SEPM Copy Notes

ICT 5th Semester

by

Mukesh Singh
ICT 5th batch 2073

Sukuna Multiple Campus
Sundarharaincha, Morang

Unit-1

Software & Software Engineering

1.1. Introduction to Software

Software is the collection of computer programs, procedures and documentation that performs different tasks on a computer system.

It is not the physical part like computer hardware. It is non-touchable part of the computer system. It makes computer hardware to perform tasks.

Examples:- Microsoft Windows xp/7, Linux, Unix, MS-word, VLC player etc. are computer software.

There are two main types of software:

i) System Software
ii) Application Software

1.2. Evolving Role of Software

Software takes the dual role. It is both a product and a vehicle for delivering a product.

As a product: It delivers the computing potential embodied by computer hardware or by a network of computers.

As a vehicle: It is the information transformer producing, managing, acquiring, modifying, displaying or transmitting the information that can be as simple as single bit or as complex as a multimedia presentation.

Software delivers the most important product of our time - information. It transfers the personal data. It provides the gateway to world wide information networks. It provides the means for acquiring information. It manages the business information to enhance competitiveness

1.3. Program vs Software

Program	Software
1. A program is set of instructions which performs only a specific type of task.	1. The software is a broad term which is designed to perform some specific set of operations.
2. A program consists of set of instructions.	2. A software consists of bundles of programs and data files.
3. A program cannot be classified into various categories.	3. A software can be classified into two categories: application and system software.
4. Programs don't have a user interface.	4. Every software has a dedicated user interface.
5. SDLC is not used to make programs.	5. SDLC is used to make / develop every software.
6. It depends on compiler.	6. It depends on OS.
7. Developer is the beginner person.	7. Developer is the experienced or well trained person.
8. Examples: adding two numbers, factorial, greatest among three numbers etc.	8. Examples: MS Word, Google Chrome, VLC player, Windows, UNIX, LINUX etc

1.4. Characteristics of Software:

Major characteristics of software are described below:

i) Functionality:

It refers to the degree of performance of the software against its intended purpose.

ii) Reliability:

It refers to the ability of the software to provide desired functionality under the given conditions.

iii) Usability:

It refers to the extent to which the software can be used with ease.

iv) Efficiency:

It refers to the ability of the software to use system resources in the most effective and efficient manner.

v) Maintainability:

It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

vi) Portability:

It refers to the ability of software to function properly on different hardware and software platforms without making any changes in it.

1.5. Types of Software

There are two types of Computer Software. They are:
a) System Software
b) Application Software

A) System Software:

It is the computer software designed to operate the computer hardware and to provide a platform for running application software.
It is a collection of operating systems; device drivers, servers, windowing systems and utilities.
Types of system software are described below:

i) Operating System Software

It is a system software, which manages the resources of the computer system such as memory, storage, processor, I/O devices and also acts as an interface between machine and user.
Examples:- Windows xp/7/8, Unix, Linux etc

ii) Utility Software

It is the helpful software that performs specific tasks related to the maintenance of computer hardware and data. It helps to keep the computer in the smooth functioning condition. It helps in transferring data and file, recovering lost data and file, searching & removing computer viruses etc.
Examples ⇒ Download Accelerator, Anti-Viruses, WinZip,

WinRar, Registry Cleaner, etc.

iii) Language Processor

Language processor is the program that converts programs written in assembly and high level language into machine language.

There are three types of language processors. They are: Assembler, Interpreter and Compiler.

B) Application Software

Application software is the special software that can do some specific tasks. It is designed to fulfill requirement of people. It helps user to perform specific tasks like preparing documents, producing bills, editing images and videos, preparing SLC results etc. Application software does its tasks with the help of operating system.

There are two types of application software. They are:

i) Packaged Software

Packaged Software is the readymade software developed for all general users to perform their generalized tasks. Software companies used to develop packaged software. They are general purpose software.

Some common types of packaged software are given below:

- Word Processing Software - Used for creating documents. Eg: MS-Word, Adobe Page Maker, Wordpad etc.

- Spreadsheet Software - Used for keeping accounts and do calculations.
  Eg: MS-Excel, Lotus 1-2-3, etc.

- Database Management Software - Used for managing databases.
  Eg: MS-Access, dBase, FoxPro, Oracle, etc.

- Graphics Software - Used for creating & manipulating images.
  Eg: MS-Paint, Photoshop etc.

- Multimedia Software - Used for creating and playing audio & video media.
  Eg: Powerpoint, VLC Media Player, Windows media player, Flash Player, etc.

ii) Tailored Software (Customized Software)

Customized (Tailored) Software is the application software which is designed to fulfill the specific requirements of an organization, office or individual. They are developed for specific task.
Examples :- banking software, hospital software, bill processing software, etc.

1.6. Generic View of Software Engineering

The process of software development/engineering has three generic views which are given below:

1. Definition Phase:

The definition phase focuses on "what". The experts get knowledge about "what".

The software engineer attempts to identify:

- What information is to be processed,
- What functions and performance are required,
- What system behaviour can be expected,
- What interfaces are to be established,
- What validation criteria are required,

to defined a successful system.

It contains three steps:

a) Analysis of System
b) Planning of project
c) Requirement analysis

2. Development Phase

The development phase focuses on "how". That is, during development, a software engineer attempts to ~~identify~~ define: ~~how~~

- How data are to be structured,
- How function is to be implemented within software architecture,
- How interfaces are to be characterized
- How the design will be translated into a programming language,
- How testing will be performed.

This phase also contains three special steps:
a) Design of Software
b) Coding
c) Testing of Software System.

3. Maintenance Phase (Support Phase)

The Support (maintenance phase focuses on "change" associated with error correction, adoption of new ideas and changes due to enhancements brough by changing customer requirements.

It includes:
- fixing the error/bugs
- Adaptation
- Enhancement.

1.7. Software Process and Software Process Model

Software Process

A software process is a set of related activities that leads to the production of software. These activities may involve the development of the software from the scratch or modifying an existing system.

Any software process must include the following four activities:

i) Software Specification: Define the main functionalities of the software and the constraints around them.

ii) Software design and implementation: The software is to be designed and programmed.

iii) Software Verification and Validation:- The software must confirm to its specification and meet the customer needs.
iv) Software evolution (Software maintenance): The software is being modified to meet customer and market requirements changes.

Software Process Model

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.

Some models are described below:

1. Waterfall Model

The waterfall model is a sequential approach, where each fundamental activity of a process is represented as a separate phase, arranged in linear order. Each phase is carried out completely before proceeding to the next.

The phases of waterfall model are:
- Requirements analysis
- System Design
- Implementation
- Testing
- Deployment
- Maintenance

2. Prototyping Model

A prototype is a version of a system or part of the system that is developed quickly to check the customer's requirements or feasibility of some design decisions.

A prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time etc.

The phases of a prototype are:

- Establish Objectives
- Define prototype functionality
- Develope the prototype
- Evaluate the prototype

3. Spiral Model

The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.

It was designed to include the best features from the waterfall model and prototyping model, and introduces a new component; risk-assessment.

It has four phases:

- Identification
- Design
- Construct or Build
- Evaluation and Risk Analysis

1.8. Myth and Ethics on Software Engineering

Myths on Software Engineering:

i) Myth 1: I need to be a genious to become a developer.
Reality - Anyone can learn how to become a software developer.

ii) Myth 2: Learning to code is like learning brain surgery!
Reality - Learning to code is easy, mastering it is hard.

iii) Myth 3: I need a college degree to know how to become a software developer.
Reality - Programmers without degrees are more common than you think.

iv) Myth 4: I need serious math skills to learn how to become a software developer.
Reality - Success as a programmer and math proficiency are not directly correlated.

v) Myth 5: Knowing the 'best' programming language will accelerate my journey to learning how to become a Software developer.
Reality - There is no 'best' language to learn.

vi) Myth 6: It is too late for me to become a developer.
Reality- It is never too late to change careers!

Ethics on Software Engineering

Ethics is a group of moral principles ~~that~~ or values that define or direct us to the right choice.

Ethics of ~~software~~ engineering are mentioned below:

i) Disclose any software related dangers.
ii) Approve ~~only~~ only safe, well tested software.
iii) Only sign documents in the area of competence.
iv) Cooperate on matters of public concerns.
v) Produce software that respects diversity.
vi) Be fair and truthful in all matters.
vii) Always put the public's interest first.
viii) Donate professional skills to good causes.
ix) Accept responsibility for your own work.

Unit-2

Software Development Process Models

2.1. Waterfall model and enhance waterfall model

Waterfall model:

Waterfall model is a sequential model that divides software development into different phases. It is termed as waterfall because the model develops systematically from one phase to another into downward fashion. The output of one phase is used as the input of the next phase. Each phase is designed for performing specific activity and has different objectives. Every phase has to be completed before the next phase starts and there is no overlapping of the phases.

Since the phase fall from higher level to lower level, like a waterfall, it is named as waterfall model.

Phases of waterfall model are described below:

i) Requirement Analysis → All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

ii) System Design → Defines the solution system based on requirement analysis phase.

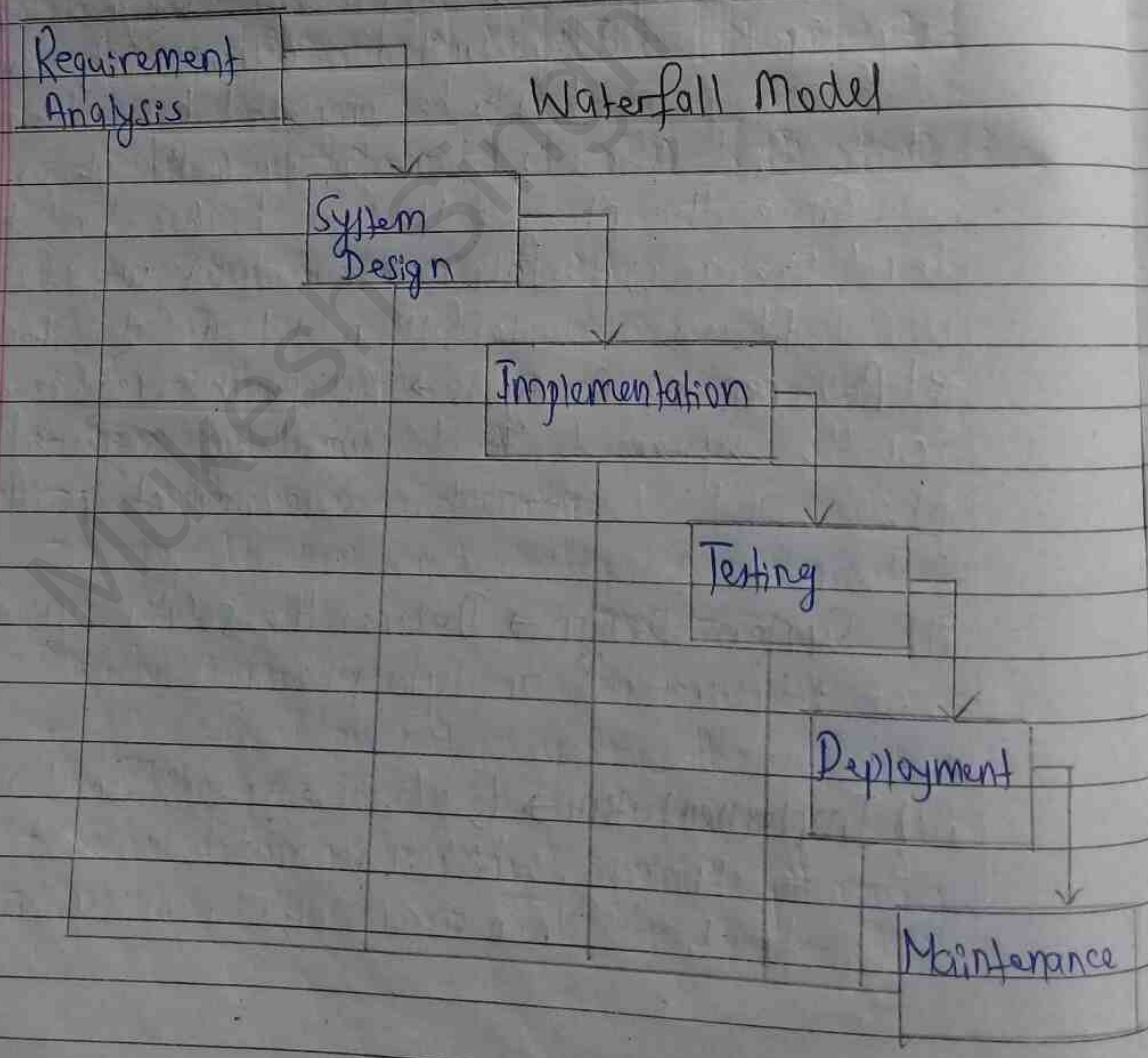
iii) Implementation → Each unit is developed and tested for functionality, which is referred to as Unit Testing. Construct, test, train users, install new system.

iv) Testing → The entire system is tested for any faults and failures.

v) Deployment → Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

vi) Maintenance → Keeps the system healthy and improved. To fix issues and updates.

