requirements from clients.

- Prototyping model is mostly used for projects where the requirements are not very clear from client. When a prototype is developed using known unclear requirements and given to customer he uses the prototype and gets the feel of the software system and then produces concrete requirements.
- In prototyping model the focus of the development is to include those features which are not properly understood as prototype is anyway to be discarded. So the well-known requirements are not implemented in the prototype.



Proto Type Model

Comparison between Adaptive model and Predictive model

Parameters	Predictive	Adaptive
Phases	Overlapping and sequential	Parallel, sequential and overlapping
High-level planning	Yes	Yes
High-level scope	Yes	Yes
Detailed Planning	At the starting stage of the project.	Only of iteration
When to use	It is used only when the product is understood appropriately.	It is used when the project is not understood appropriately, rapidly changing environments.
Involvement of customer	While changing the scope and in beginning	Continuous

Incremental Model

- Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

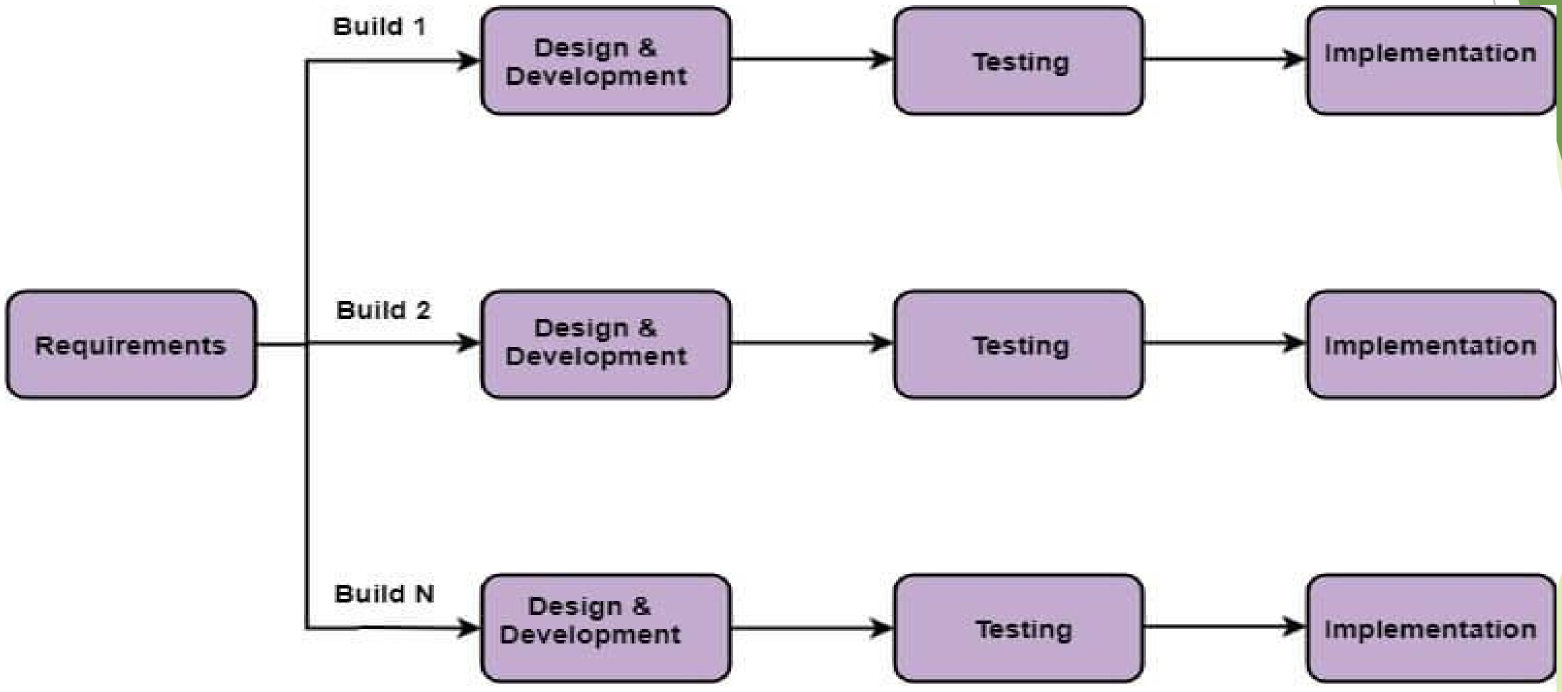


Fig: Incremental Model

- **1. Requirement analysis:** In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.
- **2. Design & Development:** In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.
- **3. Testing:** In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.
- **4. Implementation:** Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

What is Software Requirement Specification

- A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.
- **Software Requirement Specification (SRS) Format** as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-requirements depending upon type of requirement. The interaction between different customers and contractor is done because its necessary to fully understand needs of customers.

