

# Software Engineering

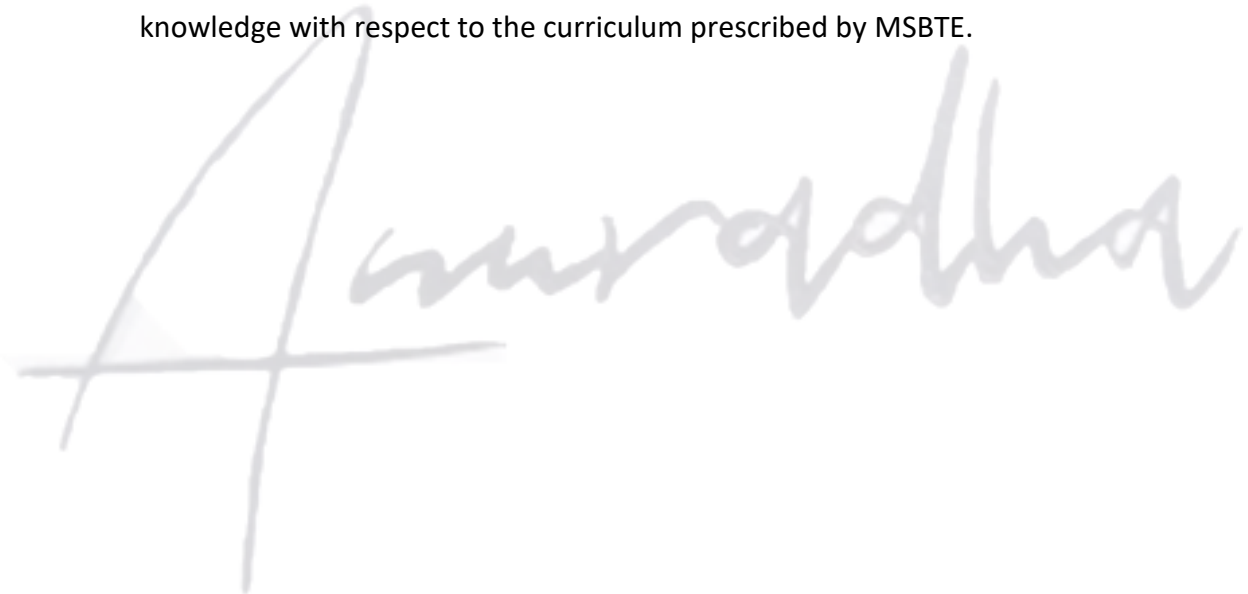
*Anuradha*  
B H A T I A

## Disclaimer

The content of the book is the copyright property of the author, to be used by the students for the reference for the subject “Software Engineering” for Fifth Semester Computer Engineering and Information technology, MSBTE.

The complete set of the e-book is available on the author’s website <https://github.com/anuradhabhatia/Notes>, and students are allowed to download it free of charge from the same.

The author does not gain any monetary benefit from the same, it is only developed and designed to help the teaching and the student fraternity to enhance their knowledge with respect to the curriculum prescribed by MSBTE.

A large, faint, light gray signature of the author, Anuradha Bhatia, is visible in the background of the page. The signature is written in a cursive style and spans across the middle of the page.

# Overview of Software Engineering and the Software Development Process

## CONTENTS

- I. Definition of Software and Characteristics of Software
- I. Types / Categories of Software
  - 1. System Software
  - 2. Real-time Software
  - 3. Business Software
  - 4. Engineering and Scientific Software
  - 5. Embedded Software
  - 6. Personal Computer Software
  - 7. Web-based Software
- II. Software Engineering –
  - 1. Definition
  - 2. Need
- III. Relationship between System Engineering and Software Engineering
- IV. Software Engineering – A Layered Technology Approach
- V. Software Development Generic Process Framework
  - 1. Software Process

2. Software Product
3. Basic Framework Activities
4. Umbrella Activities.

### VI. Personal and Team Process Models (PSP and TSP)

1. Concept
2. Significance with respect to On-going Process Improvement
3. Goals
4. List of Framework Activities

### VII. Perspective Process Models

1. The Waterfall Model (Nature, Situations in which applicable with example, Associated Problems)
2. The Incremental Model (Nature, Situations in which applicable with example, General Steps, Drawbacks)
3. RAD Model (Nature, Situations in which applicable with example, General Steps, Drawbacks)
4. Prototyping (Nature, Situations in which applicable with example, General Steps, Drawbacks)
5. Spiral Model (Nature, Situations in which applicable with example, General Steps, Advantages, Disadvantages)

### VIII. Agile Software Development

1. Differences between Perspective and Agile Process Model
2. Features of the Agile Software Development Approach
3. Concept of Extreme Programming

# I. Definition of Software and Characteristics of Software

## 1. Definition of Software

Software is defined as

1. Instructions (computer programs) that when executed provide desired function and performance.
2. Data structures that enable the programs to adequately manipulate information.
3. Documents that describe the operation and use of the programs.

## 2. Characteristics of Software

***(Question: Define software. Describe characteristics of software. – 3 Marks)***

Software is a logical rather than a physical system element. Software has characteristics that are considerably different than those of hardware:

### i. Software is developed or engineered, it is not manufactured in the classical sense.

***(Question: Justify your answer that, software is developed or engineered and not manufactured. – 3 Marks)***

1. Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
2. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
3. Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different.
4. Both activities require the construction of a "product" but the approaches are different.
5. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

### ii. Software doesn't "wear out."

***(Question: Justify the implication that the software doesn't wear out but deteriorate. – 3 Marks)***

1. Figure 1.1 explains the failure rate as a function of time for hardware.

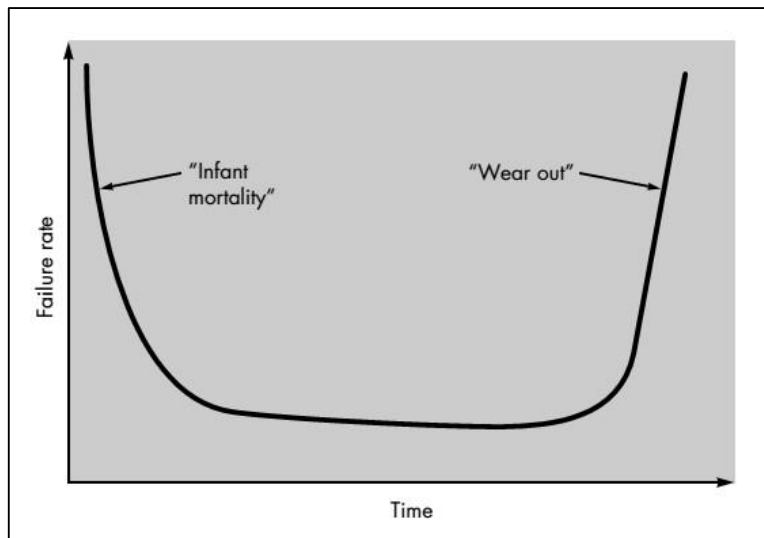


Figure 1.1

2. The relationship, called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life; defects are corrected and the failure rate drops to a steady-state level, ideally, quite low for some period of time.
3. With time the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.
4. Stated simply, the hardware begins to wear out.
5. Software is not susceptible to the environmental maladies that cause hardware to wear out.
6. The failure rate curve for software should take the form of the "idealized curve" shown in Figure 1.2.
7. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown.
8. The idealized curve is a gross oversimplification of actual failure models for software.
9. The implication is clear—software doesn't wear out. But it does deteriorate!

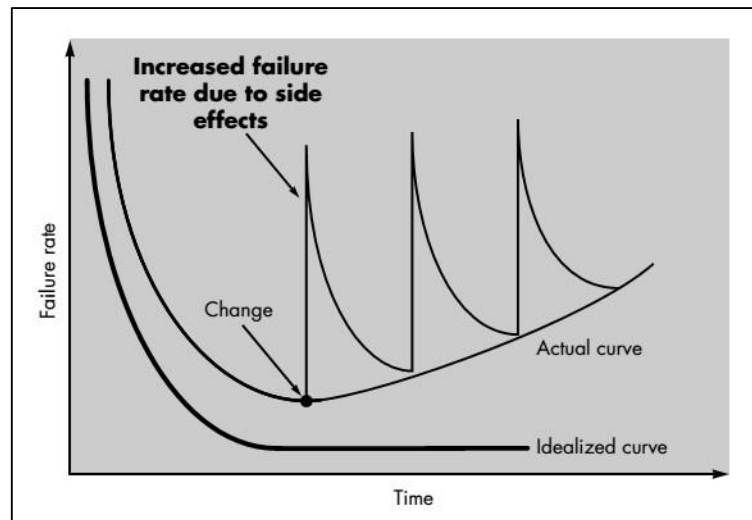


Figure 1.2

**iii. Although the industry is moving toward component-based assembly, most software continues to be custom built.**

1. Consider the manner in which the control hardware for a computer-based product is designed and built. The design engineer draws a simple schematic of the digital circuitry, does some fundamental analysis to assure that proper function will be achieved, and then goes to the shelf where catalogs of digital components exist. Each integrated circuit (called an IC or a chip) has a part number, a defined and validated function, a well-defined interface, and a standard set of integration guidelines. After each component is selected, it can be ordered off the shelf.
2. A software component should be designed and implemented so that it can be reused in many different programs. In the 1960s, we built scientific subroutine libraries that were reusable in a broad array of engineering and scientific applications. These subroutine libraries reused well-defined algorithms in an effective manner but had a limited domain of application. Today, we have extended our view of reuse to encompass not only algorithms but also data structure. Modern reusable components encapsulate both data and the processing applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today's graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction

## II. Types / Categories of Software

### 1. System Software

1. System software is a collection of programs written to service other programs.
2. Few examples of system software are compilers, editors, and file management utilities, process complex, but determinate, information structures.
3. Other systems applications are operating system components, drivers, and telecommunications.

### 2. Real-time Software

***(Question: Explain the features of real world software. – 3 Marks)***

1. Software that monitors or analyzes or controls real-world events as they occur is called real time.
2. Elements of real-time software include a data gathering component that collects and formats information from an external environment, an analysis component that transforms information as required by the application.
3. A control/output component that responds to the external environment, and a monitoring component that coordinates all other components so that real-time response can be maintained.

### 3. Business Software

1. Business information processing is the largest single software application area. Discrete "systems".
2. For example: payroll, accounts receivable/payable, inventory have evolved into management information system (MIS) software that accesses one or more large databases containing business information.
3. Applications in this area restructure existing data in a way that facilitates business operations or management decision making.
4. In addition to conventional data processing application, business software applications also encompass interactive computing.

### 4. Engineering and Scientific Software

1. Engineering and scientific software have been characterized by "number crunching" algorithms.
2. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.
3. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms.



4. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

#### **5. Embedded Software**

1. Intelligent products have become commonplace in nearly every consumer and industrial market.
2. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
3. Embedded software can perform very limited and esoteric functions, for example: keypad control for a microwave oven.
4. To provide significant function and control capability, for example: digital functions in an automobile such as fuel control, dashboard displays, and braking systems.

#### **6. Personal Computer Software**

1. The personal computer software market has burgeoned over the past two decades.
2. Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business fi applications, external network, and database access are only a few of hundreds of applications.

#### **7. Web-based Software**

1. The Web pages retrieved by a browser are software that incorporates executable instructions and data.

### **III. Software Engineering**

#### **1. Definition:**

Software Engineering is the study and application of engineering to the design, development, and maintenance of software. Software engineering can be divided into ten sub disciplines. They are:

1. **Software requirements:** The elicitation, analysis, specification, and validation of requirements for software.
2. **Software design:** The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.
3. **Software construction:** The detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging.
4. **Software testing:** The dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.

5. **Software maintenance:** The totality of activities required to provide cost-effective support to software.
6. **Software configuration management:** The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle.
7. **Software engineering management:** The application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
8. **Software engineering process:** The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.
9. **Software engineering tools and methods:** The computer-based tools that are intended to assist the software life cycle processes, see Computer Aided Software Engineering, and the methods which impose structure on the software engineering activity with the goal of making the activity systematic and ultimately more likely to be successful.
10. **Software quality:** The degree to which a set of inherent characteristics fulfills requirements.

### 2. Need of Software Engineering

1. Computer information and control systems are increasingly embedded and integrated into the fabric of human society.
  2. Computer systems are not standalone, but are usually just component parts of much larger, complex systems involving hardware, software, people, and all the unpredictable events in the natural world.
  3. Our very lives depend on these interdependent systems working reliably all the time.
  4. Most people, even most computer science graduates, are not fully aware of both the difficulty involved in building such complex systems and the essential need for those building them to be equipped with advanced techniques not taught in ordinary computer programming courses.
  5. Software Engineering is the discipline dedicated to the principles and techniques required for the sound construction of the computer systems of today and tomorrow.
- A software engineer must be equipped with techniques to
- i. Model and understand complex interactive systems.
  - ii. Identify how computer information systems can be made to improve such systems.
  - iii. Manage the construction of the information system components.

- iv. Ensure that procedures are in place for the continual testing and maintenance of operational systems.

## IV. Relationship between System Engineering and Software Engineering

**(Question: What is Software engineering? How it differs from system engineering? – 3 Marks)**

1. A software engineer works on the software, while a system engineer will just work on system.
2. They are different because the system could be the BIOS, operating system, hard drive, memory, and other things. While a software engineer will only work on the programs that the user uses.
3. Software engineering is a branch to make software this is a continuous task, while software system is a product which is made by the software engineering.