Waterfall Model



Enhanced Waterfall Model:

It is also referred to as a linear-sequential life cycle model. It is almost same as the classical waterfall model except some changes are made to increase the efficiency of the software development. The phases in this enhance model are allowed to overlap. This model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.

2.2. Increment Process Models

The incremental process model is a method of software development where the product is designed, implemented and tested incrementally until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. This model combines the elements of waterfall model and they are applied in an iterative fashion.

Example: the word-processing software is developed using the incremental model.

Phases of incremental model are:

i) Requirement analysis: → Requirements and specifications of software are collected.

ii) Design → Defines the solution system based based on requirement analysis phase. Some high-end

functions are designed during this phase.

iii) Code → Coding of software is done during this stage.

iv) Test → The entire system is tested for faults and failures.

Fig: Increment Model

Advantages
- Less development cost.
- Product delivery is faster.
- Easier to test and debug

Disadvantages
- Cost of final product may cross the cost estimated initially.
- Requires very clear and complete planning.

2.3. Rapid Application Development (RAD)

RAD is a software development methodology that uses minimal planning in favour of rapid prototyping. RAD means less talk more action.

Phases of RAD are mentioned below:

i) Business Modelling → Information flow is identified.

ii) Data Modelling → Defines data objects needed for business.

iii) Process Modelling → Data objects defined in data modelling are converted ~~into~~ to achieve the business information flow.

iv) Application Generation → Automated tools are used for the construction of software, to convert process ~~models~~ and data models into prototypes.

v) Testing and turnover → Test new components and all the interfaces.

Advantages

- Flexible and adaptable to changes.
- Possibility of lesser defects
- Productivity can be increased in short time.

Disadvantages

- Can't be used for smaller projects.
- Not all applications are compatible with RAD.

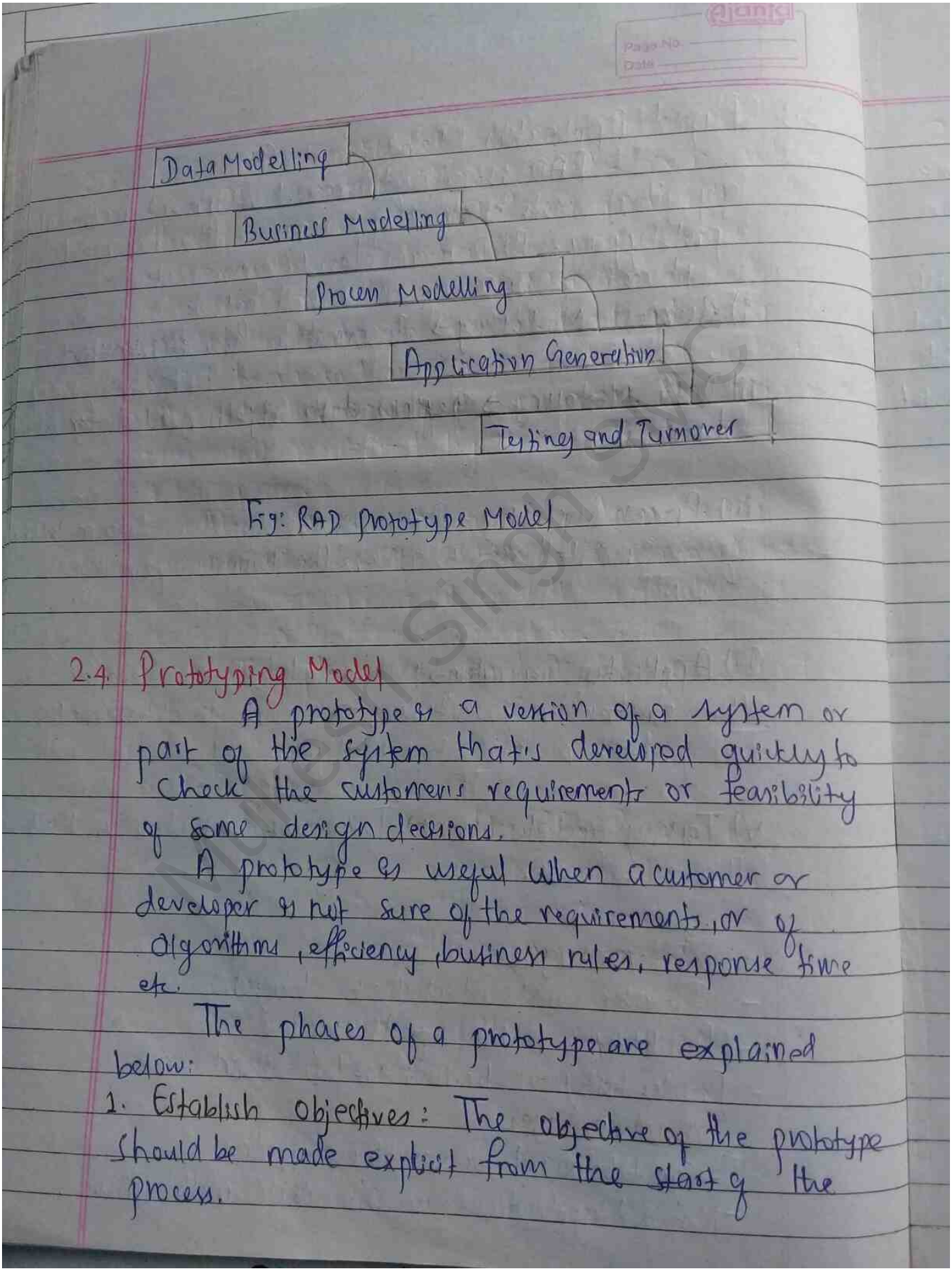


Fig: RAD Prototype Model

2.4. Prototyping Model

A prototype is a version of a system or part of the system that is developed quickly to check the customer's requirements or feasibility of some design decisions.

A prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time etc.

The phases of a prototype are explained below:

1. Establish objectives: The objective of the prototype should be made explicit from the start of the process.

2 Define Prototype functionality : It decides what are the inputs and the expected output from a prototype.

3. Develop the prototype: The initial prototype is developed that includes only user interfaces.

4. Evaluate the prototype:- Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback, both the specifications and the prototype can be improved.

2.5. Spiral Process Model

The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.

Spiral model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress. The spiral model was first mentioned by Barry Boehm in his 1986 paper.

The software engineering team adds functionality for the additional requirement in every increasing spirals until the application is ready for the production phase.

The phases of Spiral model are described below:

~~1) Identification~~

1. Planning Phase

Requirements like BRS (Business Requirement Specifications) and SRS (System Requirement Specifications) are gathered during the planning phase for continuous communication between the system analyst and the customer.

It includes estimating the cost, schedule and resources for the iteration.

2. Risk Analysis

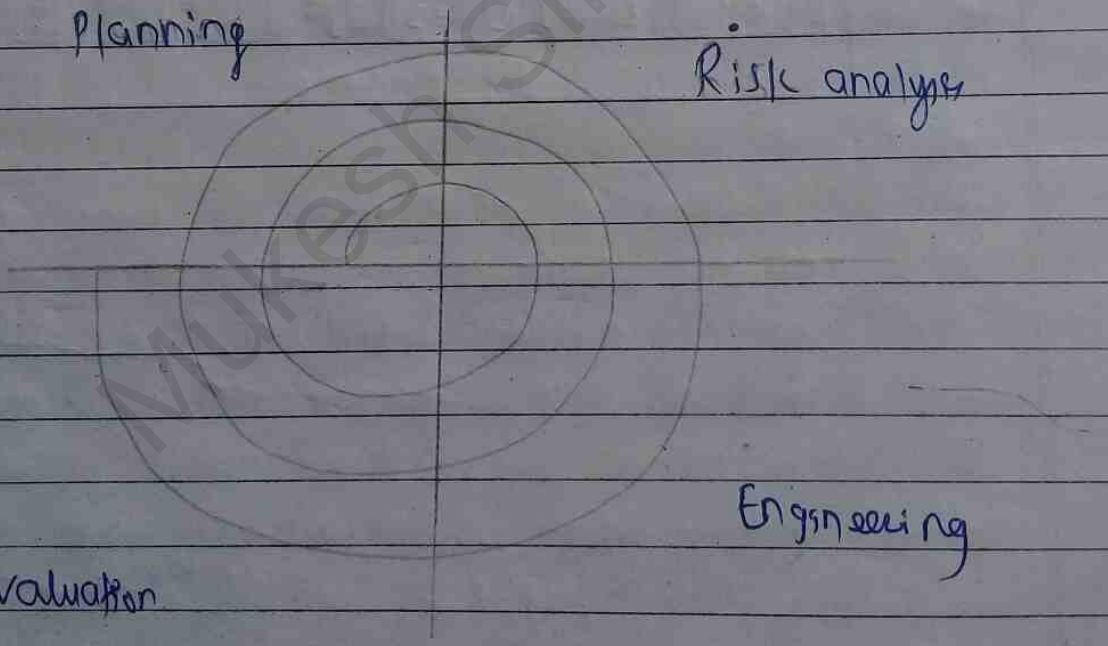
A process is undertaken to identify risk and alternate solutions. If any risk is found then alternate solutions are suggested and implemented.

3. Engineering Phase

Actual development and testing of the software take place in this phase.

4. Evaluation Phase

Customers evaluate the software and provide their feedback and approval.



Advantages
- Development is fast.
- Risk evaluation is proper.
- Control towards all the phases of development.
- Additional functionality can be added at a later date
- Software is produced early.

Disadvantages
- Requires expert people.
- Management is more complex.
- Can be costly model to use.
- Is not beneficial for smaller projects.

2.6. Rational Unified Process Model

The Rational Unified Process (RUP) Model is an iterative software development process framework created by the Rational Software.

RUP is an object oriented approach used to ensure effective project management and high quality software production.

RUP has four phases. Each phase has one key objective and milestone at the end.

1. Inception:

The idea for the project is stated. The development team determines if the project is worth pursuing and what resources will be needed.