Contd..

- Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer’s demand.
- A structure of good SRS. These are as follow

1. Introduction

(i) Purpose of this document

(ii) Scope of this document

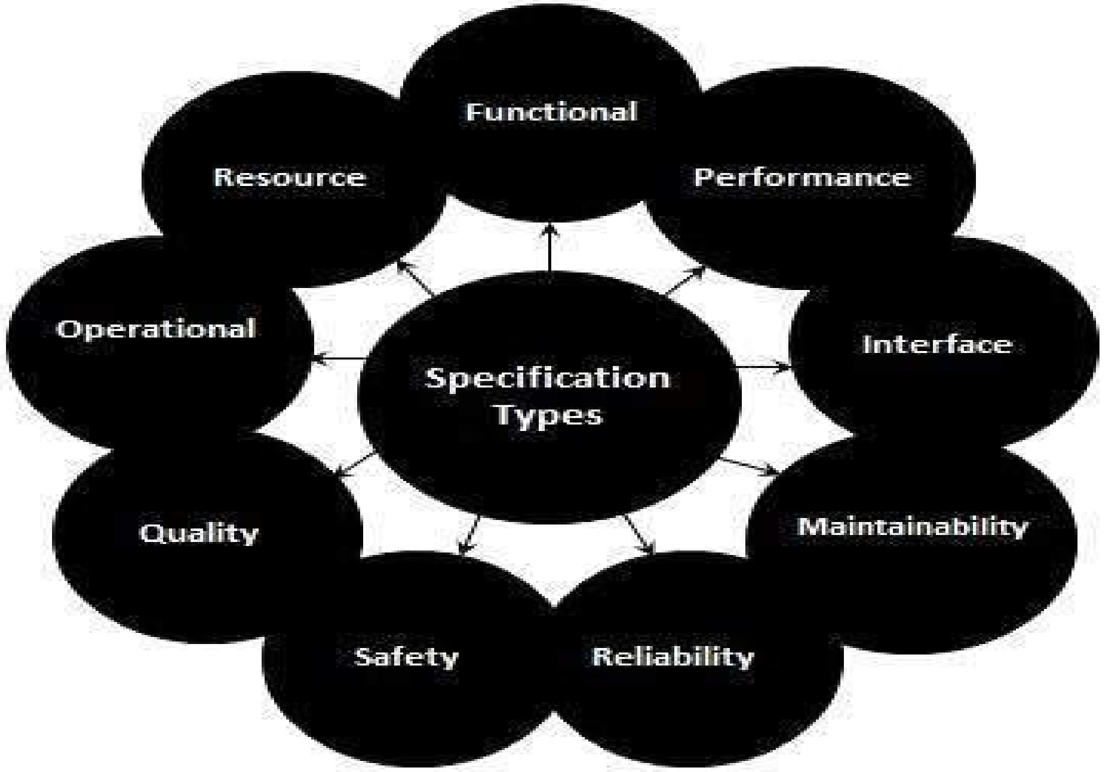
(iii) Overview

2. General description
3. Functional Requirements
4. Interface Requirements
5. Performance Requirements
6. Design Constraints
7. Non-Functional Attributes
8. Preliminary Schedule and Budget
9. Appendices

Qualities of SRS

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Types of Requirements



The diagram depicts the various types of requirements that are captured during SRS

Eliciting Software requirements

- In requirements engineering, **requirements elicitation** is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. The practice is also sometimes referred to as "**requirement gathering**".
- **Requirements elicitation** is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

There are a number of requirements elicitation methods. Few of them are listed below :

- Interviews
- Brainstorming Sessions
- Facilitated Application Specification Technique (FAST)
- Quality Function Deployment (QFD)
- Use Case Approach

Interviews:

- Objective of conducting an interview is to understand the customer's expectations from the software.
- It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
- Interviews maybe be open ended or structured.
- In open ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
- In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally a document is prepared which consists of the list of requirements and their priority if possible.

Facilitated Application Specification Technique:
It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

- A team oriented approach is developed for requirements gathering.
  Each attendee is asked to make a list of objects that are-
- Part of the environment that surrounds the system
- Produced by the system
- Used by the system
- Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

Quality Function Deployment:
In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.
3 types of requirements are identified –

- **Normal requirements -** In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements -** These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements -** It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes

Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence they only give a functional view of the system.

The components of the use case design includes three major things – Actor, Use cases, use case diagram

- **Actor -** It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.

Primary actors – It requires assistance from the system to achieve a goal.