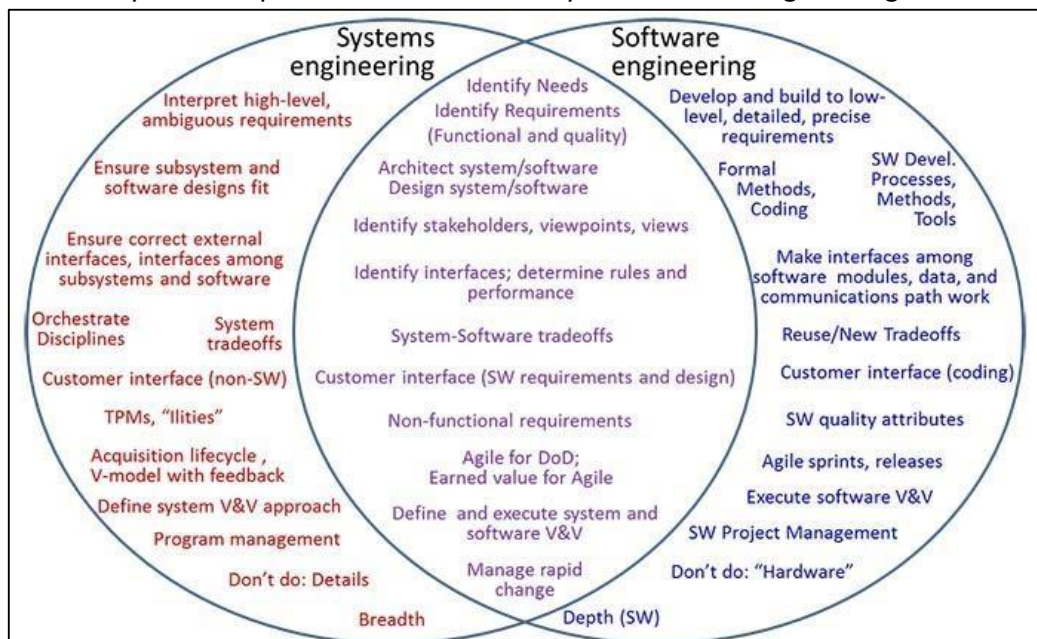


Figure 1.3

## V. Software Engineering – A Layered Approach

**(Question: Explain software engineering as a layered approach? – 6Marks)**

Software engineering deals with process, methods their implementation tools and finally the quality of the product. This is known as the layered approach of software engineering as shown in Figure 1.4:

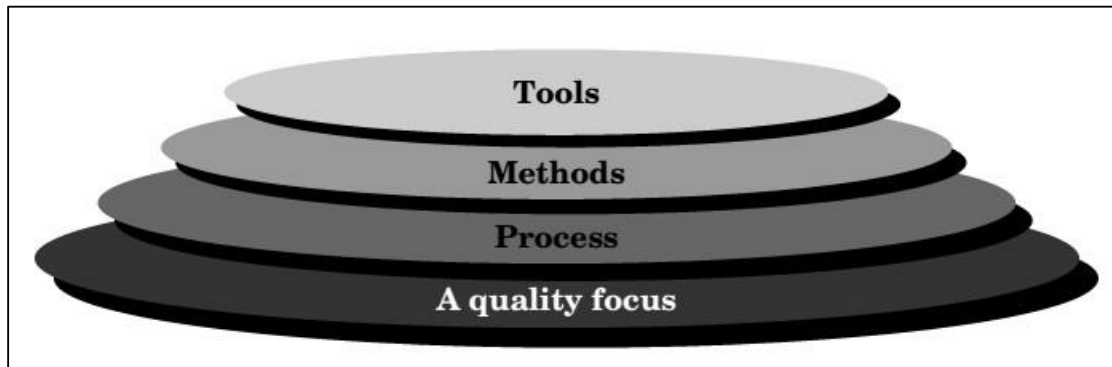


Figure 1.4

### **1. Process, Methods and Tools**

1. Software engineering is a layered technology. Referring to Figure 1.4, any engineering approach must rest on an organizational commitment to quality.
2. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering.
3. The bedrock that supports software engineering is a quality focus.

### **2. Process Layer**

1. The foundation for software engineering is the process layer.
2. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
3. Process defines a framework for a set of key process areas that must be established for effective delivery of software engineering technology.

### **3. Method Layer**

1. Software engineering methods provide the technical how-to's for building software.
2. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.

### **4. Tools Layer**

1. Software engineering tools provide automated or semi-automated for the process and the methods.

2. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established. CASE combines software, hardware, and a software engineering database.

## VI. Software Development Generic Process Framework

1. **Software Process:** A software process can be characterized as shown in Figure 1.5. A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
2. **Software Product:** A number of task sets—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
3. **Basic Framework Activities:** Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement to overlay the process model.
4. **Umbrella Activities:** Umbrella activities are independent of any one framework activity and occur throughout the process.

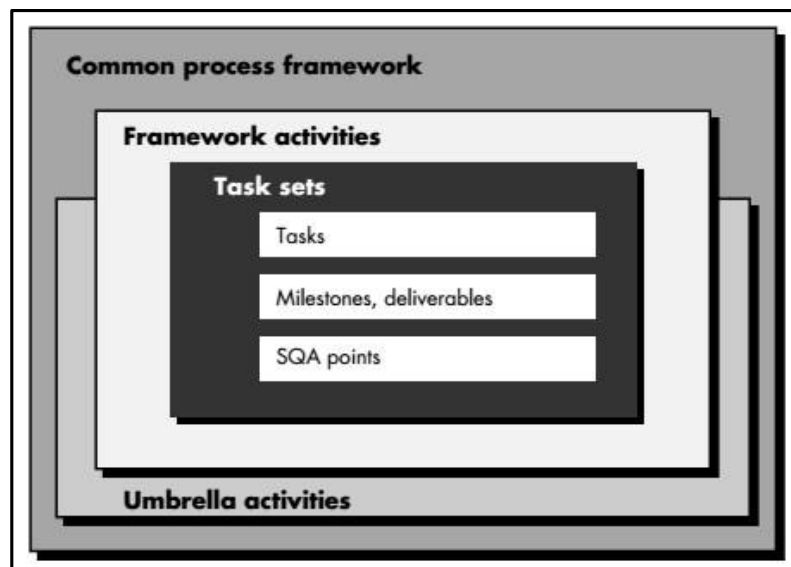


Figure 1.5

1. **Level 1:** Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
2. **Level 2:** Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3. **Level 3:** Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.
4. **Level 4:** Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.
5. **Level 5:** Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

## VII. Personal and Team Process Models (PSP and TSP)

***(Question: Compare PSP and TSP with respect to software engineering - 4 Marks)***

### 1. What is PSP?

1. PSP (Personal Software Process) provide a standard personal process structure for software developers. PSP consists of a set of methods, tables, scripts etc. that serve as a guideline for software developers to plan, measure and manage their work, including how to define their processes and measure quality and productivity.
2. PSP provides software engineers a disciplined methods for improving personal software development process.
3. PSP helps software engineers to
  - i. Improve their estimating and planning skill.
  - ii. Make commitment they can keep.
  - iii. Manage the quality of their product.
  - iv. Reduce the number of defects in their work.
4. PSP represents metrics based approach to software engineering.
5. PSP deals with software engineers to identify the errors early and to understand the types of errors.

### 2. PSP Activities

***(Question: Describe various activities of PSP, - 6 Marks)***

1. **Planning:** This activity isolates requirements and specifications to be decided prior to the development and estimates the size and the cost of the project.
2. **High Level Design Review:** Formal verification methods are applied to uncover errors

3. **Postmortem:** Metrics and measures should be providing guidance for modification.
4. **Development:** The component level design is reviewed and refined. The code is generated, reviewed and tested.
5. **High-Level Design:** External specifications and requirements for each component to be constructed and developed.

### 3. Goals of PSP

The goal of PSP is to provide software engineers with disciplined methods for improving personal software development.

#### **PSP helps software engineers to**

- i. Improve their estimating and planning skill.
- ii. Make commitment they can keep.
- iii. Manage the quality of their product.
- iv. Reduce the number of defects in their work.

### 4. Disadvantage of PSP

1. PSP is intellectually challenging and demands a level of commitment which is always not possible to obtain.
2. Training for PSP is lengthy and costs for the training is high.
3. Required level of measurement is culturally difficult.

### 5. What is TSP?

1. TSP (Team Software Process) is a guideline for software product development teams.
2. TSP focuses on helping development teams to improve their quality and productivity to better meet goals of cost and progress.
3. TSP is designed for groups ranging from 2 persons to 20 persons. TSP can be applied to large multiple-group processes for up to 150 persons.
4. There are 8 steps for implementing PSP and TSP. Each step is focused on solving particular process problems.

### 6. Goal of TSP?

1. To provide improvement guidance to high maturity organization.
2. To facilitate university teaching of industrial grade team skills.
3. To show managers how to motivate and coach their teams and how to sustain peak performance.

### 7. Activities of TSP?

***(Question: Describe various activities of TSP, - 6 Marks)***

1. **Launch:** It reviews course objective and describes the TSP structure and content. It assigns need and roles to the students and describes the customers need statement.
2. **Strategy:** It creates a conceptual design for the product and establishes the development strategy.
3. **Plan:** It estimates the size of each module to be developed. Planning also identifies tasks to be performed, and estimates the time to complete each task.
4. **Requirement:** Analyzes the need statement and interviews the customer, specify and inspect the requirements.
5. **Design:** it creates the high level design, specify the design, inspect the design and develop the integration plan.
6. **Implement:** This uses the PSP to implement the modules and the functions.
7. **Test:** Testing builds and integrates the system.
8. **Postmortem:** Writes the cycle report and produces peer and team review.

## 5. Perspective Process Models

### a. Waterfall Model

***(Question: Describe various phases of waterfall process model with neat diagram, - 6 Marks)***

1. Also known as the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.
2. Figure 1.6 illustrates the linear sequential model for software engineering.
3. Modeled after a conventional engineering cycle, the linear sequential model encompasses the following activities:



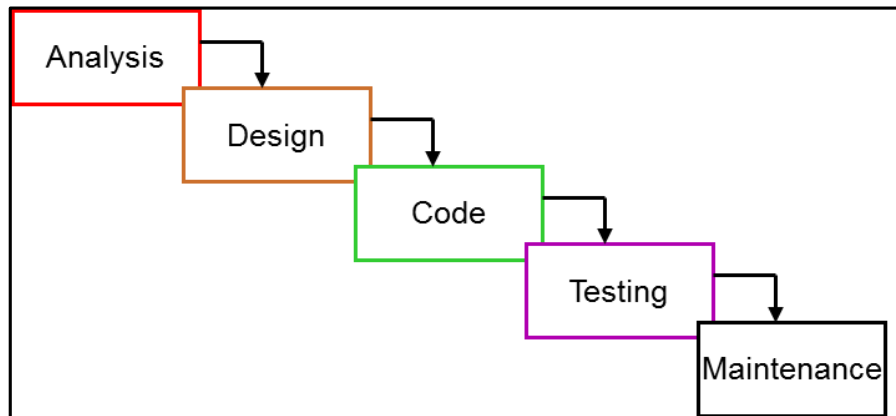


Figure 1.6

4. System/information engineering and modeling. Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software.
5. Phases of waterfall model:
  - i. **Software requirements analysis:** The requirements gathering process is intensified and focused specifically on software.
  - ii. **Design Software:** Design is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.
  - iii. **Code generation:** The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.
  - iv. **Testing:** Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.
6. Advantages
  - i. Organized approach, provides robust separation of phases.
  - ii. Reflects common engineering practice.
7. Disadvantages
  - i. Doesn't cope well with changes the client.
  - ii. Development required by teams might wait for each other.
  - iii. A working version of the product is available only late.

### 8. Applicability

- i. When requirements are well known and few changes are likely to be needed.
- ii. Can be used also for parts of larger software systems.

### b. Incremental Model

**(Question: Describe various phases of Incremental process model with neat diagram, - 6 Marks)**

1. The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping.
2. Figure 1.7, the incremental model applies linear sequences in a staggered fashion as calendar time progresses.
3. Each linear sequence produces a deliverable “increment” of the software.
4. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.
5. It should be noted that the process flow for any increment can incorporate the prototyping paradigm.

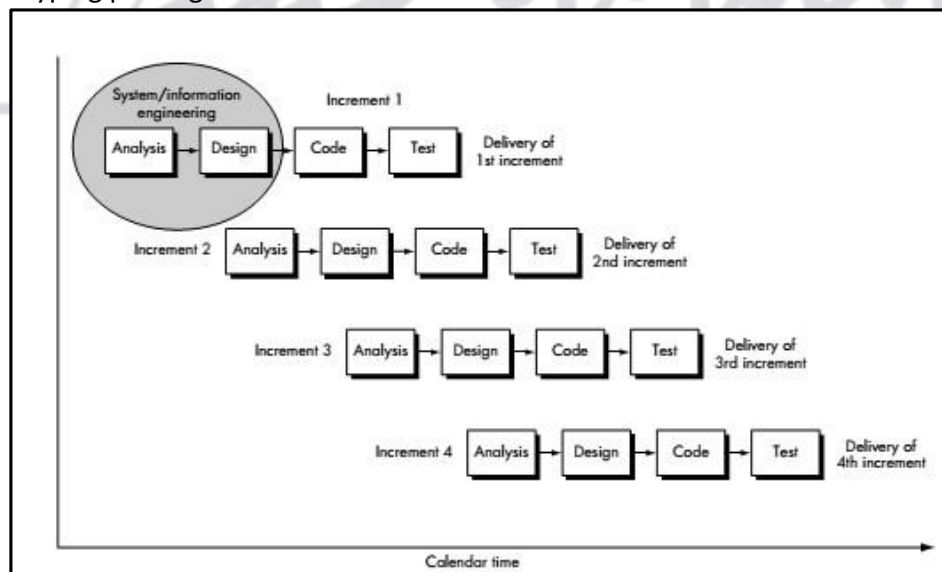


Figure 1.7

6. The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature.
7. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.
8. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.

9. Advantages:

- i. Provides better support for process iteration.
- ii. Reduces rework in the software construction process.
- iii. Some decisions on requirements may be delayed.
- iv. Allows early delivery of parts of the system.
- v. Supports easier integration of sub-systems.
- vi. Lower risk of project failure.
- vii. Delivery priorities can be more easily set.

10. Disadvantages:

- i. Increments need be relatively small
- ii. Mapping requirements to increments may not be easy.
- iii. Common software facilities may be difficult to identify.

11. Applicability:

- i. When it is possible to deliver the system “part-by-part”.

**c. RAD Model**

***(Question: Describe various phases of RAD process model with neat diagram, - 6 Marks)***

1. Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.
2. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction.
3. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days).
4. Data modeling. The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called attributes) of each object are identified and the relationships between these objects defined.
5. Process modeling. The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
6. Application generation. RAD assumes the use of fourth generation techniques. Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary).

7. In all cases, automated tools are used to facilitate construction of the software.
8. Testing and turnover. Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

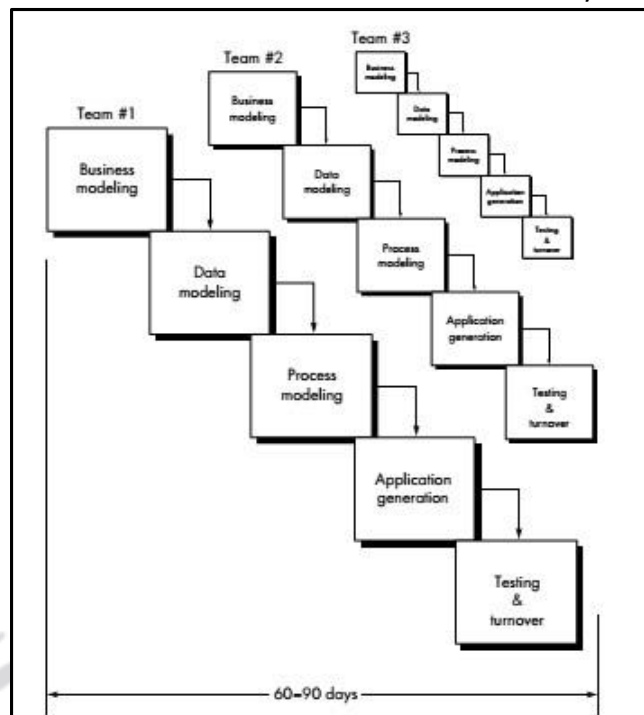


Figure 1.8

### d. Prototyping

**(Question: Describe various phases of prototyping process model with neat diagram, - 6 Marks)**

1. The prototyping paradigm (Figure 1.9) begins with requirements gathering.
2. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
3. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats).