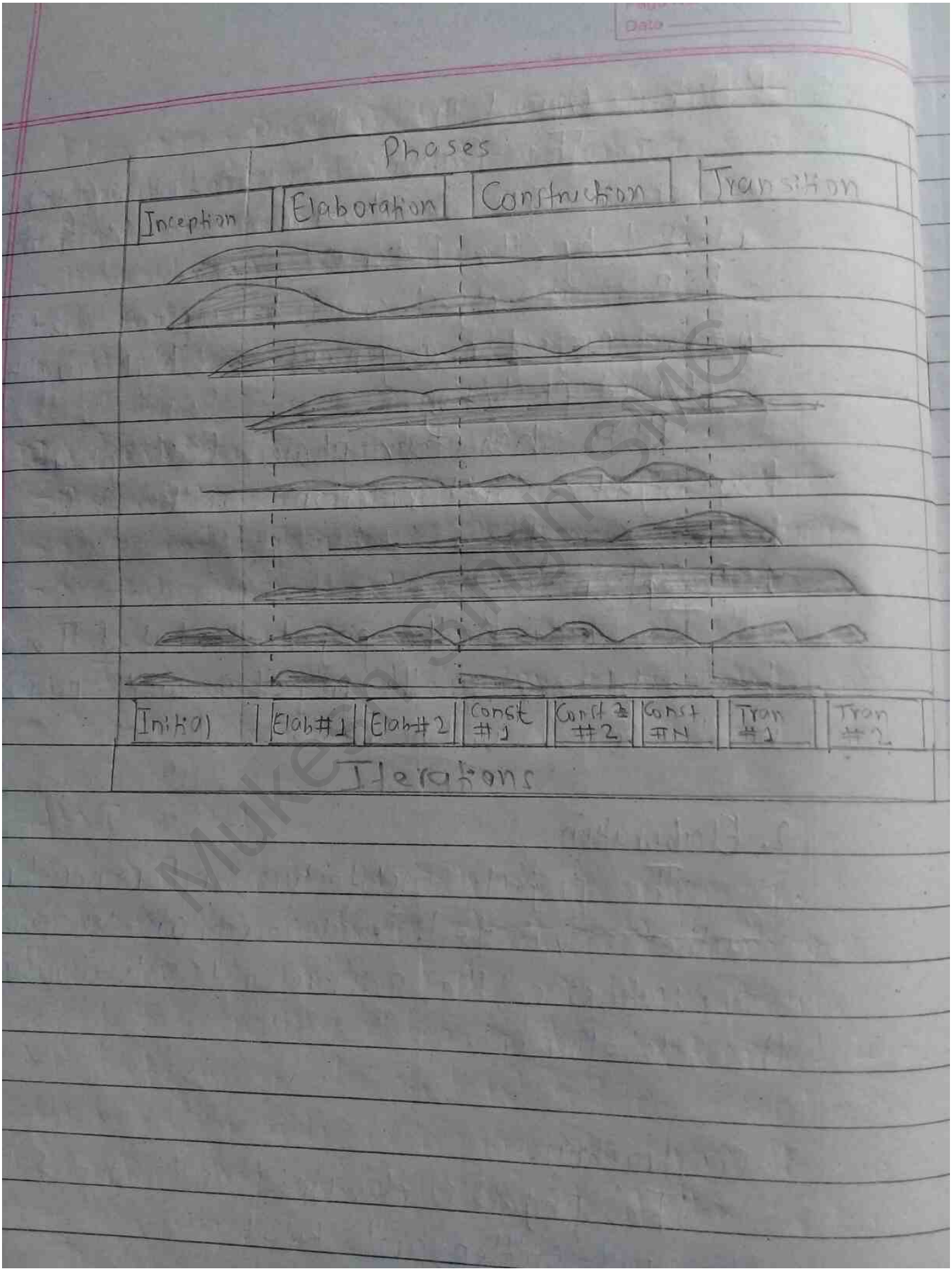
2. Elaboration

The project's architecture and required resources are further evaluated. Developers consider possible applications of the software and costs associated with the development.

3. Construction

The project is developed and completed. The software is designed, written and tested.

4. Transition: The software is released to the public. Final adjustments or updates are made based on feedback from end users.



2.7. Agile Model : XP and Scrum

Agile 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.

It promotes more involvement of customer during the software development.

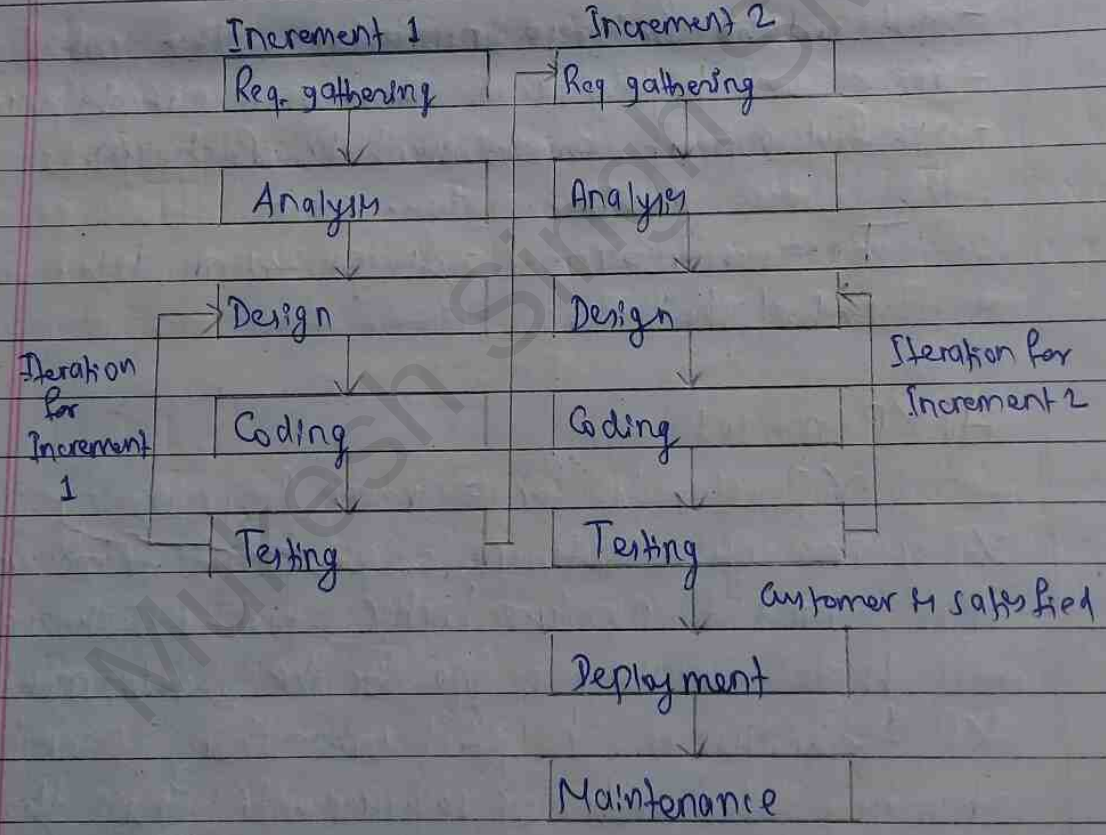


Fig: Agile Model

Advantages of Agile Model
- Supports customer involvement & customer satisfaction.
- Rapid development
- Allows changes easily
- Cost saving
- Fast delivery

Disadvantages
- Not suitable for large projects
- Senior and highly paid developers are required.
- No full support for documentation & design.

There are several agile process models. Some of described below:

A) XP model

XP stands for 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 technique is helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system.

It also introduces a checkpoint where any customer requirements can be easily implemented. The XP develops software keeping customer right in the target.

Phases of extreme Programming are mentioned below:

1) Planning
Gather requirements before development starts.
Identification of stakeholders and sponsor.

2) Analysis
Find and eliminate defects early in the development life cycle in order to cut the defect-fix costs.
Resource planning for both Development and QA teams.

3) Design
Elaborate, analyze and ~~analyze~~ verify the models before development.
Defines the solution system based on requirement analysis.

4) Coding
Coding of software is done during this phase.
Everyone reviews code and any developer can add functionality, fix bugs, or refactor.

5) Testing
The entire system is tested for any faults and failures.
The team performs unit tests and fixes bugs before the code can be released. They also run acceptance tests frequently.

6) Closure
Training
Production launch
SLA Guarantee assurance
Product support

B. Scrum Model
Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. Scrum itself is a simple framework for effective team collaboration on complex products.

Unit-3

Software Requirement Specification

3a. ~~Software requirement and its types~~

A software requirements specification (SRS) is a description of a software system to be developed. It is modeled after business requirement specification, also known as a stakeholder requirements specification.

OR

A SRS is a detailed description of a software system to be developed with its functional and non-functional requirements

Characteristics/qualities of SRS.

i) Correct - Correctness of SRS should be checked.

ii) Unambiguous - Ambiguity should be avoided.

iii) Complete - Requirements should be complete.

iv) Consistent - The SRS should be consistent within itself and consistent to its reference documents.

v) Ranked for importance and stability → All requirements are not equally important. Stability implies the probability of changes in the requirement in future.

vi) Verifiable → Requirements are verified with the help of reviews.

vii) Modifiable → Requirements ~~should~~ document should be created in such a manner that these changes can be modified easily.

viii) Traceable → SRS is traceable when the source of each requirement is clear.

3.1 Software requirement and its types:

Software requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software.

Software requirements are description of features and functionalities need to be implemented in the system.

Types of Software Requirements:

1. Business Requirements (BR):
→ These are high-level business goals of the organization building the product, or the customer who commissioned the project.
→ These are usually provided as a single page of high-level bullets.

2. Market Requirements (MR):
→ In addition to business goals, they also outline market needs.
→ These are usually provided as a bulleted list or table, and are usually less than 5 pages long.

3. Functional Requirements (FR):
These are the functions or features that must be included in the system in order to satisfy the

busines needs and be acceptable to the system users. They are listed below:

- Technical Specifications
- Busines Rules
- Calculations
- Historical data
- Data manipulation & processing
- System parameters
- System Constraints
- Administrative functions
- Certification requirements

4. Non-functional Requirements (NFR's)

It is any requirement which specifies how the system performs a certain function.

It specifies the system's quality, attributes or characteristics some of them are:

- Performance
- Series
- Information
- Security
- Control
- Efficiency
- Reliability
- Regularity
- Recoverability
- Scalibility
- Availability
- Maintainability
- Serviceability
- Usability

5. UI Requirements (UIR)

→ User interface specifications are not considered requirements in traditional requirement management theory.

→ They should be considered as integral part of requirements for any software that has a UI.

3.2. Requirement Engineering:

Requirements engineering refers to the process of defining, documenting and maintaining requirements in the engineering design process.

It is the process of gathering and defining what the services should be provided by the system.

Requirements engineering activities are as follows:

1. Requirements ~~elicitation~~ inception or requirements elicitation
→ ~~Requirements~~ It is related to the various ways used to gain knowledge about the project domain and requirements.

OR It is the gathering and discovery of requirements from stakeholders and other sources. Various techniques such as JAD, interviews, prototyping etc are used for that.

2. Requirement ~~Specification~~ Analysis:

Requirements are identified and conflicts with stakeholders are solved.

3. Requirement Specification

Requirements are documented in a formal artifact called requirements specification (RS). which will become official only after validation.

A RS can contain both writter & graphical information if necessary.

Models such as ER model, DFDs, FDs etc are used.

4. Requirements Validation:
It refers to checking that the documented requirements and models are consistent and meet the needs of the stakeholders.

5. Requirements Management:
Managing all the activities related to the requirements since inception, supervising as the system is developed and eventually it is put into use.

3.3 Requirements Elicitation

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".

Following techniques can be used for requirements elicitation:

i) Interviews → A good system analyst must be good at interviewing and no project can be conducted without interview.

ii) Questionnaire → For effective survey, the analyst should group users properly and design different questionnaires for different groups.

iii) Sampling → Predetermined number of observations from large population can be collected.

iv) Survey → It investigate the opinions or experience of a group of people by asking them questions.

iv) Joint Application Design (JAD) → It is team based approach for defining requirements for new or modified information systems. Its purpose is to collect information from system requirements simultaneously from the key people involved with the system to find conflicts.

v) Prototyping → Prototyping is means of exploring idea before you invest in them. It is needed when user requirements are not clear or well understood.

vi) Usecase list → A use case is a list of actions or events typically defining the interactions between a role and a system to achieve a goal. The role (actor) can be a human or other external system.

It is methodology used to identify, clarify and organize system requirements.

3.4. Requirement Analysis

Requirement analysis is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. Such requirements are often called functional specifications in software engineering. Requirement analysis is important aspect of project management.

3.5 Requirement documentation and validation

Requirement documentation:

Requirement documentation is a compilation of a variety of documents together. It is description of what a particular software does or shall do. The development of software begins once the requirements document is ready.

After the requirements have been documented fully, they should be reviewed by all interested stakeholders to ensure they are correct, relevant and clear.