Secondary actor – It is an actor from which the system needs assistance

Use cases - They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.

Requirements Engineering Concepts

- **What is software requirements engineering?**
- Requirements engineering is a subdiscipline of software engineering that is concerned with determining the goals, functions, and constraints of software systems. Requirements engineering also involves the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families.
- Requirements engineering is the process of eliciting, documenting, analyzing, validating, and managing requirements.
- Requirements modeling involves the techniques needed to express requirements in a way that can capture user needs.
- The requirements engineering process begins with a feasibility study activity, which leads to a feasibility report.

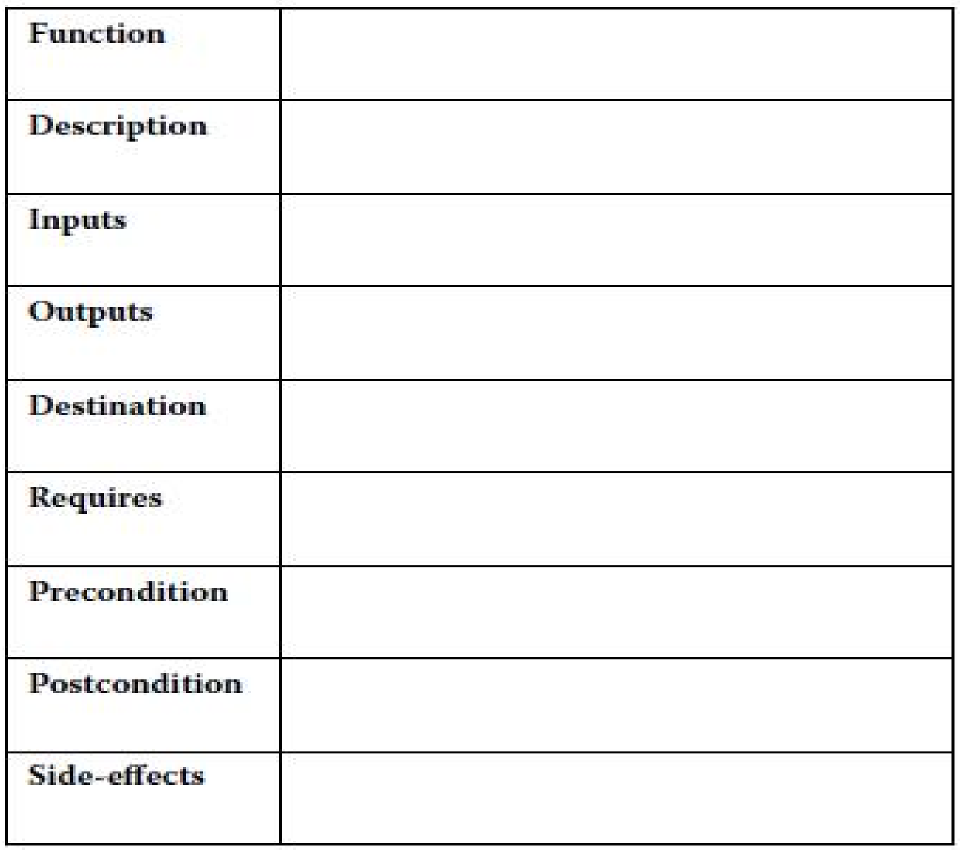
Requirements Modeling

- There are a number of ways to model software requirements.
- It include natural languages, informal and semiformal techniques, user stories, use case diagrams, structured diagrams, object-oriented techniques, formal methods, etc..
- Every clear SRS must have a great deal of narrative in clear and concise natural language.
- In case of expressing complex behavior, it is best to use formal or semiformal methods, clear diagrams or tables, and narrative as needed to tie these elements together.

Alternatives to using natural languages

- structured natural language
- design description languages
- graphical notations
- mathematical specifications
- Structured languages remove some of the problems resulting from ambiguity and flexibility and impose a degree of uniformity on a specification.
- On the other hand, use of structured languages requires a level of training that can frustrate stakeholders. To increase usability, structured languages can be facilitated using a forms-based approach