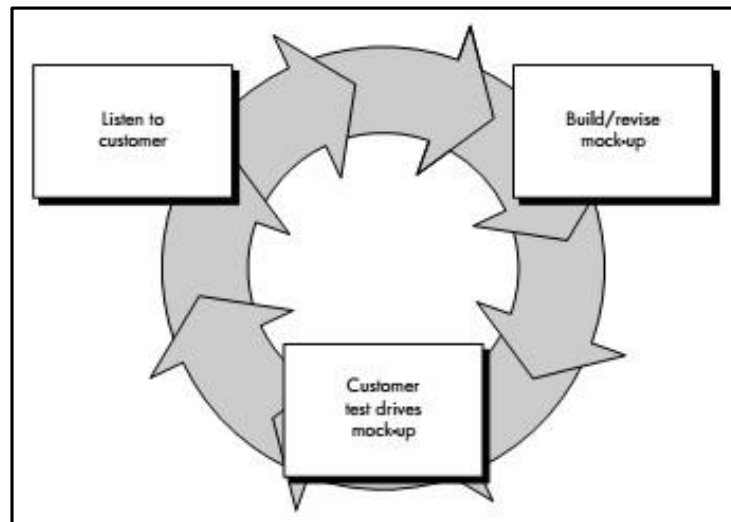


Figure 1.9

4. The quick design leads to the construction of a prototype.
5. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

Prototyping can also be problematic for the following reasons:

1. The customer sees what appears to be a working version of the software, unaware that the prototype is held together "with chewing gum and baling wire," unaware that in the rush to get it working no one has considered overall software quality or long-term maintainability.
2. When informed that the product must be rebuilt so that high levels of quality can be maintained, the customer cries foul and demands that "a few fixes" be applied to make the prototype a working product.
3. Too often, software development management relents.
4. The developer often makes implementation compromises in order to get a prototype working quickly.
5. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.
6. After a time, the developer may become familiar with these choices and forget all the reasons why they were inappropriate.
7. The less-than-ideal choice has now become an integral part of the system.

### e. Spiral Model

**(Question: Describe various phases of Spiral process model with neat diagram, - 6 Marks)**

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Figure 1.10 depicts a spiral model that contains six task regions:

1. **Customer communication**—tasks required to establish effective communication between developer and customer.
2. **Planning**—tasks required to define resources, timelines, and other project related information.
3. **Risk analysis**—tasks required to assess both technical and management risks.
4. **Engineering**—tasks required to build one or more representations of the application.
5. **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).
6. **Customer evaluation**—tasks required to evaluate the product and the project.

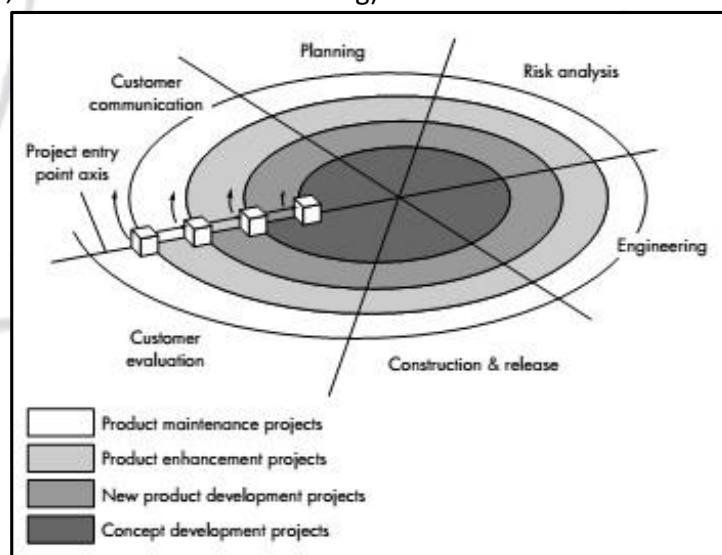


Figure 1.10

#### 6. Advantages:

- i. Risk reduction mechanisms are in place
- ii. Supports iteration and reflects real-world practices
- iii. Systematic approach

#### 7. Disadvantages:

- i. Requires expertise in risk evaluation and reduction
- ii. Complex, relatively difficult to follow strictly
- iii. Applicable only to large systems

#### 8. Applicability:

- i. Internal development of large systems

## 6. Agile Software Development

### a. Difference between Perspective and Agile Process Model

**(Question: Difference between Prescriptive and Agile Process Model - 3 marks)**

1. The Agile movement proposes alternatives to traditional project management.
2. Agile approaches are typically used in software development to help businesses respond to unpredictability.
3. It's easy to the problems with "waterfall" methodology.
4. It assumes that every requirement of the project can be identified before any design or coding occurs.
5. At the end of a project, a team might have built the software it was asked to build, but, in the time it took to create, business realities have changed so dramatically that the product is irrelevant.
6. Today very view organizations openly admit to doing waterfall or traditional command and control. But those habits persist. **Problems with Agile Method:**
  1. It can be difficult to keep the interest of customers who are involved in the process.
  2. Team members may be unsuited to the intense involvement that characterizes agile method.
  3. Prioritizing changes can be difficult where there are multiple stakeholders.
  4. Maintaining simplicity requires extra work.
  5. Contracts may be a problem as with other approaches to iterative development.

### b. Features of the Agile Software Development Approach

**(Question: Give the Features of the Agile Software Development Approach- 6 marks)**

1. Principle	Description
2. Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
3. Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
4. People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
5. Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.

6. Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
------------------------	--

### ***c. Concept of Extreme Programming***

Extreme Programming (XP) takes an 'extreme' approach to iterative development.

1. New versions may be built several times per day;
2. Increments are delivered to customers every 2 weeks;
3. All tests must be run for every build and the build is only accepted if tests run successfully.
4. The extreme programming release cycle

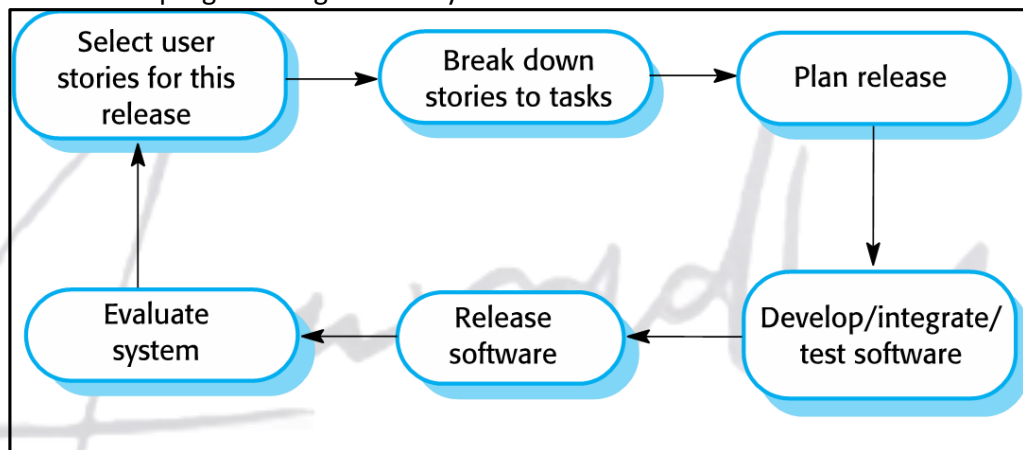


Figure 1.11

### **5. Extreme programming practices**

1. Principle or practice	Description
2. Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
3. Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
4. Simple design	Enough design is carried out to meet the current requirements and no more.



5. Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
6. Refactoring	developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.



# **Software Engineering Requirements and Development of Analysis & Design Models.**

## **CONTENTS**

- I.      Software Engineering Practice
  - 1.   Definition
  - 2.   Importance
  - 3.   Essence
- II.     Core Principles Of Software Engineering
- III.    Communication Practices
- IV.    Planning Practices
- V.     Modelling Practices
  - 1.   Meaning Of Software Modelling
  - 2.   Analysis Modelling

## Software Engineering

---

### 3. Design Of Modelling

## VI. Construction Practices

1. Meaning Of Software Construction
2. Coding
3. Testing

## VII. Software Deployment

1. Meaning Of Delivery Cycle, Support Cycle And Feedback Cycle
2. Deployment Principle

## VIII. Requirement Engineering(RE)

1. Meaning Of RE
2. RE Principle
3. RE Tasks

## IX. SRS (Software Requirements Specifications)

1. Concept of SRS
2. General Format of SRS
3. Need/Importance of SRS

## I. Software Engineering Practice

**(Question: Give the definition and importance of software engineering practice – 4 Marks, 2 marks each)**

- 1. Definition**
- i. Practice is the collection of concepts, principles, methods, and tools that a software engineer calls upon a daily basis.
  - ii. Software engineering is the practice for conducting, implementing, and guidelines for the practice to be implemented.

### 2. Importance

- i. Practice helps us to understand the concepts and the principles that must be followed for development of software engineering projects and project development.
- ii. Practice instructs us to develop the model in more systematic form and at the pace it needs to be developed for deployment.

### 3. The Essence of Practice

**(Question: Explain the Essence of practice of software engineering practice – 8 Marks, 2 marks each)**

The practice involves problem solving, modelling, designing, code generation, testing and quality assurance listed below in four steps.

**i. Understand the problem (Communication and analysis)**

- a) Who has a stake in the solution to the problem?
- b) What are the unknowns?
- c) Can the problems be compartmentalized?
- d) Can the problem be represented graphically?

**ii. Plan a solution (modeling and software design)**

- a) Have you seen similar problems before?
- b) Has a similar problem been solved?
- c) Can sub problems be defined?
- d) Can you represent a solution in a manner that leads to an effective implementation?

**iii. Carry out the plan (code generation)**

- a) Does the solution conform?
- b) Is each component part of the solution probably correct?

**iv. Examine the result for accuracy (testing and quality assurance)**

- a) Is it possible to test each component part of the solution?
- b) Does the solution produce results that conform to the data, function, features, and behavior that are required?

## II. CORE PRINCIPLES

**(Question: Explain the core principles of software engineering practice - 8 Marks)**

**1. The reason it all exists. Provide value to the user**

- i. The software system exists to provide value for the user.
- ii. Before specifying the problem the requirement and the specifications have to be laid down.
- iii. The hardware and the software platform to be decided for implementation.

**2. Keep it simple stupid**

- i. The terms and the design used for development of the project should be kept simple and easily understandable.
- ii. All the terms used should be easy to facilitate the basic concept of the project.

**3. Maintain the vision**

- i. A clear vision is important for the development of a software.
- ii. Compromising the architectural vision of the project weakens the development of the software.
- iii. The developer should hold the vision and ensure the successful development and deployment of the software.

**4. What you reproduce, someone else will have to consume. (implement knowing someone else will have to understand what you are doing)**

- i. Always specify, design and implement knowing that someone else is going to understand what is being developed.
- ii. Customers for the product development is very large.
- iii. Design the data structure and the implementation keeping implementation in mind and the end user.
- iv. Code with the concern that the product has to be implemented and maintained by the end user.

### **5. Be open to the future**

- i. The system designed today should be adaptable to the development and changes in the future at a low cost.
- ii. There should not be much changes to the software to adopt to the new changes in the future development.

### **6. Plan ahead for reuse**

- i. The design and specifications should be developed in such a way that they can be reused for other implementations.
- ii. The code and the design should be well documented for the use in future.

### **7. Think!**

- i. Before designing and implementation a proper thought should be to the end result.
- ii. Proper data structure and the design and implementation strategy should be developed if the software needs modification in the future.

## **III. Communication Practices**

***(Question: Explain the communication principles of software engineering practice – 8 Marks,)***

Communication practice is two way communication between the client and the developer, hence it is also known as requirement elicitation.

### **1. Listen carefully**

- i. To collect lots of data from the client, the developer team has to listen carefully.
- ii. Maximum information with respect to requirement and the specifications should be collected before the implementation and the designing of the software.

### **2. Prepare before you communicate**

- i. A proper agenda or the guidelines for the meetings should be prepared before the start of the meeting.
- ii. Complete detail and the description about the clients and their work area should be gathered to deliver the software up to the best expectation.

### **3. Have a facilitator for any communication meeting**

- i. The requirement gathering and the specification are important for any software development, hence the communication should continue till the requirement gathering is over.

### **4. Face-to-face communication is best**

- i. It is always better to sit across the table and have discussion on the requirement on the software development by the client and the developer.

### **5. Take notes and document decisions**

- i. The important points discussed should also be recorded.
- ii. Proper notes and the documentation is important for the successful completion and deployment of the project.

### **6. Strive for collaboration**

- i. Collaboration in terms of teamwork is required for the successful completion of the software.
- ii. The collective knowledge of the team members should be implemented in the development.

### **7. Stay focused and modularize your discussion**

- i. As the development is the working of many team members, so the possibility of the discussion going from one topic to the other topic is quite possible.
- ii. As a good software developer it is required that the discussion remains focused on the specified area.

### **8. Draw a picture if something is unclear**

- i. Drawing flowcharts, E-R diagrams and other supporting graphical representations give clarity to the discussion and the documentation.

### **9. Move on once you agree, move on when you can't agree, move on if something unclear can't be clarified at the moment**

- i. Healthy discussion leads to the final conclusion of successful implementation of the software.
- ii. Once reached to final statement recorded should move to the next step.
- iii. If no conclusion is reached than that point should be left and move ahead with new implementation which is cost effective.



### **10. Negotiation is not a contest or game**

- i. Negotiation should be mutual not to put someone down or make them feel to be the loser.

## **IV. Planning Practices**

*(Question: Explain the planning practices of software engineering practice – 8 Marks,)*

### **1. Understand the scope**

- i. To plan the development of the software after the communication principles are been laid down, analyze the scope of the software.
- ii. The implementation strategy and its effectiveness is considered and scope is decided.

### **2. Involve the customer in planning**

- i. The client should be involved in the continuous development of the software.
- ii. The requirement and the specifications should not change frequently.

### **3. Recognize that planning is iterative**

- i. Planning is iterative, but the specification should not change continuously.
- ii. After the analysis phase the requirement should be fixed and should not change.

### **4. Estimate based on what you know**

- i. The knowledge of the developer is used for the final estimation of the planning phase of the software.
- ii. New techniques if unknown should not be considered for planning.

### **5. Consider the risk**

- i. The risk in the development should be considered for the software to be with the designing standard and for future scope and development.

### **6. Be realistic**

- i. Considering the latest trends in the development and not choosing something which is not practically possible to implement.

### **7. Adjust granularity as you define the plan**

- i. The specification should be refined as the plan is defined at each stage of development.

### **8. Define how you intend to ensure quality**

- i. The quality of the software should be maintained by providing help at each level of the software.
- ii. Proper documentation should be maintained and given at the time of deployment to the client.
- iii. The end user license and agreement (EULA).should be provided

### **9. Describe how you intend to accommodate change**

- i. The up gradation option to the software should be provided to the client in the future with the regular maintenance policy.

### **10. Always keep track of the plan and make changes as required**

- i. The continuous communication with the client to be done to make the required changes to the planning and the development phase.

## **V. Modelling Practices**

***(Question: Explain the modeling practices of software engineering practice – 8 Marks)***

### **1. Meaning Of Software Modelling**

Software modelling follows the seven W5HH principles for the software development.

- i. Why is the system being developed?
- ii. What will be done?
- iii. When will it accomplished?
- iv. Who is responsible for the function?
- v. Where they are organizationally located?
- vi. How will the job be done technically and managerially?

### **2. Analysis Modelling**

***(Question: Explain the analysis modelling principles of software engineering practice – 8 Marks)***

**i. *The information domain for the problem must be represented and understood.***

- The information that flows into the system and out of the system should be clearly laid down.