Also, errors present in the SRS will adversely affect the cost if they are detected later in the development process or when the software is delivered to the user.

Types of documentation include:

i) Requirements → Attributes, capabilities or qualities of a system.

ii) Architecture/design → Overview of software.

iii) Technical → Documentation of code, algorithms, interfaces and APIs.

iv) End User → Manuals for end-user, system administrators, and support staff.

v) Marketing - How to market the product and analysis of the market demand.

Requirement Validation

To check all the issues related to requirements, requirements validation is performed.

It's a process of ensuring that specified requirements meet the customer needs. It is concerned with finding problems with the requirements.

It refers to checking that the documented requirements and models are consistent and meet the need of the stakeholder.

During the requirements validation process, different types of checks should be carried out on the requirements. These check includes:

i) Validity checks → The functions proposed by stakeholders should be aligned with what the system needs to perform.

ii) Consistency Checks → Requirements in the document shouldn't conflict or different descriptions of the same function.

iii) Completeness Check → The document should include all the requirements and constraints.

iv) Realism checks → Ensure the requirements can be actually be implemented using the knowledge of existing technology, the budget, schedule, etc.

v) Verifiability → Requirements should be written so that they can be tested.

3.6. Requirements management:

Requirements management can be defined as a process of eliciting, documenting, organizing and controlling changes to the requirements.

The process of requirements management begins as soon as the requirements document is available.

The activities performed in requirements management are listed below:

i) Change management → Recognizing the need for a change in the requirements.

ii) Establishing a relationship among stakeholders and involving them in the requirements engineering process.

iii) Identifying and tracking requirements attributes.

3.7. SRS documents:

SRS is a document that describes the nature and features of a project, software or application.

SRS document is a formal document that clearly tell you "what software system to build". It does tell you "How to build the software system".

In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known as SRS report, software document.

SRS document comprises of following sections:

Table of Contents for a SRS Document:

1. Introduction
   1.1. Purpose
   1.2. Documentation Conventions
   1.3. Project Scope
   1.4. References

2. Overall Description
   2.1. Product Perspective
   2.2. Product features
   2.3. User classes & characteristics
   2.4. Operating environment
   2.5. Design & implementation Constraints
   2.6. Assumptions & dependencies

3. System Features
   3.1. Functional Requirements

4. External Interface Requirements
4.1. User Interfaces
4.2. Hardware Interfaces
4.3. Software Interfaces
4.4. Communications Interfaces

5. Non-functional Requirements
5.1. Performance Requirements
5.2. Safety Requirements
5.3. Security Requirements
5.4. Software Quality Attributes

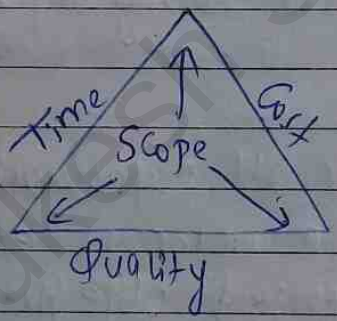
Unit-4

Software Project Management

Software project management is an art and science of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled.

4.1. Software Project

A software project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.



The above image shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constraint and deliver the project as per scheduled time.

4.2 Activities in Project Management

The activities/process of project management are mentioned below:

1. Initiating → Project initiation is the starting point of any project. All the activities related to winning a project takes place. Usually, the main activity of this phase is pre-sale.

2. Planning → Project planning is one of the main project management process. The main purpose is to plan time, cost and resources required. If the project management team gets this step wrong, there could be heavy negative consequences during the next phase of the project.

3. Execution → After all paperwork is done, in this phase, the project management executes the project in order to achieve project objectives.

4. Monitoring and Controlling

Monitoring and controlling consists of those processes performed to observe project execution so that potential problems can be identified in a timely manner and corrective action can be taken, when necessary, to control the execution of the project.

5. Completing or Closing →

Closing a project means finishing all activities across all process groups, disbanding the project team and signing off the project with the customer.

OR, Once all the project requirements are achieved, it's time to hand over the implemented system and close out the project.

4.3 Software Project Planning

Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project.

Software project planning is a task, which is performed before the production of software actually starts. It is a set of multiple processes, which facilitates software production.

Software project planning consists of following steps/activities:

Step 0: Select Project
Step 1: Identify project scope and objectives
Step 2: Develop strategies
Step 4: Develop project policies
Step 5: Make planning decisions
Step 6: Set procedures and rules for the project
Step 7: Develop a software project plan

Step 8: Prepare budget
Step 9: Conduct Risk management
Step 10: Document Software project plans

4.4 Software project management plan:

Software project manager prepares a document on the basis of decision finalised during the project planning. This document is known as Software Project management Plan Document (SPMP).

SPMP document is a well organized document that contains the project planning in detail.

A SPMP document is organized as below:

1. Introduction
   - Objectives
   - Functions
   - Performance issues

2. Project estimates
   - Historical data used
   - Cost, duration, effort estimates

3. Project Schedule
   - Work breakdown
   - Gant and PERT chart

4. Project resource
- Manpower
- Hardware & Software
- Highly skilled professionals

5. Staff organization
- Team formation and structure
- Management reporting

6. Risk Management
- Risk analysis
- Risk identification

7. Project tracking
8. Project Control
9. ~~Miscellaneo~~ Miscellenous activities

4.5 Software Project Scheduling & Techniques

Project scheduling is a mechanism to communicate what tasks need to get done and which organizational resources will be allocated to complete those tasks in what time frame.

A project schedule is a document collecting all the work needed to deliver the project on time.

Project scheduling involves:

- What tasks need to be carried out?
- How long will they take?
- When the task will occur?
- Who will do it?

Some most common scheduling techniques are mentioned below:

1. GANTT Chart

- GANTT Charts are also known as bar charts.
- It is a horizontal bar or line chart plotted over time (eg. days, weeks, months).
- Activities are represented are bars (on left hand side) and the length of each bar represents the activity duration.
- The beginning of the bar shows the start date and the end of the bar shows the end date of the activity.
- Depending on the project execution plan and resource availability, these bars may be sequential

and run in parallel.

2. Schedule Network Analysis:

It is a graphical display (from left to right across a page) of all logical interrelationships between elements of work in chronological order, from initial planning to project closure.

As a project progresses, regular analysis of the network diagram is a check to ensure the project is proceeding on track.

3. Critical Path Method:

- The Critical path method (CPM) is a visual technique which enables to show activities, activity dependencies and durations in the same diagram.
- In this method, activities are linked to each other by using dependencies in a network diagram. These activity group form paths.
- It takes longest time to complete.

Activities are listed, activity durations are determined, dependencies between activities are established and the longest path is identified by making forward and backward direction.

4. PERT (Program Evaluation & Review Technique) method:

PERT chart differs from CPM charts in the way times are calculated for activities.

In this method, activities are listed, activity durations are determined by using three estimates of times: Shortest time (ST), Longest Time (LT) and the most likely time (MT).

Estimated time of the task is calculated by using these three estimates. The formula is:

Expected time (ET) = (ST + 4MT + LT) / 6

9.6. Software Project Team Management & Organization

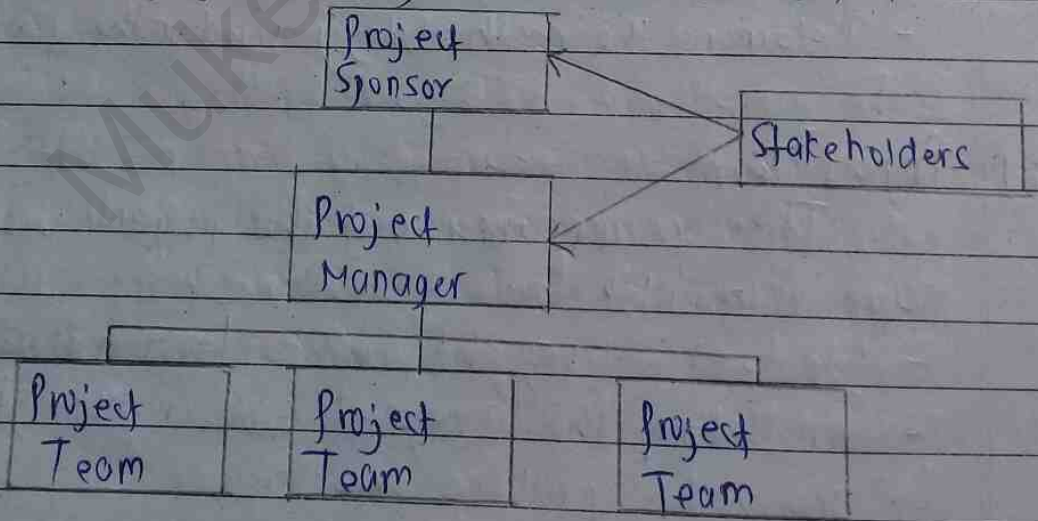
Proper ~~project~~

A Project Team is an organized group of people who are involved in performing individual tasks of the project as well as achieving individual goals and objectives for the purpose of accomplishing the project and producing its results. The team consists of full-time and part-time human resources.

Proper project team organization is one of the key constraints to project success. Successful development projects take careful planning, a talented team and collaboration of a project's team members, both internal and external (client & representatives).

Software projects only move forward when these key team members are in place.

The project management ~~chart~~ chart looks like this:



i) Project Sponsor

This is the person to whom the project's deliverables are delivered. They are one level ~~higher~~ above than the project manager. They are not diratly involved with day to day execution of project.

ii) Project Manager

Project manager is the person who handles the day to day administration of the project and project team and is ultimately accountable for the project's success.

Their job is to:

- develop a project plan
- ensure project deliverables are produced on time, on budget, on quality
- ~~Recr~~ Recruit project staff
- Lead and manage project team
- Determine the methodology used on the project

iii) Project Team

This group carries out the project work. They are responsible for:

- Understanding the work to be completed
- Completing the assigned work within the budget, timeline and quality expectations
- Informing the project manager for ~~a~~ issues, scope changes, risk and quality concerns.

iv) Stakeholders

These are the specific people or groups who have a stake or an interest in the outcome of the project.

Normally stakeholders are from within the company and could include:

- internal clients
- management
- employees
- administrators etc

A project may also have external stakeholders including:

- Suppliers
- investors
- Vendors
- Organizational groups
- government organizations

4.7. Project estimation technique: COCOMO model

COCOMO stands for Constructive Cost Model. It is a regression model based on LOC i.e number of lines of Code. It is used for estimating and evaluating the cost of software development. It uses regression formula with parameters based on historic data. It was proposed by Barry W. Boehm in the late 1970's and is based on the study of 63 projects, which make it one of the best documented models.

There are 3 types/levels of COCOMO model:

1. Basic COCOMO Model:

Basic COCOMO can be used for quick and slightly rough calculations of Software Costs but its accuracy is limited due to the absence of sufficient factor considerations.

2. Intermediate COCOMO model

It is an extension of basic model. It computes software development effort by adding a set of "cost drivers", that will determine the effort and duration of the project such as assessment of personnel and hardware.

3. Detailed COCOMO model:-

It is an extension of ~~bas~~ intermediate model. It adds effort multiplier for each phase of the project

to determine the cost driver impact of each step.

COCOMO applies to 3 classes of software projects:

i) Organic Projects :→ "small teams with "good" experience working with "less than rigid" requirements.

ii) Semi-detached Projects → "Medium" teams with "good" experience working with a mix of rigid and less than rigid requirements.

iii) Embedded Projects :→ Developed with a set of "tight" constraints (hardware, software, operational ...).

Estimation of Effort: Calculations

a) Basic Model

$E = a(KLoC)^b$

Software Projects	A	B
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

b) Intermediate Model

$E = (a(KLoC)^b) * EAF$

Software Projects	A	B
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Advantages of COCOMO model:
- It is a repeatable process.
- It supports different models and levels.
- It is thoroughly documented.
- It is easy to use.

Disadvantages
- It ignores documentation and other requirements.
- It ignores software safety issues.
- It ignores software development environment.
- It ignores many hardware issues.

4.8 Risk analysis and management:

Risk is the probability of occurence of an undesirable event.