A form based specification Template

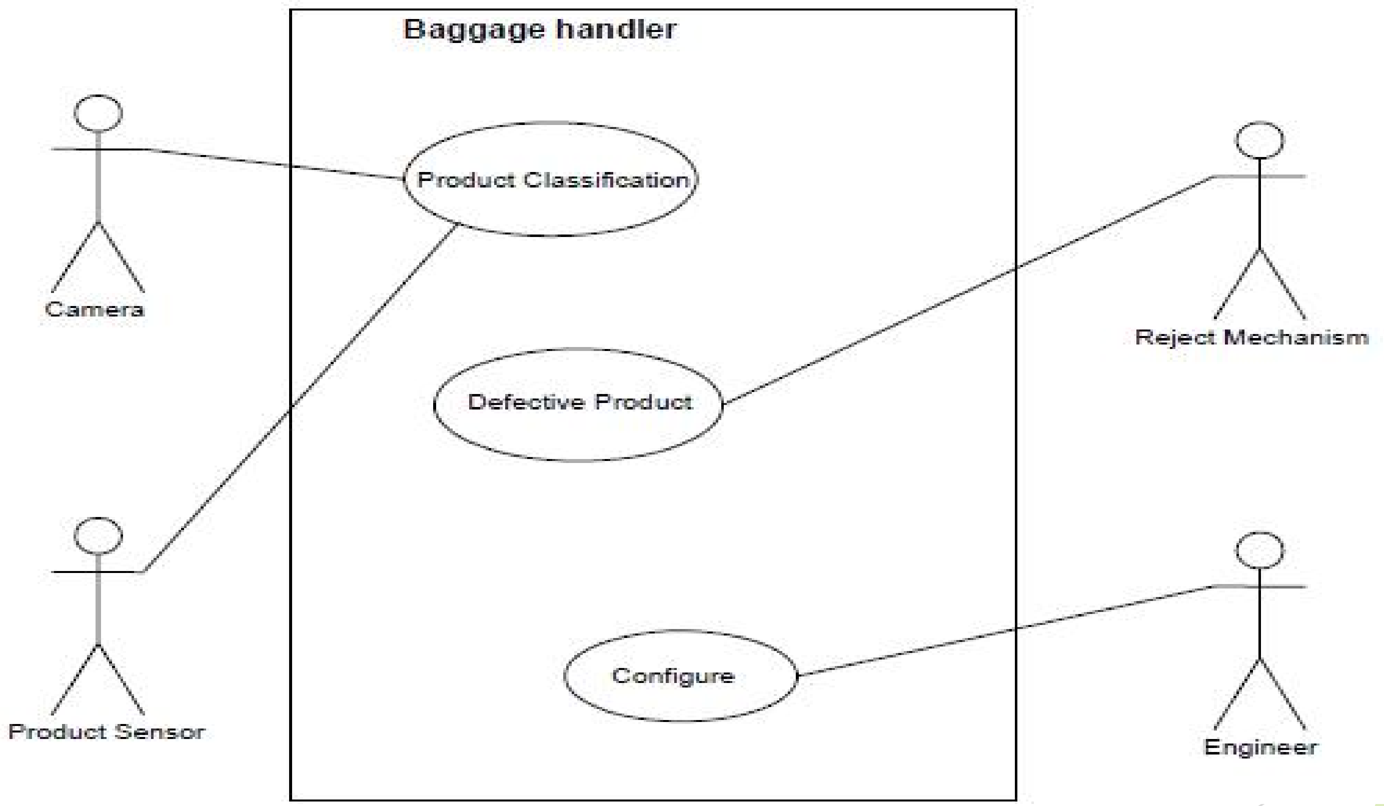


Function	
Description	
Inputs	
Outputs	
Destination	
Requires	
Precondition	
Postcondition	
Side-effects	

What are use cases?

- Use cases are an essential artifact in object-oriented requirements elicitation and analysis and are described graphically using any of several techniques.
- One representation for the use case is the use case diagram, which depicts the interactions of the software system with its external environment.

Use case diagram of the baggage inspection system



What are user stories?

- User stories are short conversational texts that are used for initial requirements discovery and project planning.
- User stories are widely used in conjunction with agile methodologies.
- User stories are written by the customers in their own “voice,” in terms of what the system needs to do for them.
- User stories usually consist of two to four sentences written by the customer in his own terminology.

Eg:

There are two pumps in the wet well control system. The control system should start the pumps to prevent the well from overflowing. The control system should stop the pumps before the well runs dry.

References

1. Philip A. Laplante, What Every Engineer Should Know about Software Engineering, CRC Press.
2. Roger S Pressman
3. https://www.javatpoint.com/software-engineering-data-flow-diagrams
4. https://www.agilebusiness.org/page/ProjectFramework_15_RequirementsandUserStories
5. https://www.geeksforgeeks.org/software-engineering-requirements-elicitation/
6. https://slideplayer.com/slide/12177621/
7. https://complextester.wordpress.com/2012/08/07/spiral-prototype-fish-bone-sashimi/
8. https://www.javatpoint.com/software-engineering-incremental-model
9. https://www.tutorialspoint.com/difference-between-incremental-model-and-waterfall-model