**ii. *Functions performed by the software must be defined***

- The software developed should provide direct benefit to the user of the application software.
- Software functions can be described at the different levels of abstraction.
- Software functions transform data that flow into the system.

**iii. *Software behavior must be represented as consequences of external events***

- The behavior of the software is driven by its interaction and nature with the external environment.
- Input data provided by the end user, control data provided by an external system.

**iv. *Models depicting the information, function, and behavior must be partitioned in manner that uncovers detail in a hierarchical fashion***

- Analysis modelling help the user to understand the problem and have the analysis for the start of the solution.

As large problems are difficult to solve, problems are divided into smaller sub problems using divide and conquer.

It is always easy to find solutions to the smaller sub problems than the large systems.

**v. *The analysis task should move from essential information toward implementation detail***

- Describe the problem with the perspective of the end user.
- The essence of the problem is explained without any consideration that how the solution can be developed.

### **3. Design Of Modelling**

**i. *Design should be traceable to the analysis model***

- The analysis model provides the information domain of the problem.
- The design should be able to translate the information about the design, sub problems and the component level design.

**ii. *Always consider the architecture of the system to be built***

- The software development is the skeleton to the product development.
- It effects the implementation with respect to design, data structure, and the manner in which testing can be performed.

**iii. *Data design is as important as algorithm design***

- Data design is an important architectural strategy.
- The data should be designed as the flow of algorithm is designed for the implementation and deployment of the software.

**iv. *Internal and external interfaces must be designed with care***

- Data flows between the modules should be designed with simplicity and processing efficiency.
- A well designed interface makes integration easier.

**v. *User interface design should be tuned to the needs of the end-user and must focus on use of user***

- The user interface is the visible implementation of the software.
- The poor interface always leads to the perception that the software is bad.

- 
- 
- vi. ***Component-level design should be functionally independent***
  - Each sub function or the module developed should focus on the perfect working functionality of that module.
- vii. ***Components should be loosely coupled to one another and to the external environment***
  - Coupling is accomplished through many ways like message passing, component interface and global data.
  - When the level of coupling increases, error propagation increases with overall maintainability of software decreases.
  - Component coupling should be kept as low as possible.
- viii. ***Design models should be easy to understand***
  - The design models are created for easy communication of information to the users about the code
  - The design model should be easily understandable by the tester.
  - If the software design model is difficult to understand and communicate than it is not the effective model.
- ix. ***Design should be developed iteratively***
  - The design work should be open for improvement at the first level.
  - At the final stages it should be easy and creative.

## VI. CONSTRUCTION PRACTICES

***(Question: Explain the construction practices of software engineering practice – 8 Marks,)***

Construction practices comprises of coding and testing principles. The initial focus is on the component level known as unit testing and the other testing are listed below:

- i. **Integration testing:** Conducted as the system is constructed.
- ii. **Validation testing:** That assesses whether requirements have been met for the complete system.

- iii. **Acceptance testing:** That is conducted by the customer in an effort to exercise all required features and functions.

### 1. CODING PRINCIPLES

**(Question: Explain the coding principles of software engineering practice – 3 Marks)**

The principles which guide the coding tasks are programming languages, programming styles and programming methods

i. ***Preparation principles: Before you write one line of code, be sure you***

- Understand of the problem you are trying to solve ➤ Understand basic design, principles & concepts.  
Pick a programming language that meets the needs of the software to be build and the environment in which the software will operate. Select a programming environment that provides tools that will make you work simpler and easier.
- Create a set of units that will be applied once the component you code is completed.

ii. ***Coding principles: As you begin writing code, be sure you:***

- Constrain your algorithms by following structured programming practice.
- Consider the use of pair programming.
- Select data structures that will meet the needs of the design.
- Understand the software architecture and create interfaces that are consistent with it.
- Keep conditional logic as simple as possible.
- Create nested loops in a way that makes them easily testable.
- Select meaningful variable names and follow other local coding standards.
- Write code that is self-documenting.
- Create a visual layout that aids understanding.

iii. ***Validation Principles: After you have completed your first coding pass, be sure you:***

- Conduct a code walkthrough when appropriate.

- 
- 
- Perform unit tests and correct errors when you have uncovered.
- Refactor the code.

### 2. TESTING PRINCIPLES

**(Question: Explain the testing principles of software engineering practice - 4 Marks)**

Testing is the process of executing programs with the intent of finding errors.

**i. All tests should be traceable to customer requirements:**

- The main objective of software testing is to uncover errors.
- It follows that the most server defects are those that cause the program to fail to meet its requirements.

**ii. Tests should be planned long before testing begins**

- Test planning can begin as soon as the requirements model is complete.



Detailed definition of test cases can begin as soon as the design model has been solidified.

For this reason all tests can be planned and designed before any code has been generated.

**iii. *The Pareto principle applies to software testing***

- In this context the Pareto principle implies that 80% of all errors uncovered during testing will likely be traceable to 20% of all program components.
- The problem, of course, is to isolate these suspect components and to thoroughly test them.

**iv. *Testing should begin “in the small” and progress towards testing “in the large”***

- The first tests planned and executed generally focus on individual components.
- As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

**v. *Exhaustive testing is not possible***

- The number of path permutations for even a moderately sized program is exceptionally large.
- For this reason, it is impossible to execute every combination of paths during testing.

## VII. SOFTWARE DEPLOYMENT

### 1. Deployment Activities:

- i. **Delivery Cycle:** Each delivery cycle provides the customer and end users with an operational software increment that provides usable functions and features.
- ii. **Support Cycle:** Each support cycle provides documentation and human assistance for all functions and features introduced during all deployment cycles to date.



## Software Engineering

---



- iii. **Feedback Cycle:** Each feedback cycle provides the software team with important guidance that results in modifications to the function, features and approach taken for the next increment.

### 2. Deployment Principles:

*(Question: Explain the deployment principles of software deployment - 4 Marks)*

#### i. **Manage customer's expectations or requirements**

In most cases customer wants more than he/she has stated earlier as his requirements or expectations.

In many cases customer is disappointed, even after getting all his/her requirements satisfied.

- Hence, at the time of software delivery, developer must have skills to manage the customer's expectations and requirements.

#### ii. **Record-keeping mechanism must be established for customer support**

- 'Customer Support' is essential and important factor in deployment phase.
- The support should be well planned and with proper record keeping mechanism.

#### iii. **Provide essential instructions, documentations and manual**

- Actual software project delivery includes all documentations, help files and guidance for handling the software by user.

#### iv. **Assemble and test complete delivery package**

- The customer side must get all supporting and essential help from developer's side.
- For this reason CD with complete assembled and tested delivery package with the following support should be delivered:
  - (i) Necessary supported operational features and help.
  - (ii) Essential manuals required for software troubleshooting.
  - (iii) All executable file of developed software.
  - (iv) Necessary installation procedures.

#### v. **Do not deliver any defective or buggy software to the customer**

- 
- 
- The software should be tested before deployment to the customer.
- The specification should be with the requirement to the customer.

### VIII. REQUIREMENTS ENGINEERING

- ✓ Requirement is a condition possessed by the software component in order to solve a real world problems.
- ✓ Requirement describe how a system should act, appear or perform.
- ✓ IEEE defines a requirement as: "A condition that must be possessed by a system to satisfy a contract specification, standard or other formally imposed document.

#### 1. Principles of Requirement Engineering

*(Question: Explain the principles of requirement engineering – 4 Marks)*

**i. Understand the problem before you start to create the analysis model**

There is a tendency to rush to a solution, even before the problem is understood.

This often leads to elegant software that solves the wrong problem.

**ii. Develop prototypes that enable a user to understand how humanmachine interaction will occur**

- Since the perception of the quality of software is often is based on perception of time "friendliness" of the interface, prototyping (and the interaction that results) is highly recommended.

**iii. Record the origin and the reason for every document**

- This is the step in establishing traceability back to the customer.

**iv. Use multiple views of requirement**

- Building data, functional and behavioral models provides software engineer three different views.
- This reduces the chances of missing errors.

**v. Prioritize the requirements**

- Requirements should be followed for the tight implementation and delivery of the product.

## Software Engineering

---



**vi. Work to eliminate ambiguity**

- The use of more technical reviews should be used for no ambiguity.

### **2. Requirement Engineering Task**

***(Question: Explain the various requirement engineering tasks – 8 Marks)***

The requirement engineering process tasks are achieved through seven distinct functions as shown in Figure 1



## Software Engineering

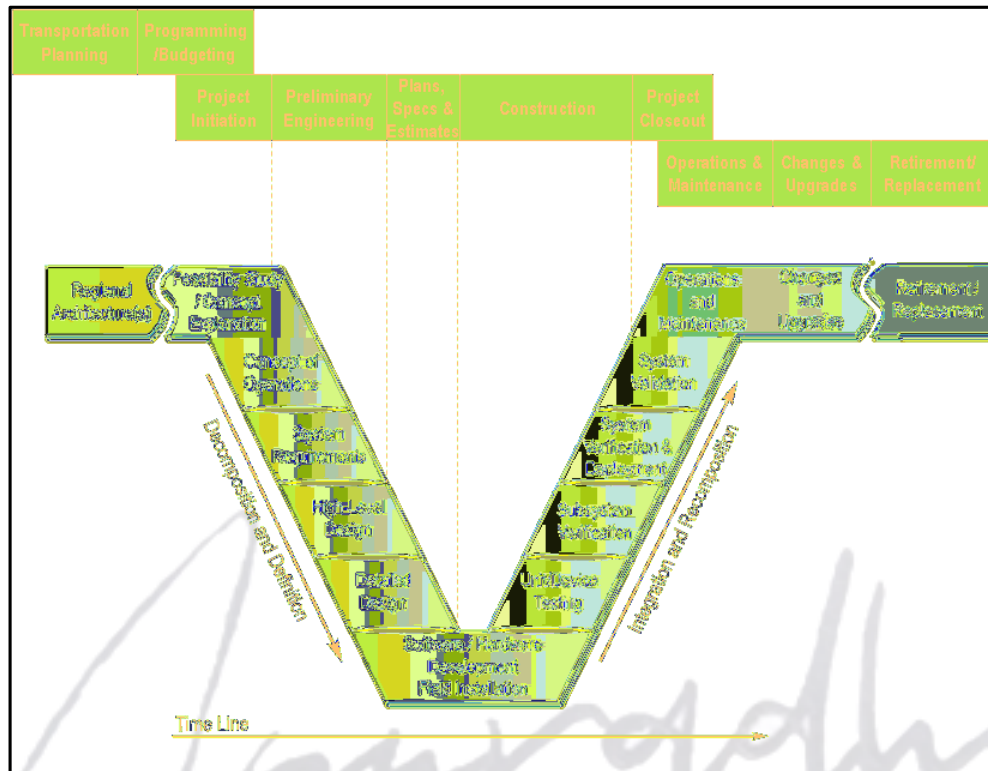


Figure 1: Requirement Engineering

### i. Inception

- Inception is also known as the beginning.
- Inception needs various questions to be answered.
- How does a software project get started?

### ii. Elicitation

- This is the process by which users of the system are interviewed in order to reveal and understand their requirements.
- This sub-phase involves a high level of client input.

### iii. Elaboration

- The most relevant, mission critical requirements are emphasized and developed first.
- This ensures the final product satisfies all of the important requirements, and does so in the most time and cost efficient manner possible.

- 
- Other “should-have” and “nice-to-have” requirements can be relegated to future phases, should they be necessary. **iv.**

### **Negotiation**

- A milestone within that phase may consist of implementing Use Case x, y and z .  
By scheduling the project in this iterative way, our client has a good understanding of when certain things will be delivered and when their input will be required.
- Additionally, features are generally developed in a “vertical” fashion; once Use Case x has been developed, its unabridged functionality can be demonstrated to the client.
- In this way, our client has a clear understanding of where development time is being spent, and potential issues can be caught early and dealt with efficiently.

### **v. Specification**

- This is the process by which requirements and their analyses are committed to some formal media.
- For this project, we would generate four documents during this sub-phase:
- **Use Case document:** a use case is a description of the system’s response as a result of a specific request from a user. Each use case will cover certain requirements discovered during elicitation and analysis. For example, a use case may cover “Adding a New Employee”, or “Generating Site Report”.
- **Interface prototypes:** an interface prototype is a rough representation of certain critical functionality, recorded as anything from a hand-drawn image to an elaborate HTML mockup. The prototypes are completely horizontal in nature; they roughly illustrate how the interface for a certain feature will look and what information is accessible but they will have no back end (vertical) functionality.
- **Architecture Diagram:** an architecture diagram will be generated for developer use as it is a high level view of how the software will be structured. Only if we feel it will help in our understanding of the system will this document be created.
- **Database Diagram:** as with the architecture diagram, a database diagram is generally created for developer use. Through these final



two media, we are beginning the shift from words (requirements) to code (software). **vi. Validation**

- This is the final phase of the requirements engineering process.
- It involves scrutinizing the documents generated during the specification sub-phase and ensuring their relevance and validity.
- This must be done by all interested parties so they are in agreement with the proposed system's behavior, especially in the case of the Use Case document and prototypes. **vii. Management**

As requirements are elicited, analyzed, specified and validated, we will be estimating most features.

- This is an on-going process that will culminate in the delivery of the Project Estimation and Timeline document.
- This will outline the duration of work required to develop and deliver Phase 2.
- As indicated in the Requirements Analysis sub-phase, Phase 2 will likely consist of the majority of the "must-have" features; however it is ultimately up to the client to decide what is most important (even within the "must-have" category), and this will be developed first.

## IX. SRS (Software Requirements Specifications)

### 1. Concept of SRS

*(Question: Explain the concept of SRS with general format – 4 Marks)*

- i. A software requirement specification (SRS) is a comprehensive description of the intended purpose and environment for software under development.
- ii. The SRS fully describes what the software will do and how it will be expected to perform.
- iii. An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost.
- iv. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations.



- v. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

### **2. General Format of SRS**

- i. Functional Requirements Definition**
  - Functional requirements are a subset of the overall system requirement.
  - These requirements are used to consider system behavior.
  - Trade-offs may be between hardware and software issues.

#### **ii. Non-functional Requirements Definition**

- It measures the documentation and the inputs for the various requirements in the system.

#### **iii. System Evolution**

Starting from the specification and the data base schema, on how the system should be designed.

### **3. Need/Importance of SRS**

- i. What will be the business impact of the software?**
  - The development of the software will help the company to grow on the larger perspective.
- ii. What the customer wants exactly?**
  - The communication principles to be followed of what exactly is the requirement of the customer.
- iii. How end user will interact with the system?**
  - Proper documentation and step by step procedure of working module of the project or the software.
- iv. Developed as a joint effort between the developer and the customer**
  - By understanding the complete requirement the project is developed.