Risk analysis is the review of the risks associated with a particular event or action.

It is a technique used to identify and assess factors that may jeopardize the success of a project or achieving a goal.

Risk management is the process of identifying, analyzing and then responding to any risk that arises over the life cycle of a project to help the project remain on track and meet its goal.

4.9 Risk Management Process:
The steps of Risk management process are explained below:

1. Identify the Risk :→ It is the process of determining which risk may affect the project most. This process involves documentation of existing tasks. There are many types of risks - legal risks, environmental risks, market risks, regulatory risks etc.

2. Analyze the Risk → Once a risk has been identified, it needs to be analyzed. The scope of risk must be determined.
- How likely are these risks to occur?
- And if they do occur, what will the ramifications be?

3. Evaluate or Rank the Risk :→
Risks need to rank and prioritize. Rank each risk by factoring in both its likelihood of happening and its potential effect on the project.

4. Treat or Resolve the Risk → Every risks need to be eliminated as much as possible. This is done by connecting with the experts of the field to which the risks belongs to.

5. Monitor and Review the Risk :→ Not all risk can be eliminated - some risks are always present. Market risks and environmental risks are just two examples of risks that always needs to be monitored. Computers are much better at continuously monitoring risks than people.

The overall risk management process should also be reviewed and updated accordingly.

Unit -5

Software Design

Software design is a process to transfer user requirements into some suitable form, which helps the programmer in software coding and implementation.

5.1. Design framework

A design framework is a simple visual structure that helps organize the information and ideas of a problem so you can work on it more effatively.

A framework is often composed of a relevant list of categories. These categories are developed from initial research that should be a part of every new project.

A software framework is a platform for developing software applications. It provides foundations on which software developers can build programs for specific platform.

5.2. Software Design Models

Software design is the process of defining software ~~models~~ methods, functions, objects and the overall structure and interactions of your code so that the resulting functionality will satisfy your user requirements.

The design model is an object model describing the realization of use cases, and serves as an abstraction of the implementation model and its source code.

The design model is developed by the software designer only whose main goal is to develop a model or abstraction of the product based on experience design principles its guidelines, and fundamental concepts. This model is then evaluated ~~to~~ with respect to the requirements and operating and economic environments.

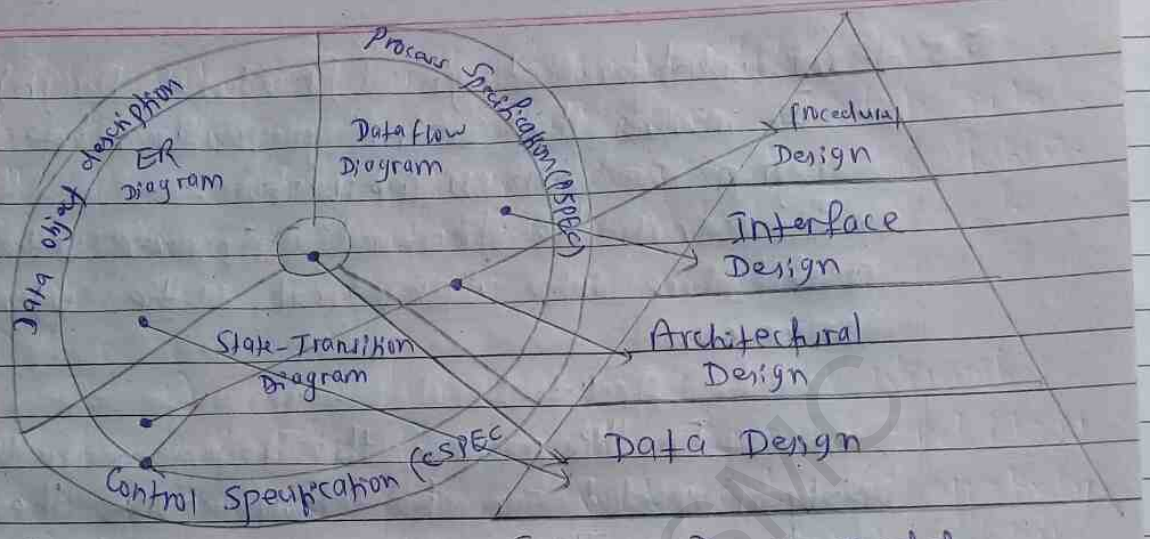


fig: Translating analysis model into software design model.

1. Data Design
   - Data design requires data dictionary and entity relationship diagrams.
   - The structure of data is most important part of software design.

2. Architectural Design:
   - It consists of data flow diagrams.
   - It provides the overall view of the system.

3. Interface Design
   - It requires control specification, state transition diagrams and data flow diagrams.
   - It represents the information flow within it and out of the system.

4. Component level design (Procedural Design)
- It involves the control specification, state transition diagram and process specification.
- It converts the structural components of the software architecture into a procedural of the software components.

In this manner, the requirements through analysis model can be translated into the software design model. This model serves as the base of the coding phase. So, design is the only tool that depicts the transformations of customer requirements into finished software product. Design also describes the quality of the software work product, which goes throughout the life cycle phases - Coding, testing and maintenance.

5.3 Design Process:

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

5.4 Architecture Design

Architectural design is the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

Architectural design is an early stage of the system design process. It represents the link between specification and design processes and is often carried out in parallel with some specification activities. It involves identifying major system components and their communications.

Software architectures can be designed at two levels of abstraction:

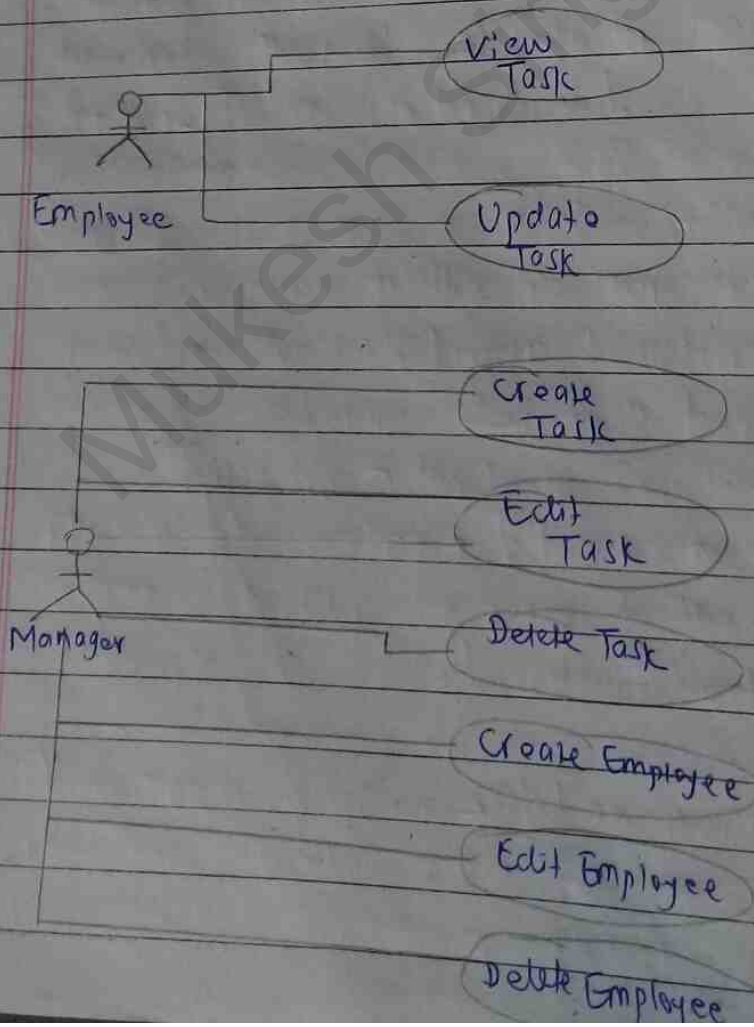
i) Architecture in the small - is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.

ii) Architecture in the large - is concerned with the architecture of complex enterprise systems that include other systems, programs and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

5.5 Low level design

Low Level design (LLD) is a component level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall the data organization may be defined during requirement analysis and then refined during data design work. Post-built, each component is specified in detail.

The LLD phase is the stage where the actual software components are designed.



5.6. Coupling and Cohesion

Cohesion:

Cohesion is a measure of degree to which the elements of the module are functionally related. A good software design will have high cohesion. There are seven types of Cohesion:

i) Co-incidental Cohesion → There is no relationship among elements of a module. The parts of component are not related but simply bundled into a single component. It is unplanned and random cohesion.

ii) Logical Cohesion → When logically categorized elements are put together into a module, it is called logical cohesion. Operations are related, but functions are significantly different.

iii) Temporal Cohesion → When elements ~~are~~ of module are organized such that they are processed at a similar point in time, it is called temporal cohesion. It is same as but higher than logical cohesion since all elements are executed together.

iv) Procedural Cohesion → When elements of module are grouped together, which are executed sequentially ~~and work on same data, it is called~~ in order to perform a task, it is called procedural cohesion.

v) Communicational Cohesion → When elements of module are grouped together, which are executed sequentially and work on same data, it is called communicational cohesion.

vi) Sequential Cohesion → When elements of module are grouped ~~together~~ because the output of one element serves as input to another and so on, it is called sequential cohesion.

vii) Functional Cohesion → It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

Coupling is the measure of degree of interdependence between the software modules. A good software ~~dat~~ will have low coupling.

There are 5 levels of coupling:

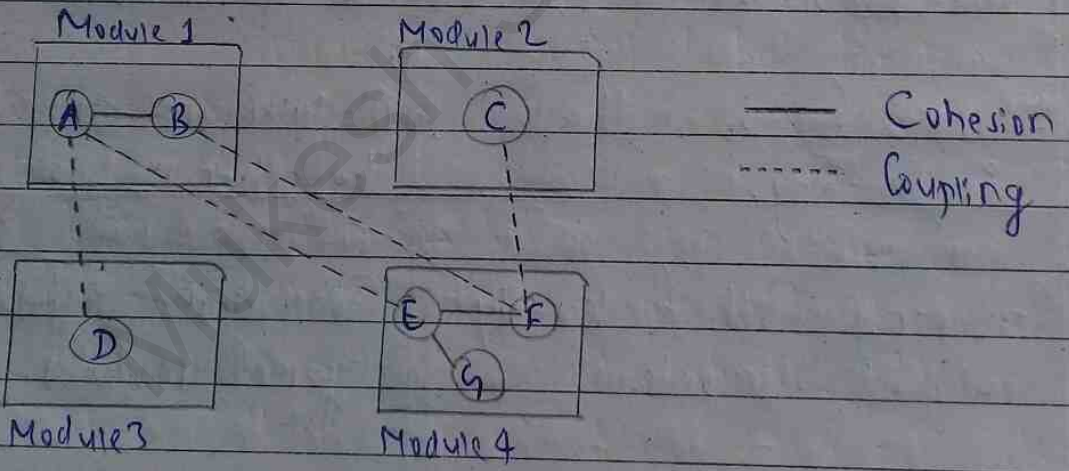
i) Content Coupling → When a module can directly access or modify or refer to the content of another module, it is called content level coupling.

ii) Common Coupling → When multiple modules have read and write access to some ~~logical~~ global data, it is called common or global coupling.

iii) Control Coupling → Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.

iv) Stamp Coupling → When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

v) Data Coupling → Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.



Cohesion	Coupling
1. Cohesion is the indication of relationship within module.	1. Coupling is the indication of the relationship between modules.
2. Cohesion is the concept of intra module.	2. Coupling is the concept of inter module.
3. Increasing in cohesion is good for software.	3. Increasing in coupling is avoided (bad) for software.
4. Cohesion represents the functional strength of modules.	4. Coupling represents the independence among modules.
5. Highly cohesive gives the best software.	5. Loosely coupling gives the best software.
6. Module focuses on the single thing.	6. Modules are connected to the other modules.
7. Grouping of all functionally related elements.	7. Communication between different modules.

5.7. Software design Strategies:

Software design is a process of implementing software solutions to one or more sets of problems.

