Software Engineering

Modules

1. Introduction To Software Engineering and Process Models
2. Software Requirements Analysis and Modelling
3. Software Estimation Metrics
4. Software Design
5. Software Testing
6. Software Configuration Management, Quality Assurance and Maintenance

Software Engineering

Software engineering is a systematic and disciplined approach towards the development of the software operation and maintenance. Software engineering is an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures.

Characteristics

- Software does not wear out i.e. it does not lose the material.
- Software should be inherently complex.
- Software must be efficient
- Software must be integral

Umbrella Activity

- Software project tracking and control: This activity allows the software team to check the progress of software development.
- Risk management: Risk management is a series of steps to help software development teams understand and manage uncertainty.
- Software quality assurance: As its name suggest this defines and conducts the activities required to ensure software quality.
- Technical reviews: It assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity
- Measurement: This includes all measurements of all aspects of the software project.
- Software configuration management: It manages the impact of changes throughout the software development process

- Reusability management: Define the standards for the reuse of work products (including software components), and develop mechanisms to implement reusable components.
- Work product preparation and production: It encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Generic Framework

1. Communication: The software development starts with the communication between customer and developer.

2. Planning: It consists of complete estimation, scheduling for project development and tracking.

3. Modelling:
   i. Modelling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
   ii. The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

4. Construction:
   i. Construction consists of code generation and the testing part.
   ii. Coding part implements the design details using an appropriate programming language.
   iii. Testing is to check whether the flow of coding is correct or not.
   iv. Testing also check that the program provides desired output.

5. Deployment:
   i. Deployment step consists of delivering the product to the customer and take feedback from them.
   ii. If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Capability maturity model

- The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.
- It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.

Levels

- Level One: Initial - The software process is characterized as inconsistent, and occasionally even chaotic.
- Level Two: Repeatable - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality.
- Level Three: Defined - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization.
- Level Four: Managed - Management can effectively control the software development effort using precise measurements
- Level Five: Optimizing - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements.

The Waterfall Model

- The waterfall model is also called as 'Linear sequential model' or 'Classic life cycle model
- In this model, each phase is fully completed before the beginning of the next phase. This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.

V-model

The V - model is SDLC model where execution of processes happens in a sequential manner in Vshape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

Incremental Process model

1. The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
2. The first increment in this model is generally a core product.
3. Each increment builds the product and submits it to the customer for any suggested modifications.
4. The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
5. This process is repeated until the product is finished

RAD model

1. RAD is a Rapid Application Development model.
2. Using the RAD model, software product is developed in a short period of time.
3. The initial activity starts with the communication between customer and developer.
4. Planning depends upon the initial requirements and then the requirements are divided into groups.
5. Planning is more important to work together on different modules.

The Spiral model

- Spiral model is a risk driven process model.
- It is used for generating the software projects.
- In spiral model, an alternate solution is provided if the risk is found in the risk analysis, then alternate solutions are suggested and implemented.
- It is a combination of prototype and sequential model or waterfall model.
- In one iteration all activities are done, for large project's the output is small

The concurrent development model

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under-development state into the awaiting change state.

Agile Model

- Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into small incremental builds.
- These builds are provided in iterations.
- Each iteration typically lasts from about one to three weeks.

Extreme Programming (XP)

- Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team.
- XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.
- It is used to improve software quality and responsive to customer requirements.

Scrum

- Scrum is an agile process most commonly used for product development, especially software development.
- Scrum is a project management framework that is applicable to any project with aggressive deadlines, complex requirements and a degree of uniqueness.
- In Scrum, projects move forward via a series of iterations called sprints. Each sprint is typically two to four weeks long.

Kanban

- Kanban is a popular framework used to implement agile software development.
- It requires real-time communication of capacity and full transparency of work.
- Work items are represented visually on a Kanban board, allowing team members to see the state of every piece of work at any time.
- Kanban can be used in any knowledge work setting, and is particularly applicable in situations where work arrives in an unpredictable fashion and/or when you want to deploy work as soon as it is ready, rather than waiting for other work items.

Data Flow Diagram

- A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects.
- Often it is a preliminary step used to create an overview of the system that can later be elaborated.
- DFDs can also be used for the visualization of data processing (structured design) and show what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored.
- It does not show information about the timing of processes or information about whether processes will operate in sequence or in parallel.

Design concepts

1. Abstraction

   A collection of data that describes a data object is a data abstraction.

2. Architecture

   The complete structure of the software is known as software architecture. Structure provides conceptual integrity for a system in a number of ways.

3. Patterns

   A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

   A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem

requirements. Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.

The functional independence is accessed using two criteria i.e. Cohesion and coupling.

- Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

- Cohesion is the indication of the relationship within module.
- Cohesion shows the module's relative functional strength.
- Cohesion is a degree (quality) to which a component / module focuses on the single thing.
- While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.
- Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.
- Cohesion is Intra — Module Concept.

- Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program

- Coupling is the indication of the relationships between modules.
- Coupling shows the relative independence among the modules.
- Coupling is a degree to which a component / module is connected to the other modules.
- While designing you should strive for low coupling i.e. dependency between modules should be less
- Making private fields, private methods and non public classes provides loose coupling.
- Coupling is Inter -Module Concept.