# Analysis and Design Modelling

## CONTENTS

- I. Analysis Modelling
  - 1. Concept and need of Analysis Modelling
  - 2. Objectives of Analysis Modelling
- II. Analysis Modelling approaches
  - 1. Structured Analysis (Concept)
  - 2. Object Oriented Analysis (Concept)
- III. Domain Analysis
  - 1. Concept of Technical Domain of the software
  - 2. Concept of Application Domain of the Software
  - 3. Inputs and Output of Domain analysis
- IV. Building the Analysis Model
  - 1. Data Modelling Concepts
  - 2. Flow- Oriented Modelling
    - i. DFD
    - ii. Data Dictionary
    - iii. Creating a Control Flow Model
    - iv. Creating Control Specifications (CSPEC)
    - v. Creating Process Specifications (PSPEC)
  - 3. Scenario- Based Modelling
    - i. Developing Use Cases
      - ii. What is a Use Case?
      - iii. Purpose of a Use Case
    - iv. Use Case Diagram



## I. Analysis Modelling

*(Question: Define Analysis modelling and the analysis principles - 3 Marks each)*

### 1. Definition

- i. The development process starts with the analysis phase.
- ii. This phase results in a specification document that shows what the software will do without specifying how it will be done.
- iii. The analysis phase can use two separate approaches, depending on whether the implementation phase is done using a procedural programming language or an object-oriented language.

### 2. Analysis Principles

- i. The information domain must be represented and understood.
- ii. Models should be developed to give emphasis on system information, function and behavior.
- iii. Models should uncover and give details of all the layers of the development process.
- iv. The function and the problem statement must be defined.
- v. The various analysis models are flow oriented modelling, scenario based modelling, class based modelling, and behavioral modelling.

### 3. Need

*(Question: Explain the need of analysis modelling- 4 Marks)*

- i. Analysis modelling describes the operational and behavioral characteristics.
- ii. Shows the relationship between software interface and other software elements.
- iii. Provides the software developer the representation of the information, function, and behavior.
- iv. Coverts the design into the more descriptive models like use case, ER diagram.
- v. Provides the customer and the developer the means to maintain the quality.

### 4. Objective

- i. Describe what the customer requires.
- ii. Establish a basis for the creation of a software design.
- iii. Devise a set of requirements that can be validated once the software is built.

- iv. Analysis model bridges the gap between system level description and the overall system functionality.

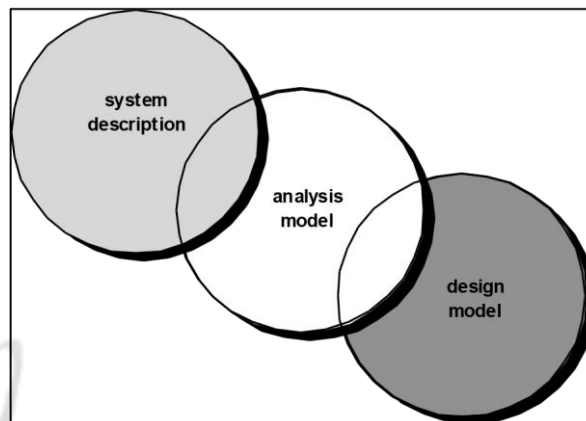


Figure 1. Analysis Model

### 5. Analysis Rules Of Thumb

***(Question: Explain with diagram the structure of analysis modelling  
Or***

***Explain the rules of thumb for analysis modelling – 6 Marks)***

- i. The model should focus on requirements that are visible within the problem or business domain and be written as a relatively high level of abstraction.
- ii. Each element of the analysis model should add to the understanding of the requirements and provide insight into the information domain, function, and behavior of the system.
- iii. Delay consideration of infrastructure and other non-functional models until design.
- iv. Minimize coupling throughout the system.
- v. Be certain the analysis model provides value to all stakeholders.
- vi. Keep the model as simple as possible.
- vii. The Figure 2 shows the structure of analysis modelling.

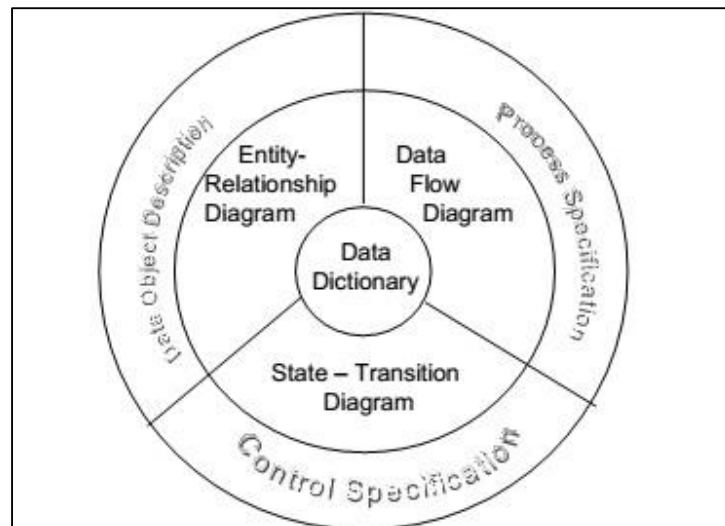


Figure 2. Structure of analysis modelling

## II. Analysis Modeling Approaches

*(Question: Explain the various analysis modelling approaches- 8 Marks)*

### 1. Structured analysis

- i. Considers data and processes that transform data as separate entities.
- ii. Structure analysis is a top down approach.
- iii. It focuses on refining the problem with the help of the functions performed on the problem domain.

### 2. Object-oriented analysis

- i. Focuses on the definition of classes and the manner in which they collaborate to effect the customer requirements.
- ii. Defines the system as a set of objects which interact with each other with the services provided.
- iii. Analyses the problem domain and then partitions the problem with the help of objects.
- iv. The concept of object, attributes, class, operation, inheritance, and polymorphism should be known to work on object oriented modelling.

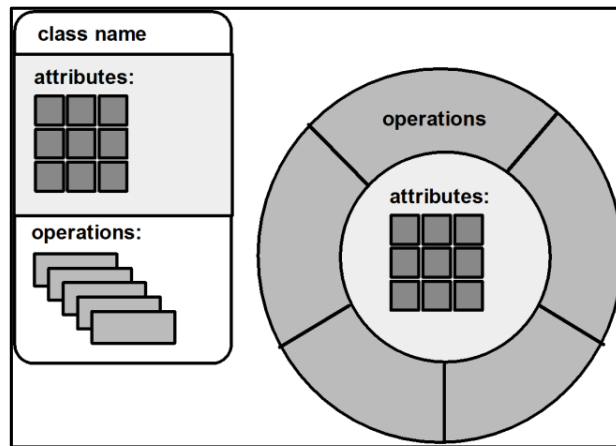


Figure 3: Class Domain

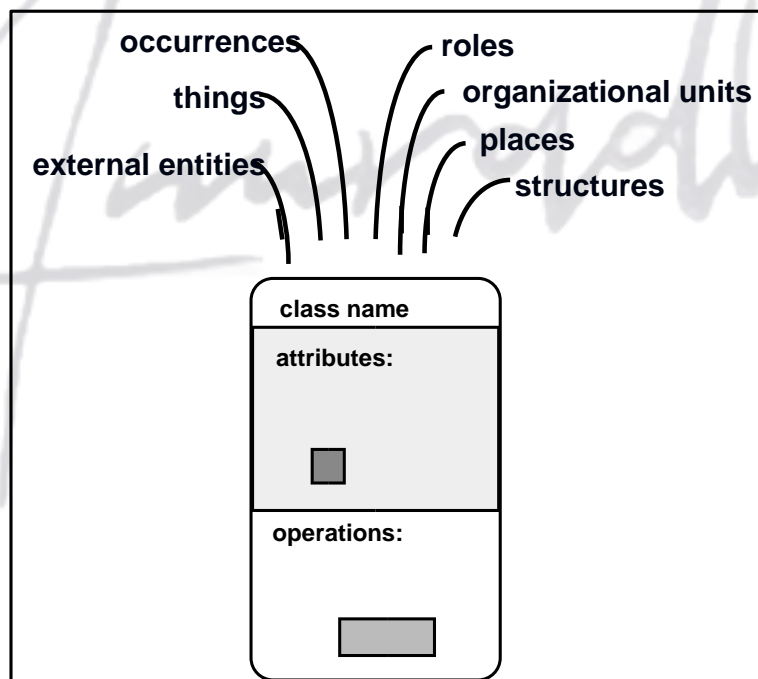


Figure 4: Class, attributes and operations

### III. Domain Analysis

*(Question: What is meant by Domain Analysis in modelling – 4 Marks)*

#### 1. Definition

- i. Domain Analysis is the process that identifies the relevant objects of an application domain.

- ii. The goal of Domain Analysis is Software Reuse.
- iii. The higher is the level of the life-cycle object to reuse, the larger are the benefits coming from its reuse, and the harder is the definition of a workable process.

### 2. Concept and technical application domain of the software

- i. Frameworks are excellent candidates for Domain Analysis: they are at a higher level than code but average programmers can understand them.
- ii. Umbrella activity involving the Identification, analysis, and specification of common requirements from a specific application domain, typically for reuse in multiple projects
- iii. Object-oriented domain analysis involves the identification, analysis, and specification of reusable capabilities within a specific application domain in terms of common objects, classes, subassemblies, and frameworks

### 3. Input and Output Structure of domain analysis

**(Question: Explain the input output structure of domain analysis- 6 Marks)**

- i. Figure 5 shows the flow of the input and the output data in the domain analysis module.
- ii. The main goal is to create the analysis classes and common functions.
- iii. The input consists of the knowledge domain.
- iv. The input is based on the technical survey, customer survey and expert advice.
- v. The output domain consists of using the input as the reference and developing the functional models

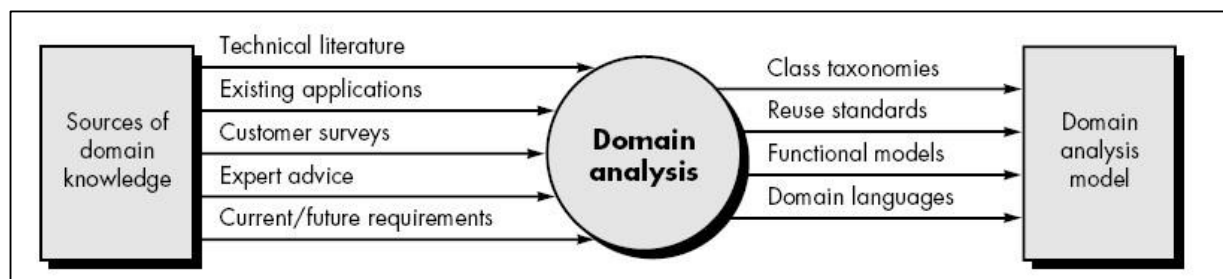


Figure 5: Domain Analysis

## IV. Building The Analysis Model

### 1. Data Modelling Concepts

***(Question: Explain the various data modelling concepts in building the analysis models- 8 Marks)***

- i. Data modeling is the analysis of data objects that are used in a business or other context and the identification of the relationships among these data objects.
- ii. Data modeling is a first step in doing object-oriented programming.
- iii. A data model can be thought of as a diagram or flowchart that illustrates the relationships between data.
- iv. Data modelers often use multiple models to view the same data and ensure that all processes, entities, relationships and data flows have been identified.
- v. There are several different approaches to data modeling, including:  
Conceptual Data Modeling - identifies the highest-level relationships between different entities.
- vi. Enterprise Data Modeling - similar to conceptual data modeling, but addresses the unique requirements of a specific business.
- vii. Logical Data Modeling - illustrates the specific entities, attributes and relationships involved in a business function. Serves as the basis for the creation of the physical data model.
- viii. Physical Data Modeling - represents an application and database-specific implementation of a logical data model.
- ix. Data objects are modeled to define their attributes and relationships.

#### ***a) Data objects (Entities)***

- i. The Figure 6 the relations between the objects and their attributes.
- ii. Data objects are the representation of the most composite information of the system.
- iii. Data object description incorporates the data object and all of its attributes.
- iv. Data objects are all related to each other.

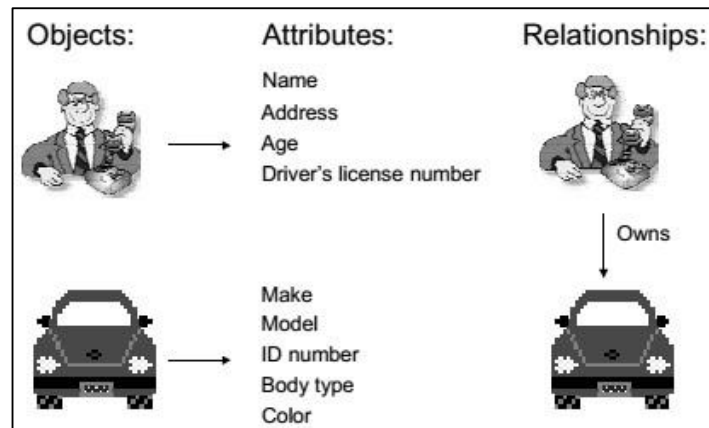


Figure 6: Relation between the object and its attributes

### ***b) Data attributes***

- i. Attributes define the properties of the data object as shown in Figure 3.
- ii. Attributes are used to name the instances of the data. iii. They describe the instance of the data.
- iv. It helps to make reference to the other instance in another table.

### ***c) Relationships***

- i. Data objects are linked with each other in different ways. These links and connections of data objects are known as relationships.
- ii. The two objects person and car are linked with the relationship as person owns the car as shown in Figure 3.

### ***d) Cardinality (number of occurrences)***

- i. Cardinality is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object
- ii. The cardinality is referred to as "one" or "many", One-to-one (1:1), Oneto-many (1: N), Many-to-many (M: N).
- iii. When one instance of object A relates to one instance of object B, its one to one cardinality.

### ***e) Modality***

- i. Modality is 1 if an occurrence of the relationship is mandatory.

- ii. Modality is 0 if there is no explicit need for the relationship to occur or the relationship is optional.
- iii. Each faculty member advises many students, each student has only one advisor.
- iv. Every faculty member may not be advisor, each student must have an advisor

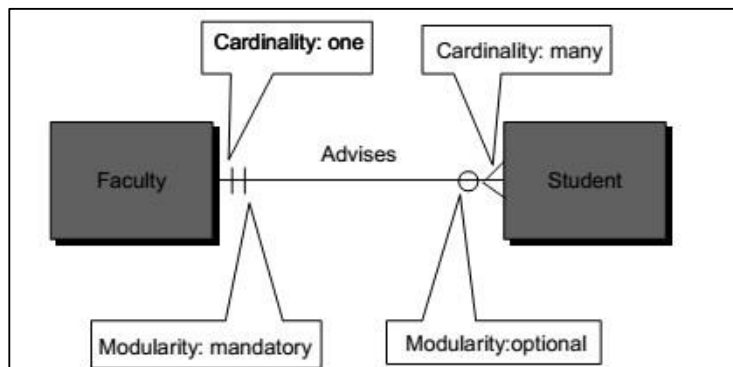


Figure 7: Cardinality and modularity.

## 2. Flow oriented modelling

This represents how the data objects are transformed as they move through the system. The flow modelling provides the view of the system in the graphical approach.

### 1. DFD

**(Question: Explain the concept of DFD- 8 Marks)**

- i. The Data Flow Diagram is a graphical technique that depicts information flow and the transforms that are applied as data move from input to output.
- ii. Can be used at any level of abstraction.
- iii. A level 0 DFD, also called a fundamental system model or context diagrams represents the entire software system as a single bubble with input and output data indicated by incoming and outgoing arrows respectively
- iv. A level 1 DFD might contain five or six bubbles with interconnecting arrows; each of the processes represented at level 1 are sub functions of the overall system depicted in the context model.
- v. Figure 8 depicts the symbols used.