Software design strategy is a discipline that helps firms in deciding what to make, why to make it and how to innovate contextually, both immediately and over the long term. The software design strategy is mainly about organizing design activities during the course design.

Various software design strategies are mentioned below:-

1. Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved.

Structured design is mainly based on 'divide and conquer' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

A good structured design has high cohesion and low coupling arrangements.

2.

2 Function Oriented Design

In function oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Design Process:

- Data flow diagram depicts the flow of data in the system.
- The process of changing the state and the data of the system is depicted by DFD.
- The system is divided into different sub-system based on their operations called functions.
- Each of the functions are described.

3. Object Oriented Design

The main focus of object oriented design is on entities and their characteristics rather than with the functions involved in the software system.

Some important Concepts of Object Oriented Design:

a) Objects → All entities involved in the solution design are known as objects. For example: person, bank, company and customer are treated as objects.

b) Classes → Collection of objects is called a class. It is a logical entity. A class is generalized description of an object. The features of objects, functions and methods are described and defined by

the class.

c) Encapsulation → Encapsulation is a combination of attributes and methods together. The encapsulation not only bundles the important information about the object together, but also restricts the access of data from the outside world.

d) Inheritance → When one object acquires all the properties and behaviours of a parent object it is known as inheritance.

e) Polymorphism → If one task is performed in different ways, it is known as polymorphism.

Design Process

- A solution design is created from requirement or previous used system and system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is ~~identified~~ defined.
- Application framework is defined.

5.10

Function Oriented Design	vs	Object Oriented Design
1. The basic abstractions, which are given to the user, are real world functions.		1. The basic abstractions are not the real world functions.
2. State information is often represented in a centralized shared memory.		2. State information is not represented in a centralized shared memory but distributed among objects of the system.
3. We decompose in function/ procedure level.		3. We decompose in class level.
4. Top-down approach.		4. Bottom-up approach.
5. Begins by considering the use-case diagrams and scenarios.		5. Begins by identifying objects and classes.
6.		6.

①

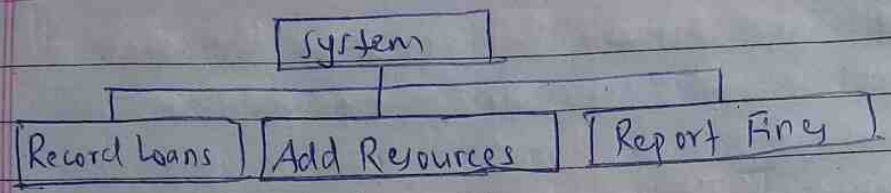


Fig: Structured Approach

②

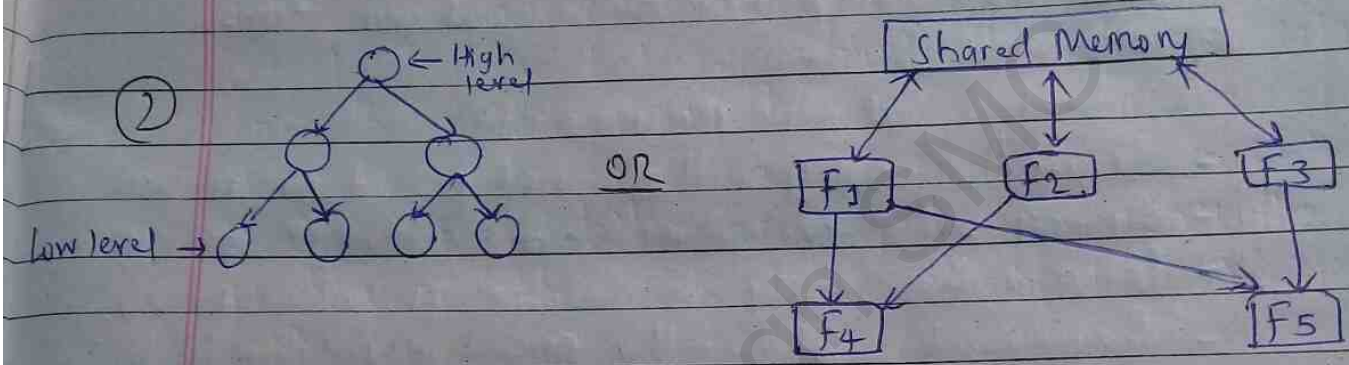
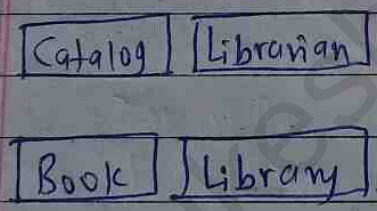


Fig: ~~Object~~ function oriented Design

③



Unit-6

Software Measurement and Metrices

6.1. Software measurement:

A measurement is a manifestation (indication) of the size, quantity, amount or dimension of a particular attributes of a product or process.

Software measurement is a quantified attribute of a characteristic of a software product or the software process. It is a discipline within software engineering. The process of software measurement is defined and governed by ISO standard, ISO 15939.

There are two types of software measurement:

1. Direct Measurement

In direct measurement, the product, process, or thing is measured directly using standard scale.

2. Indirect measurement:

In indirect measurement, the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

Software measurement consists a set of five activities:

i) Formulation:→ Performs measurement and develops appropriate metric for software under consideration.

ii) Collection → This collects data to derive the formulated metrics.

iii) Analysis → This calculates metrics and the use of mathematical tools.

iv) Interpretation → The analyzes the metrics to attain insight into the quality of representation.

v) Feedback → The communicates recommendation derived from product metrics to the software team.

Need of Software Measurement:
- Create the quality of the current product or process.
- Anticipate future qualities of the product or process.
- Enhance the quality of ~~software~~ product or process.
- Regulate the state of the project in relation to budget and schedule

for example, number of errors in a system is measurement.

5.2 Software Metrics:

A metrics is a measurement of the degree that any attribute belongs to a system, product or process. For example the number of errors per person hours would be a metrics.

(i) Software metrics is a measure of software characteristics which are quantifiable or countable.

(ii) Software metric is defined as quantitative measure of an attribute a software system possesses with respect to cost, quality, size and schedule.

(iii) Software metric is a standard of measure of a degree to which a software system or process possesses some property.

There are 3 types of software metrics:

1. Product Metrics: → It describes the characteristics or state of product such as size, complexity, design features, performance and quality level.

2. Process Metrics: → It can be used to improve software development and maintenance.

3. Project Metrics → It describes the project characteristics and execution process. Examples include: number of software developer, staffing pattern over the life cycle of software, cost and schedule, productivity etc.

There are 4 functions related to Software metrics:

1. Planning
2. Organizing
3. Controlling
4. Improving

6.3 Control Flow Graph

A control flow graph is a the graphical representation of control flow or computation during the execution of programs or applications.

A control flow graph is process oriented and can show all paths that can be traversed during a program execution. It is a directed graph.

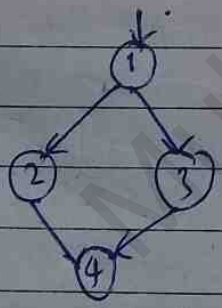
There are 2 designated blocks in Control flow Graph:

i) Entry Block:→ It allows the control to enter into the control flow graph.

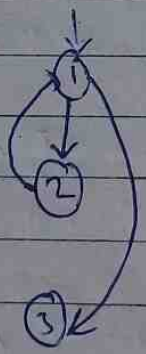
ii) Exit Block → Control flow leaves through the exit block.

General Control Flow Graphs

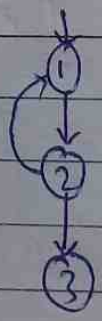
1. If-else



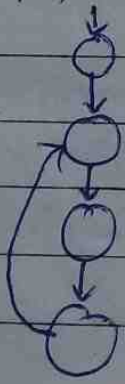
2. While

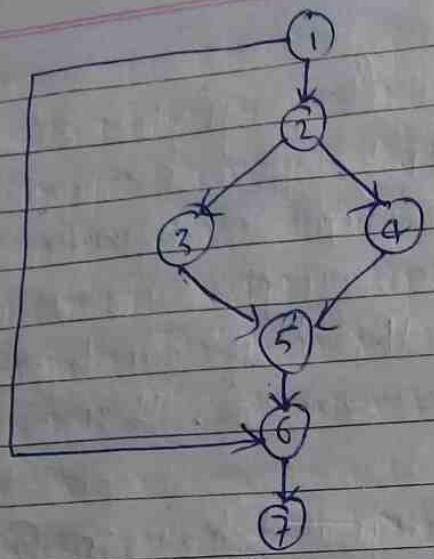


3. do while



4. for





Control flow graph:

Advantages:

- Can easily encapsulate the information per each basic block.
- Can easily locate inaccessible codes of a program, and syntactic structures.

6.4. Cyclomatic Complexity

- Cyclomatic Complexity is a software metric used to measure the complexity of a program.
- It is a source code complexity measurement that is being correlated to a number of coding error.
- It is calculated by developing a Control flow Graph of the program. The nodes in the ~~progra~~ graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e. if second command might immediately follow the first command.

or, - Cyclic complexity can be calculated with respect to functions, modules, methods or classes within a program.

- It was developed by Thomas J. McCabe in 1976.
- Lower the Program's cyclomatic complexity, lower the risk to modify and easier to understand. It can be represented using the below formula:

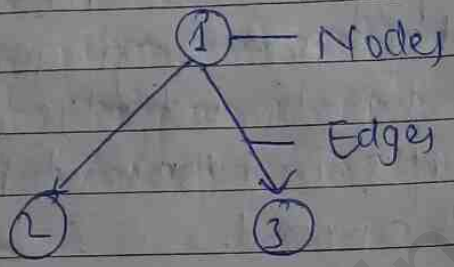
Cylomatic Complexity = E - N + 2*P

Where,

E = number of edges in the control flow graph.
N = number of nodes in the control flow graph
P = number of connected components

Steps for calculating Cyclomatic Complexity :

① Construction of ~~control flow~~ graph with nodes and edges from code.
② Identification of independent paths
③ Cyclomatic Complexity Calculation
④ Design of Test Cases.



6.5 Object Oriented Matrices

The metrics for object oriented system focus on measurements that are applied to the class and the design characteristics, for example encapsulation, information hiding, inheritances, localization, etc. So, object oriented oriented metrics are usually used to access the quality of software designs.

For object-oriented projects, different sets of metrics have been proposed. There are listed below:

i) Number of Scenario Scripts → There are detailed sequence of steps, which depicts the interaction between the user and the application. There are directly related to ~~number of~~ application size and number of test cases that are developed to test the software.
Note that scenario scripts are analogues to use-cases.

ii) Number of key classes → Key classes are independent components. They indicate amount of development effort and potential reuse.

iii) Number of support classes → These classes are required to implement the system but not directly related to problem domain. Example - User interface classes and computation class are support classes. These indicate the amount of development effort and potential reuse.

iv) Average number of Support Classes per Key class
→ Key classes are defined early in the software project while Support classes are defined throughout the project. If these are known then estimation would be much simplified.

v) Number of Subsystems → A collection of classes that supports a function visible to the user is known as a subsystem. Identifying subsystems makes it easier to prepare a reasonable schedule in which work on subsystems is divided among project members.

6.6. Lossless Decomposition:

A functional decomposition is the process of breaking down the functions of an organization into progressively finer and finer levels of detail.

There are two types of decomposition:

1. Lossy decomposition -

The decomposition of relation R into R_1 and R_2 is lossy when join of R_1 and R_2 does not yield the same relation as in 'R'.

i.e If we decompose R into R_1 and R_2, it is lossy if $R_1 \bowtie R_2 \supset R$

2 Lossless Decomposition

The decomposition of relation R into R_1 and R_2 is lossless when the join of R_1 and R_2 yield the same relation as in R.

i.e If we decompose R into R_1 and R_2, it is lossless if $R_1 \bowtie R_2 = R$

Unit-7

Configuration Management

7.1. Software Configuration Management

Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products.

SCM helps in identifying individual elements and configurations, tracking changes, and version selection, control and baselining.

SCM is also known as Software Control Management. SCM aims to control changes introduced to large complex software systems through reliable version selection and version control.