7. Refinement

Refinement is a top-down design approach. It is a process of elaboration.

8. Refactoring

It is a reorganization technique which simplifies the design of components without changing its function behavior.

9. Design classes

The model of software is defined as a set of design classes. Every class describes the elements of problem domain and that focus on features of the problem which are user visible

10. Separation of Data

Known as separation of concerns, this principle states that the software code must be separated into two sections called layers and components.

11. Data Hiding

Also known as information hiding, data hiding allows modules to pass only the required information between themselves without sharing the internal structures and processing.

Cohesion	Coupling
Cohesion measures, about how much the functionality are related to each Other within the module.	Coupling measures, how much one module is dependent on the other program modules within the whole application.
Strong cohesion provides the best software.	Loose coupling provides the best software.
In a good design, the various component parts (e.g. the classes) have high cohesion.	In a good design, the various component parts (e.g. the classes) have low coupling.
For example: making all members visible within the class provide high cohesion.	For example : Having private fields, non-public classes and private methods provide loose-coupling.

Unit Testing

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself. It is correlated with the functional correctness of the independent modules. Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

System Testing

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested

BLACK-BOX TESTING

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon

White box:

White Box Testing is the testing of a software solution's internal coding and infrastructure. It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White box testing is also known as clear box testing, open box testing, logic driven testing or path driven testing or structural testing and glass box testing. Using white-box testing methods, the software engineer can derive test cases that (1) guarantee that all independent paths within a module have been exercised at least once, (2) exercise all logical decisions on their true and false sides, (3) execute all loops at their boundaries and within their operational bounds, and (4) exercise internal data structures to ensure their validity.

SRS

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-functional depending upon type of requirement. The interaction between different customers and contractor is done because its necessary to fully understand needs of customers.

1. Introduction
    - Purpose of this document
    - Scope of this document
    - 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

Function Point

Function Point Analysis (FPA) is a method or set of rules of Functional Size Measurement. It assesses the functionality delivered to its users, based on the user's external view of the functional requirements. It measures the logical view of an application, not the physically implemented view or the internal technical view.

The Function Point Analysis technique is used to analyze the functionality delivered by software and Unadjusted Function Point (UFP) is the unit of measurement.

Objectives of FPA:

- The objective of FPA is to measure the functionality that the user requests and receives.
- The objective of FPA is to measure software development and maintenance independently of the technology used for implementation.
- It should be simple enough to minimize the overhead of the measurement process.
- It should be a consistent measure among various projects and organizations.

Benefits of FPA:

- FPA is a tool to determine the size of a purchased application package by counting all the functions included in the package.
- It is a tool to help users discover the benefit of an application package to their organization by counting functions that specifically match their requirements.
- It is a tool to measure the units of a software product to support quality and productivity analysis.
- It s a vehicle to estimate the cost and resources required for software development and maintenance.
- It is a normalization factor for software comparison.

Lines Of Code

line of code (LOC) is any line of text in a code that is not a comment or blank line, and also header lines, in any case of the number of statements or fragments of statements on the line. LOC clearly consists of all lines containing the declaration of any variable, and executable and non-executable statements. As Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.

Features :

- Variations such as “source lines of code”, are used to set out a codebase.
- LOC is frequently used in some kinds of arguments.
- They are used in assessing a project’s performance or efficiency.

RMMM Plan:

A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan.

Risk Mitigation:

It is an activity used to avoid problems (Risk Avoidance).

- Finding out the risk.
- Removing causes that are the reason for risk creation.
- Controlling the corresponding documents from time to time.
- Conducting timely reviews to speed up the work.

Risk Monitoring:

It is an activity used for project tracking.

- To check if predicted risks occur or not.
- To ensure proper application of risk aversion steps defined for risk.
- To collect data for future risk analysis.
- To allocate what problems are caused by which risks throughout the project.

Risk Management and planning :

It assumes that the mitigation activity failed and the risk is a reality. This task is done by Project manager when risk becomes reality and causes severe problems. If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks.

Software Configuration Management

Configuration Management helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. It is abbreviated as the SCM process. It aims to control cost and work effort involved in making changes to the software system. The primary goal is to increase productivity with minimal mistakes.

Reverse Engineering:

Reverse engineering, also called back engineering, is the processes of extracting knowledge or design information from anything man-made and re-producing it or reproducing anything based on the extracted information. Reverse engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and a human analyst work together, and the directionality of the process are highly variable.

The abstraction level of a reverse engineering process and the tools used to effect it refers to the sophistication of the design information that can be extracted from source code. Ideally, the abstraction level should be as high as possible
The completeness of a reverse engineering process refers to the level of detail that is provided at an abstraction level. In most cases, the completeness decreases as the abstraction level increases

SOFTWARE RE-ENGINEERING

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

Objectives of Re-engineering:
- To describe a cost-effective option for system evolution.
- To describe the activities involved in the software maintenance process.
- To distinguish between software and data re-engineering and to explain the problems of data re-engineering.

Used Case Diagram

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system

UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case

Project Scheduling

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following:

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.