## Software Engineering

- vi. Depicts how input is transformed into output as data objects move through a system.
- vii. Functional modeling and information flow Indicates how data are transformed as they move through the system
- viii. Depicts the functions that transform the data flow. ix. Each function description is contained in a process specification.

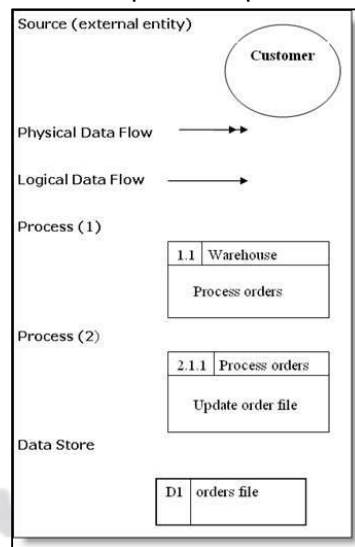


Figure 8: Symbols of DFD

### **Level 0 DFD of a banking system**

**(Question: Draw the Level 0 DFD for banking system - 4 marks)**

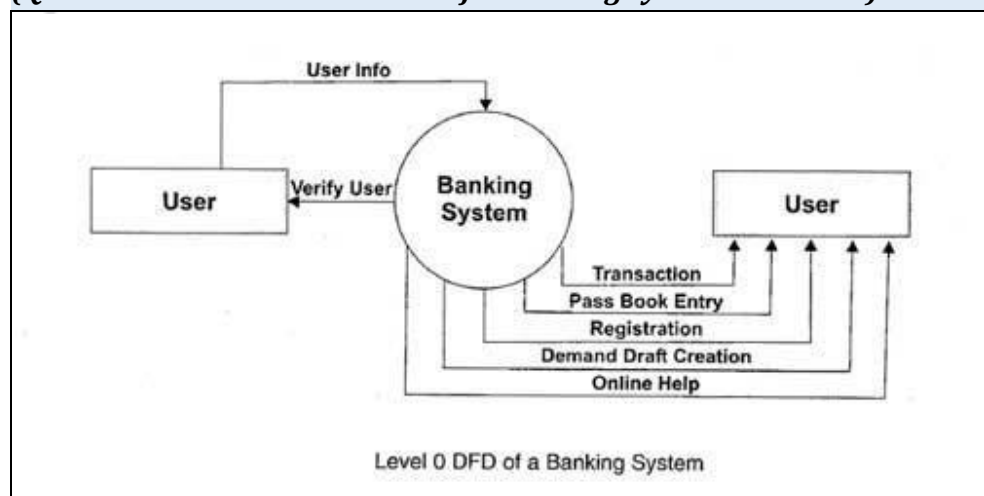


Figure 9: Banking System

***(Question: draw the Level 0 DFD for DVD Rental system- 4 marks)***

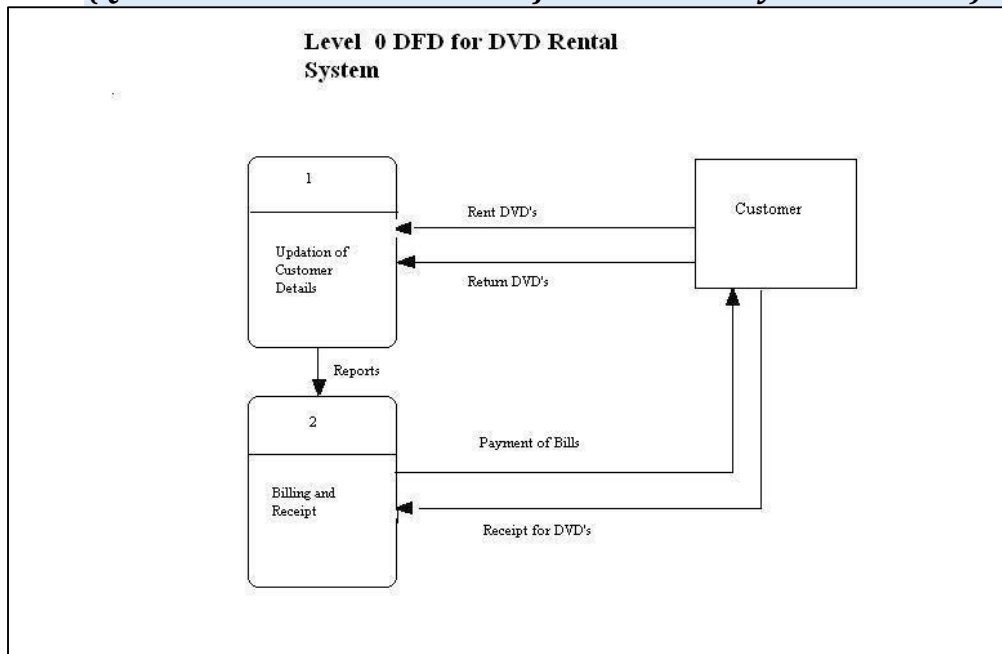


Figure 10: DVD Rental System

***(Question: draw the Level 0 DFD for Control Surveillance System - 4 marks)***

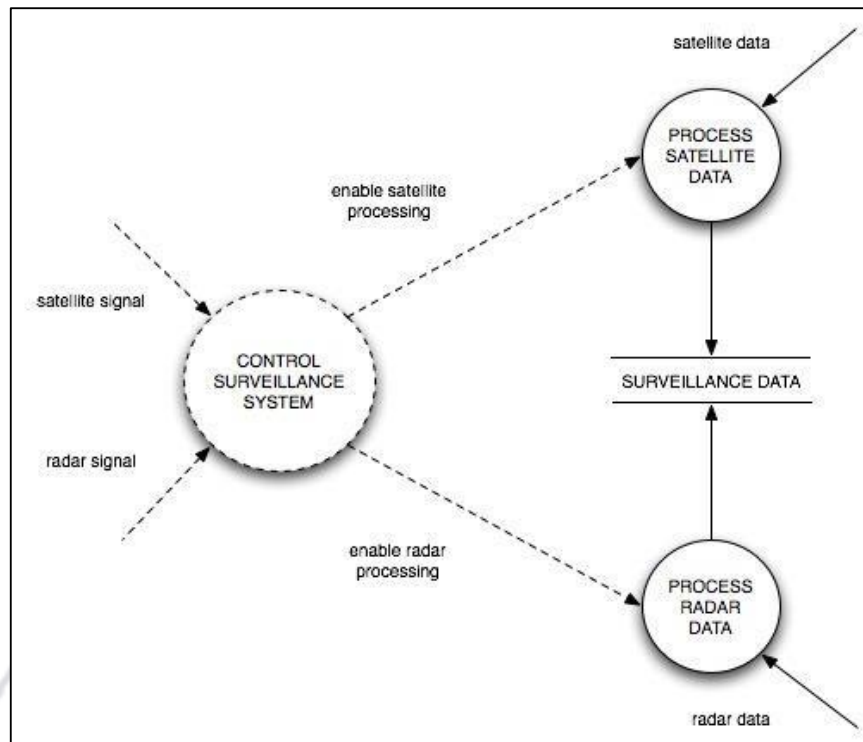


Figure 11: Control Surveillance System

**(Question: Draw the Level 0 DFD for Railway Reservation system - 4 marks)**

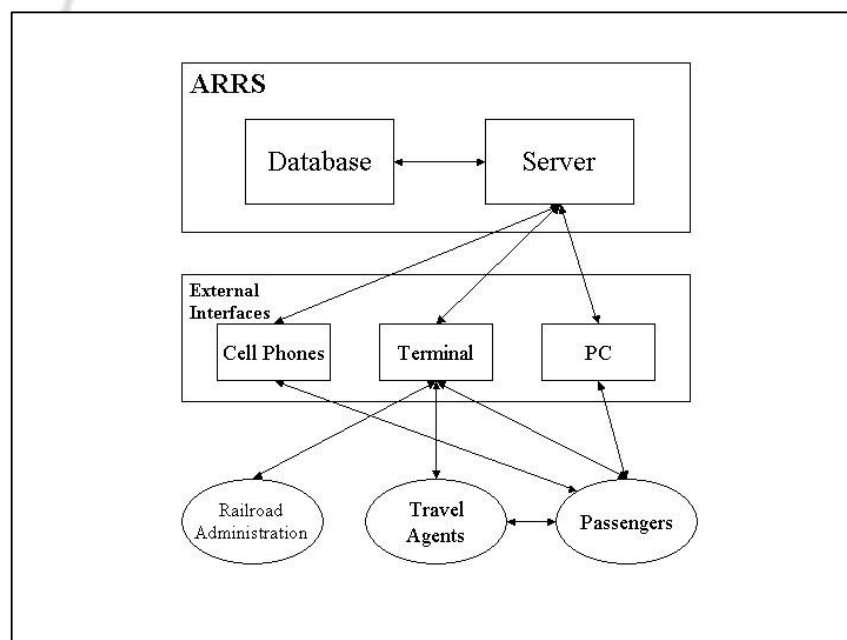


Figure 12: Level 0 DFD of Railway Reservation system

### Rules for DFD

1. **Rule 1** - Does each function have input and output?
2. **Rule 2** - Does each function have all the information it needs in order to produce its output?
3. **Rule 3** - If not, then what information does it need and where will it get that information from?

### LEVEL 1 DFD

(Question: Draw the Level 1 DFD for banking system - 4 marks)

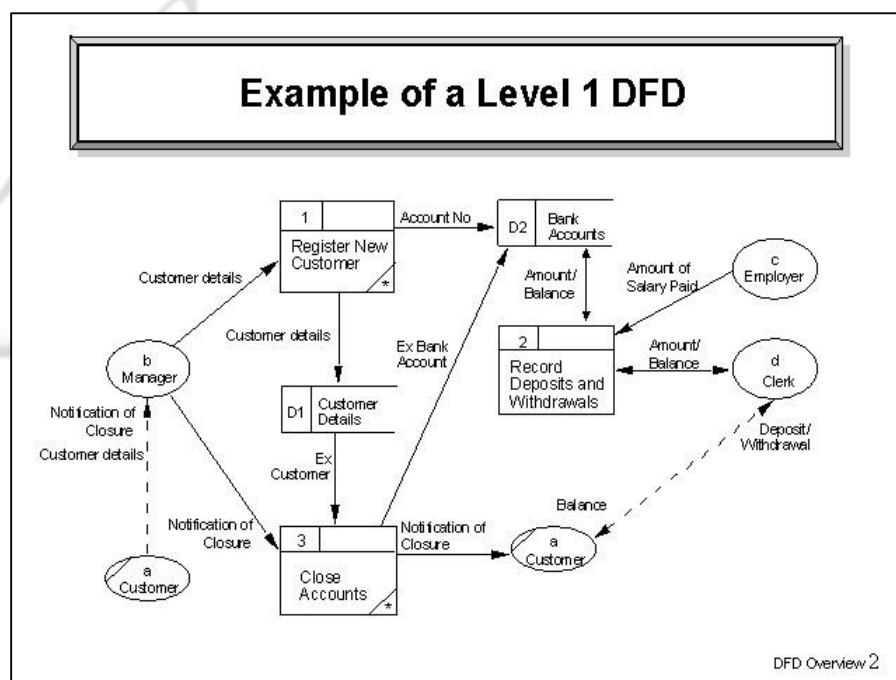


Figure 13: Banking System

**(Question: Draw the Level 1 DFD for Student Record system - 4 marks)**

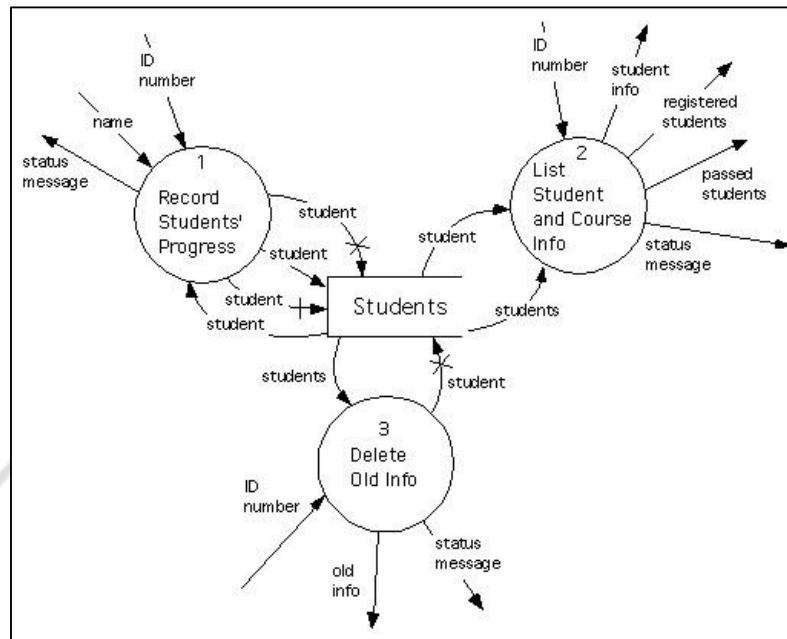


Figure 14: Student Record system

**(Question: Draw the Level 1 DFD for Fast Food system - 4 marks)**

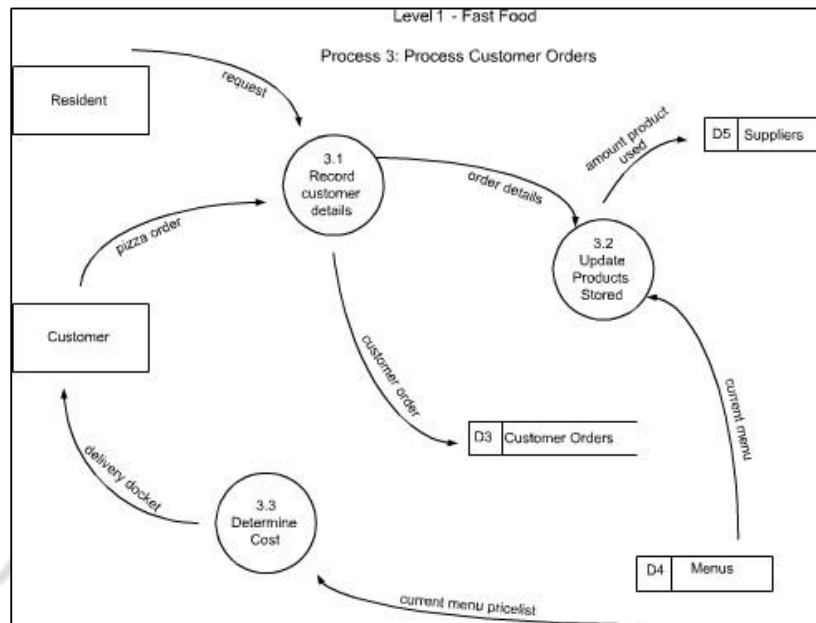


Figure 15: Fast Food System

**Level 2 DFD**

(Question: Draw the Level 2 DFD for Tender system - 4 marks)