SCM is the task of tracking and ~~the~~ controlling changes in software, part of the larger cross-disciplinary field of configuration management.

Software Configuration Management Tasks:

i) Identification → tracking multiple versions to enable efficient changes.

ii) Version Control → Control changes before and after release to customer

iii) Change Control → Authority to approve and prioritize changes.

iv) Configuration auditing → ensure changes made properly.

v) Reporting → tell others about changes made.

7.2. Software Change Management

Software change management is the process of controlling change to software.

It is the process of selecting which changes to encourage, which to ~~forbid~~ allow, and which to prevent, according to project criteria such as schedule and cost.

OR, It is the process of requesting, determining attainability, planning, implementing and evaluating of changes to a system.

The general change management process:

→ The change is requested.
→ The change request is assessed against requirements and project constraints.
→ Following the assessment, the change request is accepted.
→ If it is accepted, the change is assigned to a developer and implemented.
→ The implemented change is audited.

7.3. Version and Release Management

Version Management (Version Control)

Version management is also called Version Control or Revision Control, is a means to effectively track and control changes to a collection of related entities.

Version Control is the maintenance of a history of revisions in such a way that they can be compared to each other and a copy from an arbitrary revision can be exported at any time.

Developers use version control to coordinate development efforts.

Release Management

Release management is the process by which source code is converted to a final product.

It is the process of managing, planning, scheduling and controlling a software built through different stages and environments; including testing and deploying software releases.

Development teams use release management to produce an output that can be tested or delivered to customers/end-user.

7.4 Need for Software Maintenance

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes.

Software maintenance can consume as much as 90% of the total effort expanded on a system in its lifetime.

Need/importance of software maintenance are mention below:

i) Correct faults (Bug fixing or error free software)
ii) Improve the design and performance
iii) Implement the enhancements capability or adopt to a changing environment
iv) Interface with other systems
v) Providing continuity of service
vi) Supporting mandatory updates
vii) Removal of outdated functions
viii) Retire software

7.5 Types of Software Maintenance

There are four types of software maintenance:

1. Corrective Maintenance:

This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.

2. Adaptive Maintenance:

This includes modifications and updations applied to keep the software product up to date and tuned to the ever changing world of technology and business environment.

3. Perfective Maintenance:

This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.

4. Preventive Maintenance:

This includes modifications and updations to prevent future problems of software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

7.6. Software maintenance process model:

Software maintenance phase includes following phases:

1. Identification & Tracing → Requests for modifications in the software are identified and analysed.

2. Analysis → The cost of modification/maintenance is analyzed and estimation is concluded.

3. Design → The new modules that need to be replaced or modified are designed as per the requirements specified in the earlier stages.

4. Implementation → Actual modification in the software code are made, new features are added, and the modified software is installed.

5. System Testing → Regression testing is done to ensure no defect, error or bug is left undetected. The system is tested as a whole, following regression testing procedures.

6. Acceptance Testing → The system is tested for acceptance with the help of users.

7. Delivery → After succesful acceptance testing, the modified system is delivered to the users.

8. Some software maintenance models are described below:

1. Quick-fix Model:

There is an adhoc approach used for maintaining the software system. The objective of the model is to identify the problem and then fix it as quick as possible.

The advantage is that it performs its work quickly and at a low cost.

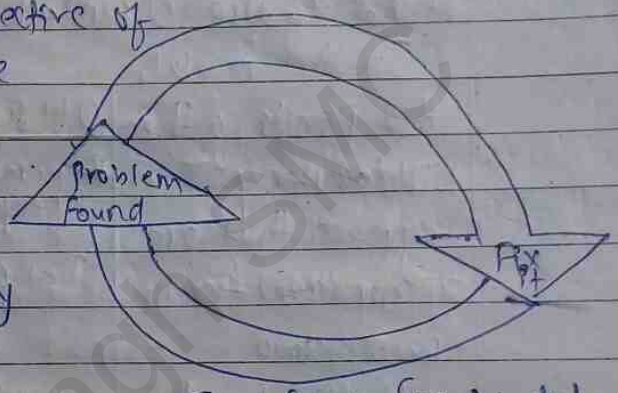


Fig: Quick fix Model

2. Iterative Enhancement Model:

This model considers the changes made to the system are iterative in nature. This model incorporates the changes in the software based on the analysis of the existing system. It attempts to control complexity and tries to maintain good design.

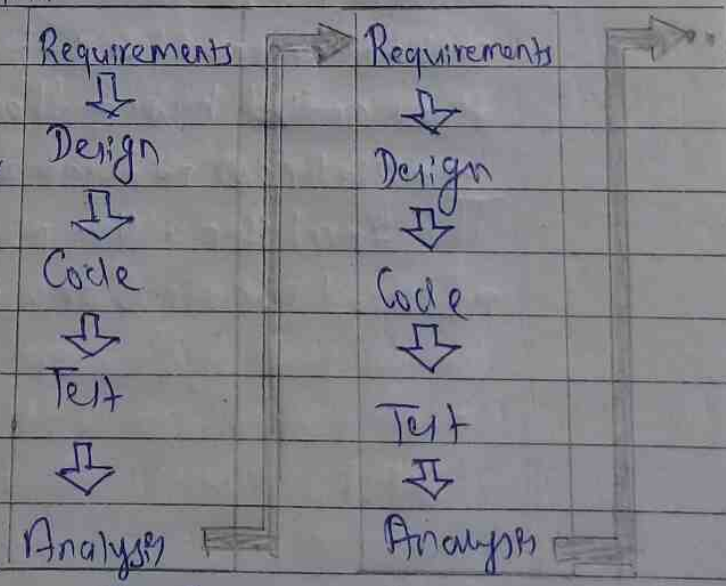


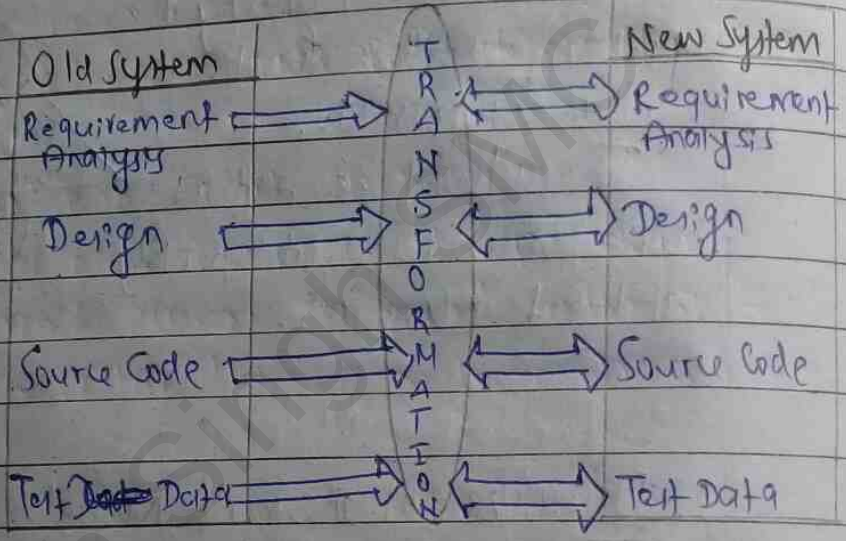
Fig: Iterative Enhancement Model

This model is divided into 3 stages:

i) Analysis of Software System
ii) Clarification of requested modifications
iii) Implementation of requested modifications.

3. The Re-use Oriented Model:

In this model, the parts of the old/existing system that are appropriate for reuse are identified and understood. These parts then go through modification and enhancement, which are done on the basis of the specified new requirements.



The final step of this model is the integration of modified parts into the new system.

7.7 Software maintenance Cost:

Software maintenance cost is derived from the changes made to software after it has been ~~deployed~~ delivered to the end user. Software does not "wear out" but it will become less useful as it gets older, plus there will always be issues within the software itself.

Software maintenance costs will typically form 75% of TCO.

Software maintenance Cost includes:

- Corrective maintenance - costs due to modifying software to correct issues discovered after initial deployment (generally 20% of software maintenance costs).

- Adaptive maintenance - Costs due to modifying a software solution to allow it to remain effective in a changing business environment (25% of software maintenance costs).

- Perfective maintenance - costs due to improving or enhancing a software solution to improve overall performance (generally 5% of software maintenance costs).

- Enhancements - Costs due to continuing innovations (generally 50% or more of software maintenance costs)

Unit - 8

Software re-engineering

Software re-engineering is the examination and alteration of a system to reconstitute it in a new form.

The principles of Re-engineering when applied to the software development process is called software re-engineering. ~~It aff~~

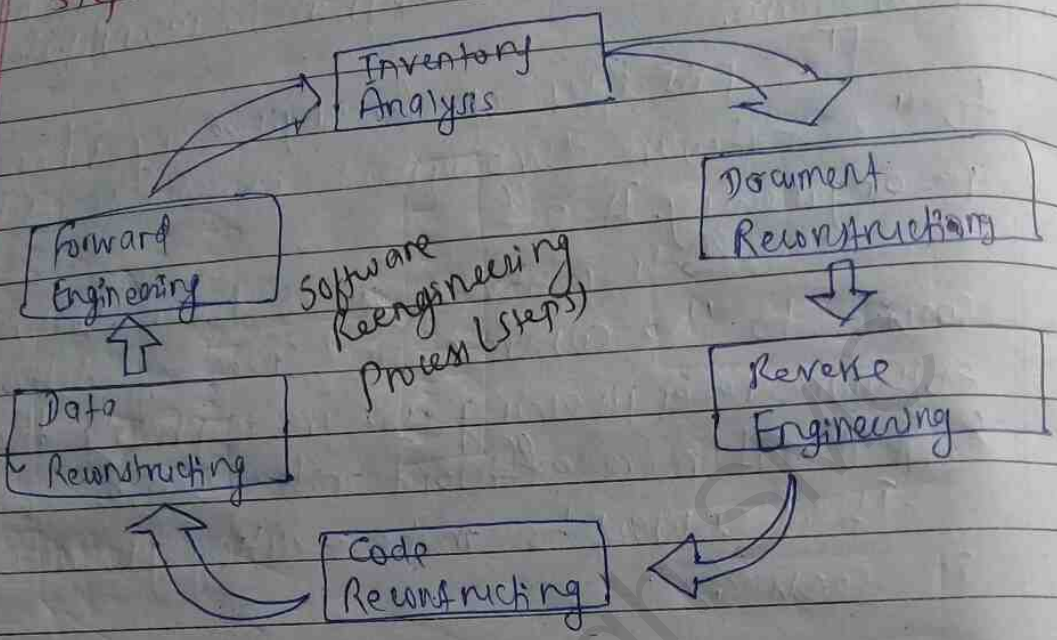
It affects positively at software cost, quality, service to the customer and speed of delivery. In software re-engineering, we are improving the software to make it more efficient and effective.

It is a way to make existing products continue in service.

~~Steps in reengineering~~

Re-engineering is the reorganizing and modifying existing software systems to make them more maintainable.

8.1. Steps in reengineering



1. Inventory Analysis

Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.

2. Document Reconstructing:

Documentation of a system either explains how it operate or how to use it.

- Document must be updated.
- The system is business critical and must be fully re-documented.

3. Reverse Engineering:

It is a technique used to analyze software in order to identify and understand the parts it is composed of.

Reverse engineering is a process of design recovery. Reverse engineering tools extracts data, architectural and procedural design information from an existing program.

4. Code Reconstructing:

. To accomplish code reconstructing, the source code is analyzed using a reconstructing tool.

The resultant ~~reconstructed~~ restructured code is reviewed and tested to ensure that no anomalies have been introduced.

5. Data Restructuring:

Data restructuring begins with the reverse engineering activity.

Data objects and attributes are identified, and existing data structure are reviewed for quality.

It is process of analyzing and reorganizing data in a system to make it more understandable.

6. Forward Engineering:

Forward engineering is a method of creating or making an application with the help of given requirements.

It is a process of building from high-level model or concept to build in complexities and lower-level details.