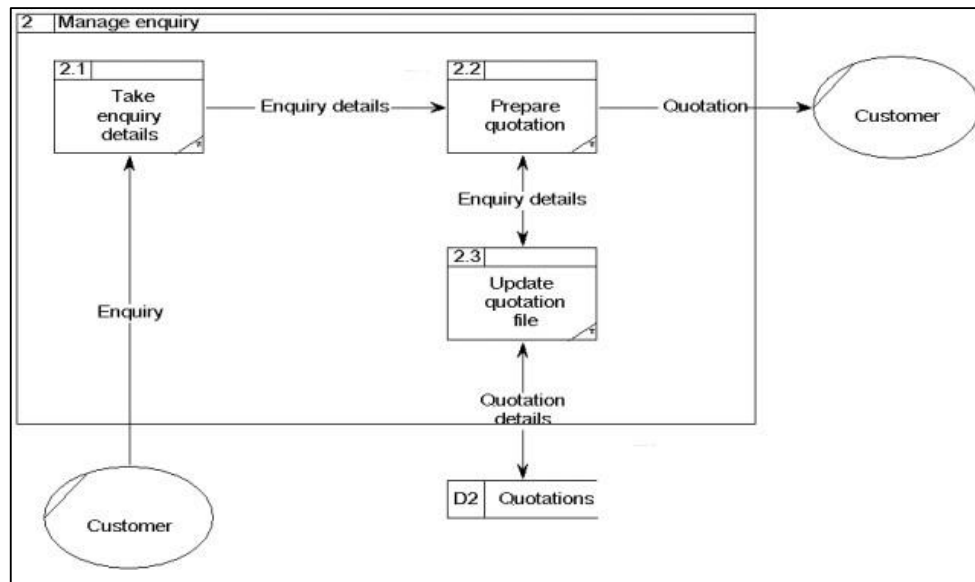


Figure 16: Tender System

**Logical DFD**

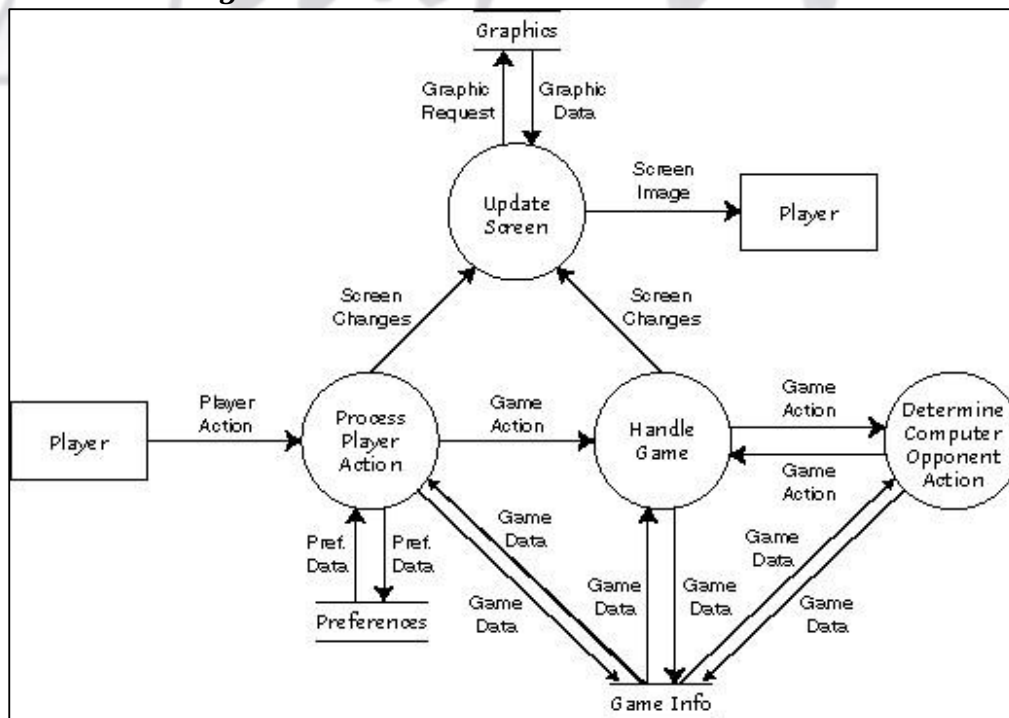


Figure 17: Logical DFD for computer game development

***The difference between a logical and a physical data flow diagram, typically referred to as a DFD, lies primarily in how the data is identified and represented.***

- i. A data flow diagram in general represents the movement of data within an organization, concentrating on its information system.
- ii. **A logical DFD** focuses more on the organization itself and identifies the datagenerating events that take place.
- iii. **A physical DFD** instead is concerned with how that data is represented.
- iv. Both types of DFDs are valuable tools for allowing users to monitor the flow of information from its entry point to its movement throughout the organization, and eventually to its exit point. Interpretation of the data along the way relies partially on recognizing whether the information is processed sequentially or in a parallel fashion.
- v. The benefits of a logical DFD include easy communication between employees, the potential for more stable systems, better understanding of the data and the system by analysts, and an overall flexibility.
- vi. It is also easy to maintain and to remove redundancies as they accumulate.
- vii. A physical DFD, on the other hand, has a clear distinction between manual and automated processes, provides more controls over the system, and identifies temporary data stores.

## 2. Data Dictionary

***(Question: Explain data dictionary with diagram - 4 marks)***

- i. Data dictionary is a set of meta-data which contains the definition and representation of data elements.
- ii. It gives a single point of reference of data repository of an organization. Data dictionary lists all data elements but does not say anything about relationships between data elements.
- iii. A data dictionary or database dictionary is a file that defines the basic organization of a database.
- iv. A database dictionary contains a list of all files in the database, the number of records in each file, and the names and types of each data field.



- v. Most database management systems keep the data dictionary hidden from users to prevent them from accidentally destroying its contents.
- vi. Data dictionaries do not contain any actual data from the database, only bookkeeping information for managing it.
- vii. Without a data dictionary, however, a database management system cannot access data from the database.

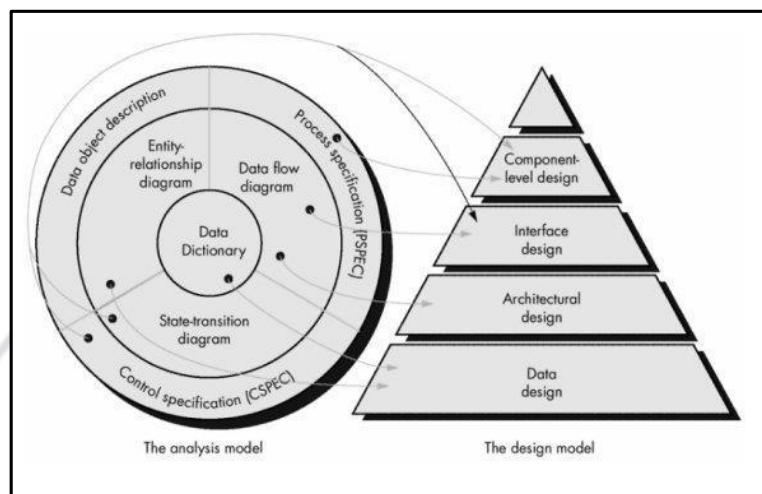


Figure 18: Data Dictionary

### 3. Creating Control Flow Model

- i. Illustrates how events affect the behavior of a system through the use of state diagrams.
- ii. Data flow and the control flow diagrams are necessary to obtain the meaningful insight of the software requirements.
- iii. There are large class of events that are driven by events rather than data.
- iv. Such applications that are driven by events require control flow model.

### 4. Creating Process Specifications

- i. Six class selection characteristics that retain information.
- ii. Information must be remembered about the system over time.
- iii. Needed services
- iv. Set of operations that can change the attributes of a class.

- v. Multiple attributes: Whereas, a single attribute may denote an atomic variable rather than a class.
- vi. Common attributes: A set of attributes apply to all instances of a class vii. Common operations: A set of operations apply to all instances of a class viii. Essential requirements: Entities that produce or consume information.

### 3. Scenario Based modelling 1. Use case

- i. “[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases).” Ivar Jacobson ii. The concept is relatively easy to understand- describe a specific usage scenario in straightforward language from the point of view of a defined actor.

### 2. Writing Use-Cases

- i. What should we write about?
- ii. Inception and elicitation provide us the information we need to begin writing use cases.
- iii. How much should we write about it?
- iv. How detailed should we make our description?
- v. How should we organize the description?

### 3. Developing an Activity Diagram

- i. What are the main tasks or functions that are performed by the actor? ii. What system information will the actor acquire, produce or change?
- iii. Will the actor have to inform the system about changes in the external environment?
- iv. What information does the actor desire from the system?
- v. Does the actor wish to be informed about unexpected changes?

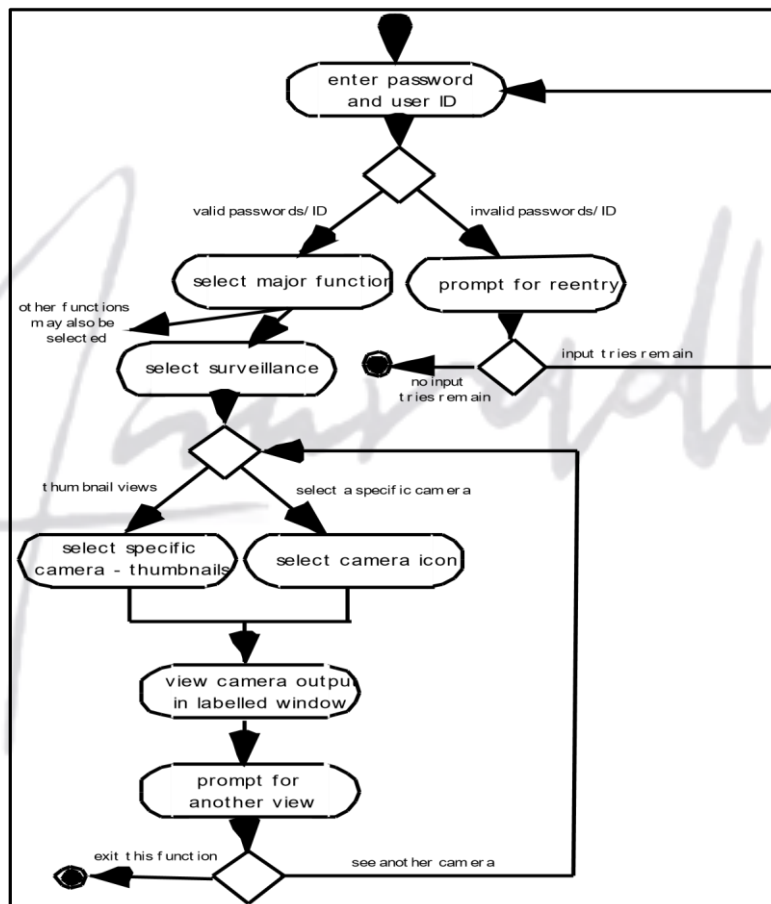


Figure 19: Activity Diagram

### 4. Swim lane Diagrams

- i. The UML swim lane diagram is a useful variation of the activity diagram and allows the modeler to represent the flow of activities described by the user-

case and at the same time indicate which actor or analysis class has responsibility for the action described by an activity rectangle.

- ii. Responsibilities are represented as parallel segments that divide the diagram vertically, like the lanes in a swimming pool.

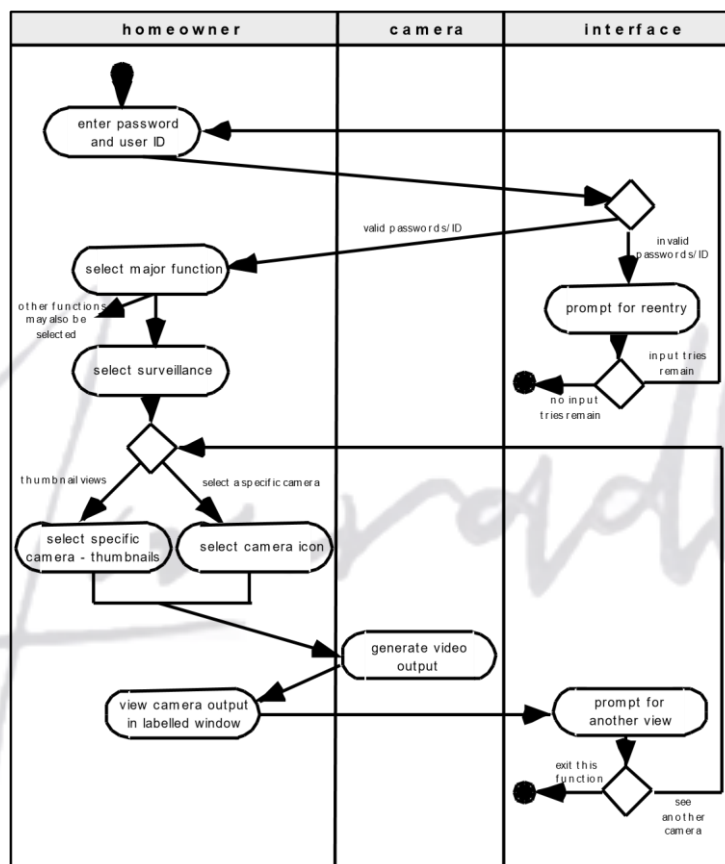


Figure 20: Swim lane Diagram

# Software Testing

A large, faint, stylized signature in a cursive script, appearing to read 'Anuradha', is centered on the page. The signature is light gray and has a soft, ethereal quality.

# Strategies and Methods

## CONTENTS

- I. Software Testing Fundamentals
  - 1. Definition of Software Testing
- II. Characteristics of Testing Strategies
- III. Software Verification and Validation (V&V)
- IV. Testing Strategies
  - 1. Unit Testing
  - 2. Integration Testing
- V. Alpha and Beta Testing ( Concept and differences)
- VI. System Testing
  - 1. Concept of System Testing
- VII. Concept of White-box and Black-Box Testing
- VIII. Debugging
  - 1. Concept and need of Debugging
  - 2. Characteristics of bugs
- IX. Debugging Strategies
  - 1. Concept of Brute Force, Back Tracking, Induction, Deduction

## **I. Software Testing Fundamentals**

### **1. Definition of Software Testing**

- i. Software testing is a method of assessing the functionality of a software program.
- ii. There are many different types of software testing but the two main categories are dynamic testing and static testing.

## II. Characteristics of Testing Strategies

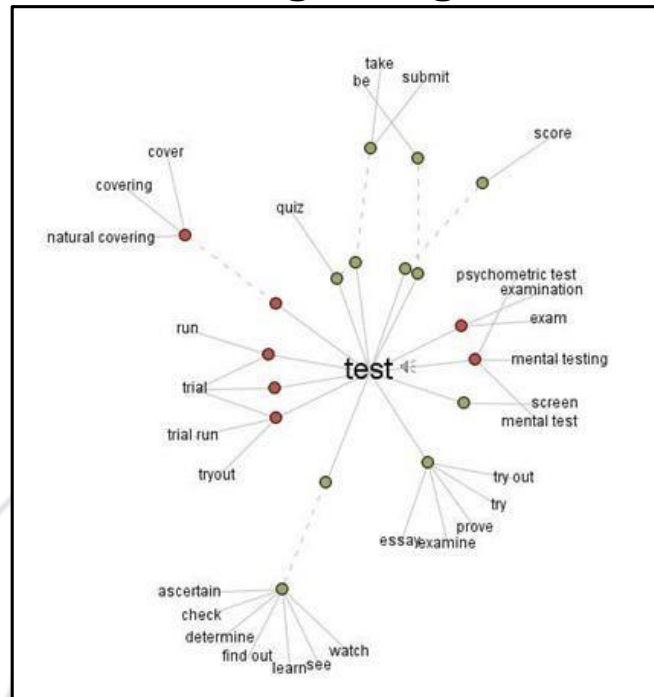


Figure 1: Software Testing

Software Testing Myths and Facts: Just as every field has its myths, so does the field of Software Testing. Software testing myths have arisen primarily due to the following:

- i. Lack of authoritative facts.
- ii. Evolving nature of the industry.
- iii. General flaws in human logic.