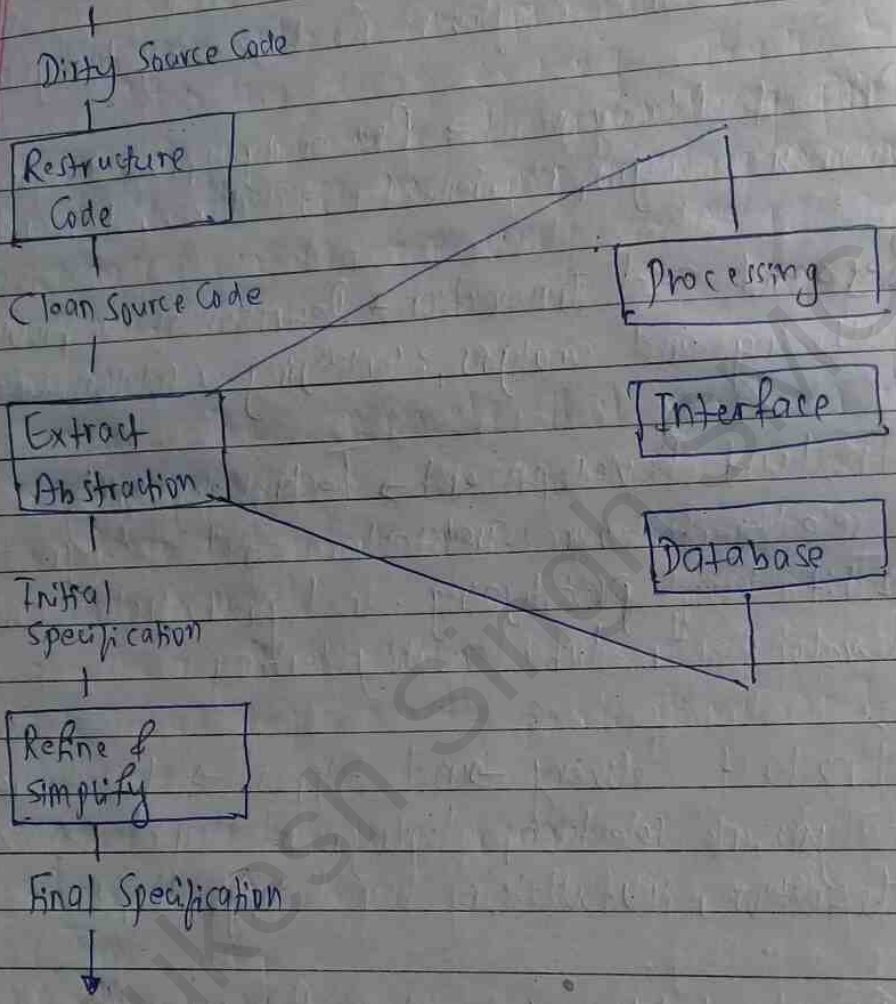
8.2 Re-engineering Process

8.3. Software re-engineering ~~lifecycle and~~ process model

The Complete software Re-Engineering lifecycle includes:

- Product Management → Risk analysis, root cause analysis, business analysis, requirement elicitation and management.
- Research and Innovation → Definition of a problem, data gathering and analysis, identifying a solution. ~~and~~
- Product Development → Technology analysis and selection, software architecture and design, data architecture, prototyping and production code development, data quality testing.
- Product delivery and Support → Hardware/ platform analysis and selection, deployment and release procedures definition, installations and upgrades etc.
- Product Management → Bring efficiency and productivity to your software re-engineering project by utilizing modern, practical software project management, Software quality assurance, data quality assurance, and advanced risk management.

8.5 Reverse Engineering Process



Process

- Initially, the dirty source code or unstructured ~~code~~ source code is taken and processed and code is restructured.
- After restructuring process, the source code becomes clean source code.

- The core to reverse engineering is an activity called extract abstractions.
- In abstraction activity, the engineer must evaluate older program and extract information about procedures, interface, data structure or database used.
- The output of reverse engineering process is a clear, unambiguous final specification obtained from unstructured source code.
- The final specification helps in easy understanding of source code.

8.8. Software reuse

Software reuse is the process of creating software systems from predefined software components.

It is the use of existing software to build new software following the reusability principles.

It is the process of implementing or updating software systems using existing software assets.

Some of the components that can be reused are:

- Source Code
- Design and interfaces
- User manuals
- Software Documentation
- Software requirements specifications and many more.

Advantages of software reuse:

- Less effort
- Time-saving
- Reduce cost
- Less rework
- Increase software productivity
- Utilize fewer resources
- Leads to a better quality software

8.7. Difference between reverse, forward and re-engineering

	Forward Engineering	Reverse Engineering
1.	The application are developed with the given requirements.	The information are collected from the given application.
2	It is high proficiency skill.	2 It is low proficiency skill.
3.	It takes more time to develop an application.	3. It takes less time to develop an application.
4	The nature of forward engineering is prescriptive.	4 The nature of reverse engineering is adaptive.
5.	The production is started with given requirements.	5. The production is started by taking existing product.
6.	Example: Construction of electronic kit, construction of DC Motor. etc	6 Example: Research on instruments.

Unit-9
Software Testing, and quality Assurance

9.1. Software Testing principle

Software testing is a process of executing a program with the aim of finding the error. To make our software perform well it should be error free. If testing is done successfully it will remove all the errors from the software.

There are seven principles in software testing:

1. Testing shows the presence of defects:
The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Testing can reduce the number of defects but not remove all defects.

2. Exhaustive testing is not possible
It is the process of testing the functionality of a software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing.
Exhaustive testing is impossible means the software can never test at every test cases. If the software will test ~~ad~~ every test cases then it will take more cost, effort, etc. and which is impractical.

3. Early testing
To find the ~~error~~ defect in the software, early test activity shall be started. The defect detected in early phase of SDLC will be very less expensive. for better performance of software, software testing will start at initial phase i.e. testing will perform at the requirement analysis phase.

4. Defect Clustering
Defect Clustering means, that ~~a small~~ in a ~~number~~ project, a small number of the module can contain most of the defects.
Pareto Principle to software testing stated that 80% of software defect comes from 20% of modules.

5. Pesticide Paradox
Pesticide Pardox means ~~to~~ repeating the same test cases again and again will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

6. Testing is Context dependent
Testing approach depends on context of software developed. Different types of software need to perform different types of testing. for example, the testing of the e-commerce site is different

from the testing of the android application.

7. Absence of errors fallacy:
If 99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and software is not addressing the business needs. It is not necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

9.2. Software testing approach

A test approach is the test strategy implementation of a project, defines how testing would be carried out.

Test approach has two techniques:

- Proactive - An approach in which the test design process is initiated as early as possible in order to find and fix the defects before the build is created.

- Reactive - An approach in which the testing is not started until after design and coding are completed.

There are many strategies that a project can adopt depending on the context and some of them are:

- Dynamic and heuristic approaches
- Consultative approaches
- Model-based approach that uses statistical information

about failure rates.
- Approaches based on risk-based testing where the entire development take place based on the risk.
- Methodical approach, which is based on failures.
- Standard-compliant approach specified by industry-specific standards.

9.3. Unit, integration and system testing

i) Unit testing

Unit testing is the practice of testing small pieces of code, typically individual functions, alone and isolated.

A unit is the smallest testable portion of system or application which can be compiled, linked, loaded, and executed. This kind of testing helps to test each module separately. It checks that components are fulfilling functionalities or not. This kind of testing is performed by developers.

ii) Integration testing:

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly.

In this testing phase, different software modules are combined and tested as a group to make sure

that integrated system is ready for system testing.
It checks the data flow from one module to other modules. This kind of testing is performed by testers.

iii) System testing:
System testing tests the system as a whole. It tests a completely integrated system to verify that it meets its requirements. It is the bringing together of all programs that a system comprises for testing purpose.

iv) Acceptance testing
Acceptance testing is done a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer.

Unit testing	Integration testing
1. Tests the single component of the whole system i.e. tests a unit in isolation.	1. Tests the system components working together i.e. test the collaboration of multiple units.
2. It is kind of White Box Testing.	2. It is kind of Black Box Testing.
3. Faster to execute.	3. Can run slow.
4. Simple	4. Complex
5. Conducted by developer.	5. Conducted by tester.
6. Can be performed at any time.	6. Usually carried out after unit testing and before system testing.
7. Cheap maintenance.	7. Expensive maintenance.

Mukesh Singh

9.4. Software quality attributes and quality factors

Software quality attributes are non-~~realized~~ functional requirements used to evaluate the performance of a system.

Various software quality attributes and factors are mentioned below:

i) Correctness → Correctness is the extent to which a program satisfies its specifications.

ii) Reliability → It is the property that defines how well the software meets its requirements.

iii) Efficiency → It is the amount of ~~to computing~~ computing time and code required by a program to perform its function. Factor such as response time, memory requirement and throughput comes under efficiency.

iv) Integrity → It is the extent to which access to software and data is denied to unauthorized users.

v) Usability → It is the labour or effort required ~~to~~ to ~~understand, operate in locating and fixing errors in operating programs.~~ operate, prepare input and interpret output of a program.

ii) Maintainability → It is the effort required to locate and fix an error in a program.

iii) Flexibility → Effort needed to modify an operational program.

iii) Testability → Effort required to test the programs for their functionality.

ix) Portability → Effort required to run the program from one platform to other or to different hardware.

x) Reusability → Extent to which parts of the software can be reused in other related applications.

xi) Interoperability → Effort required to couple one system to another.

Mukesh Singh

9.5 Software Quality Control and Quality Assurance

Software Quality Control (SQC) is a sequence of tasks to ensure the quality of the software by identifying defects and correction of defects in the developed software.

Software Quality Assurance (SQA) is a planned and systematic way of creating an environment to assure that the software product being developed meets the quality requirements.

Software Quality Assurance	Software Quality Control
1. It is a procedure that focuses on providing assurance that quality requested will be achieved.	1. It is a procedure that focuses on fulfilling the quality requested.
2. It is process oriented.	2. It is product oriented.
3. It aims to prevent defects.	3. It aims to identify and fix defects.
4. It is a preventive technique.	4. It is a corrective technique.
5. It is a proactive process.	5. It is a reactive process.
6. It is performed before QC.	6. It is performed only after QA.
7. Verification is example of QA.	7. Validation is the example of QC.

9.6 Software Safety

Software safety refers to software that has as its primary purpose improves the safety of an organization through the more efficient management of its safety protocols.

Safety software allows organizations to standardize their safety procedures and track, analyze, and optimize safety related activities more efficiently.

9.7. The ISO 9000 model

ISO 9000 is a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements to be implemented to maintain an effective quality system.

OR, ISO 900 is a series of standards, developed and published by the ISO, that define, establish, and maintain an effective quality assurance system for manufacturing and service industries.

It is designed to help ~~the~~ organizations ensure that they meet the needs customers and other stakeholders while meeting statutory and regulatory requirements related to a product or service.

2.8. SEI Capability Maturity Model

CMM is a methodology used to develop and refine organizations software development process. It was developed and is promoted by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DoD).

It consists of 5 levels of maturity:

Level 1: Initial → Processes followed are ad hoc and immature and are not well defined. It is unstable environment for software development.

Level 2: Repeatable → It focuses on establishing basic project management policies.

Level 3: Defined → Documentation of standard guidelines and procedures take place in this phase.

Level 4: Managed → Quantitative quality goals are set for the organization for software products as well as software process.

Level 5: Optimizing → It is the highest level of process maturity. It focuses on continuous process improvement and process performance using feedback.

9.9. Verification and Validation

Verification	Validation
1. Are we building the system right?	1. Are we building the right system?
2. Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	2. Validation is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
3. Verification is carried out before validation.	3. Validation is carried out just after the verification.
4. Cost of errors caught is less.	4. Cost of errors caught is more.
5. It does not involve execution of code.	5. It always involves execution of code.
6. It is human based checking of documents & files.	6. It is Computer based ~~checking~~ of execution of program.
7. Activities: Reviews, meetings and inspections.	7. Activities: Testing like black box testing, whit box testing, grey box testing etc