Some of the myths are explained below, along with their related facts:

1. **MYTH:** Quality Control = Testing.
  - **FACT:** Testing is just one component of software quality control. Quality Control includes other activities such as Reviews.
2. **MYTH:** The objective of Testing is to ensure a 100% defect- free product.
  - **FACT:** The objective of testing is to uncover as many defects as possible. Identifying all defects and getting rid of them is impossible.
3. **MYTH:** Testing is easy.
  - **FACT:** Testing can be difficult and challenging (sometimes, even more so than coding).

## Software Engineering

---

4. **MYTH:** Anyone can test.
  - **FACT:** Testing is a rigorous discipline and requires many kinds of skills.
5. **MYTH:** There is no creativity in testing.
  - **FACT:** Creativity can be applied when formulating test approaches, when designing tests, and even when executing tests.
6. **MYTH:** Automated testing eliminates the need for manual testing.
  - **FACT:** 100% test automation cannot be achieved. Manual Testing, to some level, is always necessary.
7. **MYTH:** When a defect slips, it is the fault of the Testers.
  - **FACT:** Quality is the responsibility of all members/stakeholders, including developers, of a project.
8. **MYTH:** Software Testing does not offer opportunities for career growth.
  - **FACT:** Gone are the days when users had to accept whatever product was dished to them; no matter what the quality.
  - With the abundance of competing software and increasingly demanding users, the need for software testers to ensure high quality will continue to grow.

### III. Software Verification and Validation (V&V)

(Question: Differentiate between verification and validation- 6 Marks)

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
Objective	To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.	To ensure that the product actually meets the user's needs, and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use



## Software Engineering

---

		when placed in its intended environment.
Question	Are we building the product <i>right</i> ?	Are we building the <i>right</i> product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	<ul style="list-style-type: none"><li>▪ Reviews</li><li>▪ Walkthroughs</li><li>▪ Inspections</li></ul>	<ul style="list-style-type: none"><li>▪ Testing</li></ul>

## IV. Testing Strategies

**(Question: Explain the software testing strategies - 8 Marks)**

1. **(Question: Explain unit testing with Diagram- 4 marks)**

- i. **Unit Testing** is a level of the software testing process where individual units/components of a software/system are tested.
- ii. The purpose is to validate that each unit of the software performs as designed.

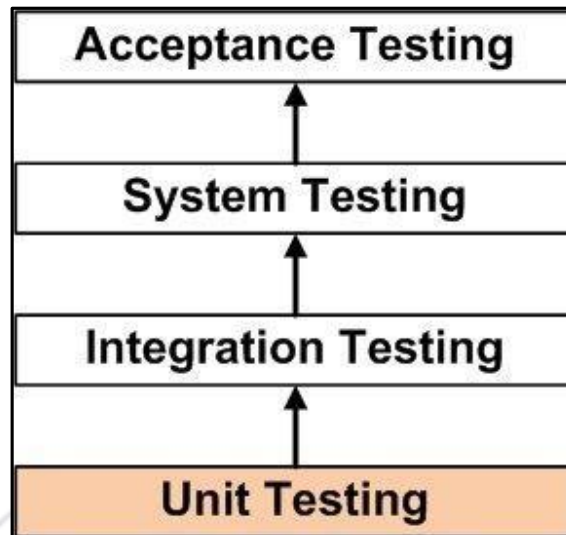


Figure 2: Unit Testing

- iii. A unit is the smallest testable part of software.
- iv. It usually has one or a few inputs and usually a single output.
- v. In procedural programming a unit may be an individual program, function, procedure, etc.
- vi. In object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

### Advantages

- i. Unit testing increases confidence in changing/maintaining code.
- ii. If good unit tests are written and if they are run every time any code is changed, the likelihood of any defects due to the change being promptly caught is very high.
- iii. If unit testing is not in place, the most one can do is hope for the best and wait till the test results at higher levels of testing are out.
- iv. If codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
- v. Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
- vi. The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.
- vii. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or say when the software is live.

- viii. Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be debugged.

2. *(Question: Explain integration testing with diagram and give its advantages and disadvantages- 6 Marks)*

- i. **Integration Testing** is a level of the software testing process where **Integration Testing**

individual units are combined and tested as a group.

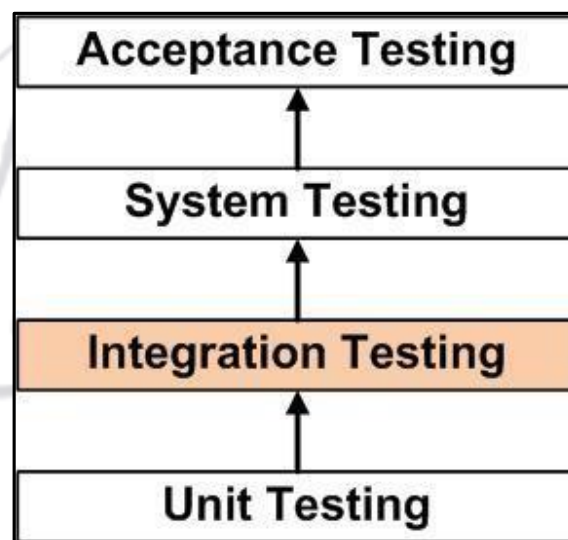


Figure 3: Integration Testing

- ii. The purpose of this level of testing is to expose faults in the interaction between integrated units.
- iii. Test drivers and test stubs are used to assist in Integration Testing.

## V. Alpha and Beta Testing

### 1. Alpha Testing

*(Question: Explain alpha testing - 4 Marks)*

- i. Alpha testing is one of the most common software testing strategy used in software development. It's specially used by product development organizations. ii. This **test takes place at the developer's site**. Developers observe the users and note problems.

## Software Engineering

---

- iii. Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing.
- iv. Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers.
- v. Alpha testing is final testing before the software is released to the general public. It has two phases:
  - a. In the **first phase** of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly.
  - b. In the **second phase** of alpha testing, the software is handed over to the software QA staff, for additional testing in an environment that is similar to the intended use.
- vi. Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site.
- vii. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

### 2. Beta testing

**(Question: Explain beta testing. – 4 Marks)**

- i. It is also known as field testing.
- ii. It takes place at **customer's site**.
- iii. It sends the system to users who install it and use it under real-world working conditions.
- iv. A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out.
- v. Beta testing can be considered "pre-release testing."
- vi. The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user's perspective that you would not want to have in your final, released version of the application.
- vii. Closed beta versions are released to a select group of individuals for a user test and are invitation only, while
- viii. Open betas are from a larger group to the general public and anyone interested.

- ix. The testers report any bugs that they find, and sometimes suggest additional features they think should be available in the final version.

## VI. System Testing

### 1. Concept of System Testing

- i. In system testing the behavior of whole system/product is tested as defined by the scope of the development project or product.
- ii. It may include tests based on risks and/or requirement specifications, business process, use cases, or other high level descriptions of system behavior, interactions with the operating systems, and system resources.
- iii. System testing is most often the final test to verify that the system to be delivered meets the specification and its purpose.
- iv. System testing is carried out by specialist's testers or independent testers.
- v. System testing should investigate both functional and non-functional requirements of the testing.

## VII. Concept of White-box and Black-Box Testing

*(Question: Differentiate between black box testing and white box testing - 6 Marks)*

### 1. Black Box Testing

- i. **Black Box Testing**, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester.
- ii. These tests can be functional or non-functional, though usually functional.

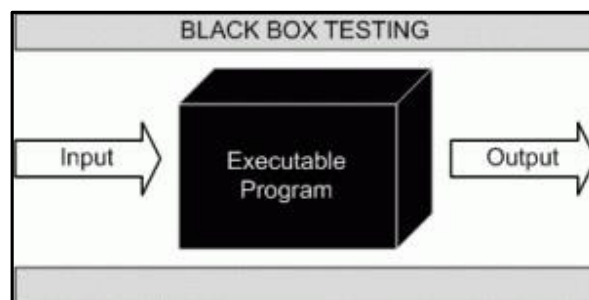


Figure 4: Black Box Testing

- iii. This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.
- iv. Black Box Testing is contrasted with White Box Testing. View Differences between Black Box Testing and White Box Testing.
- v. This method of attempts to find errors in the following categories:
  - Incorrect or missing functions
  - Interface errors
  - Errors in data structures or external database access
  - Behavior or performance errors
  - Initialization and termination errors

### **Black Box Testing Techniques**

**(Question: Explain the various black box testing techniques- 6 Marks)**

- i. **Equivalence partitioning**
  - a) Equivalence Partitioning is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- ii. **Boundary Value Analysis**
  - a) Boundary Value Analysis is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/outside of the boundaries as test data.
- iii. **Cause Effect Graphing**
  - a) Cause Effect Graphing is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

### **BLACK BOX TESTING ADVANTAGES**

- i. Tests are done from a user's point of view and will help in exposing discrepancies in the specifications
- ii. Tester need not know programming languages or how the software has been implemented
- iii. Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias
- iv. Test cases can be designed as soon as the specifications are complete

### BLACK BOX TESTING DISADVANTAGES

- i. Only a small number of possible inputs can be tested and many program paths will be left untested
- ii. Without clear specifications, which is the situation in many projects, test cases will be difficult to design
- iii. Tests can be redundant if the software designer/ developer has already run a test case.
- iv. Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

### 2. White Box Testing

- i. **White Box Testing** (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester.
- ii. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs.
- iii. Programming know-how and the implementation knowledge is essential.
- iv. White box testing is testing beyond the user interface and into the nittygritty of a system.

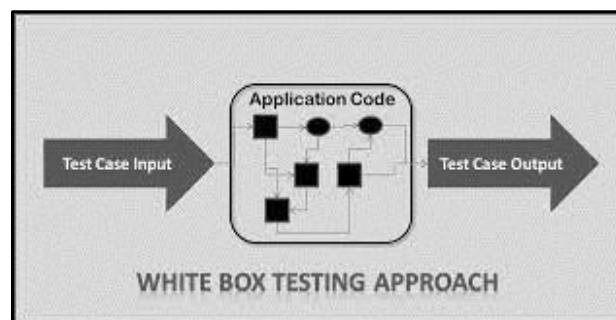


Figure 5: White Box Testing

### White Box Testing Advantages

- i. Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- ii. Testing is more thorough, with the possibility of covering most paths.

## Software Engineering

---

### White Box Testing Disadvantages

- i. Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
- ii. Test script maintenance can be a burden if the implementation changes too frequently.
- iii. Since this method of testing is closely tied with the application being tested, tools to cater to every kind of implementation/platform may not be readily available.
- iv. White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

Criteria	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing: Acceptance Testing System Testing	Mainly applicable to lower levels of testing: Unit Testing Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required



Basis for Test  
Cases

Requirement Specifications

Detail Design

## VIII. Debugging

### 1. Concept and need of Debugging

**(Question: Describe the concept and need of debugging in software testing- 6 Marks)**

- i. Debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code or the engineering of a hardware device.
- ii. To *debug* a program or hardware device is to start with a problem, isolate the source of the problem, and then fix it.
- iii. A user of a program that does not know how to fix the problem may learn enough about the problem to be able to avoid it until it is permanently fixed.
- iv. When someone says they've debugged a program or "worked the bugs out" of a program, they imply that they fixed it so that the bugs no longer exist.
- v. Debugging is a necessary process in almost any new software or hardware development process, whether a commercial product or an enterprise or personal application program.
- vi. For complex products, debugging is done as the result of the unit test for the smallest unit of a system, again at component test when parts are brought together, again at system test when the product is used with other existing products, and again during customer beta test, when users try the product out in a real world situation.
- vii. Because most computer programs and many programmed hardware devices contain thousands of lines of code, almost any new product is likely to contain a few bugs.
- viii. Invariably, the bugs in the functions that get most use are found and fixed first. An early version of a program that has lots of bugs is referred to as "buggy."
- ix. Debugging tools (called debuggers) help identify coding errors at various development stages. Some programming language packages include a facility for checking the code for errors as it is being written.

### 2. Characteristics of bugs

***(Question: Explain the various characteristics of bugs – 4 Marks)***

- i. A defect is an error or a bug, in the application which is created.
- ii. A programmer while designing and building the software can make mistakes or error.
- iii. These mistakes or errors mean that there are flaws in the software. These are called defects.
- iv. When actual result deviates from the expected result while testing a software application or product then it results into a defect.
- v. Hence, any deviation from the specification mentioned in the product functional specification document is a defect. In different organizations it's called differently like bug, issue, incidents or problem.
- vi. When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a Bug or Defect.
- vii. These defects or bugs occur because of an error in logic or in coding which results into the failure or unpredicted or unanticipated results.

## IX. Debugging Strategies

***(Question: Explain any two debugging strategies- 8 marks)***

### 1. Brute Force

- i. This method is most common and least efficient for isolating the cause of a software error.
- ii. We apply this method when all else fail. In this method, a printout of all registers and relevant memory locations is obtained and studied.
- iii. All dumps should be well documented and retained for possible use on subsequent problems.

### 2. Back Tracking

- i. It is a quite popular approach of debugging which is used effectively in case of small applications.
- ii. The process starts from the site where a particular symptom gets detected, from there on backward tracing is done across the entire source code till we are able to lay our hands on the site being the cause.
- iii. As the number of source lines increases, the number of potential backward paths may become unmanageably large.

### 3. Cause Elimination(Induction and Deduction)

- i. The third approach to debugging, because elimination, is manifested by induction or deduction and introduces the concept of binary partitioning. This approach is also called induction and deduction.
- ii. Data related to the error occurrence are organized to isolate potential causes.
- iii. A "cause hypothesis" is devised and the data are used to prove or disprove the hypothesis. iv. A list of all possible causes is developed and tests are conducted to eliminate each.
- v. If initial tests indicate that a particular cause hypothesis shows promise, the data are refined in an attempt to isolate the bug.

# Software Project Management

## CONTENTS

- I. Introduction to Software Project Management and its need.
- II. The Management Spectrum – 4 Ps and their Significance
- III. Project Scheduling
  - 1. Concept of Project Scheduling
  - 2. Factors that delay Project Schedule
  - 3. Principles of Project Scheduling
  - 4. Project Scheduling Techniques- Concept of Gantt Chart, PERT, CPM
- IV. Concept of Task Network V. Ways of Project Tracking
- VI. Risk Management
  - 1. What is Software Risk?
  - 2. Concept of Proactive and Reactive risk strategies
  - 3. Types of Software Risks VII. Risk Assessment
    - 1. Risk Identification
    - 2. Risk Analysis
- VIII. Risk control- Need, RMMM strategy
- IX. Software Configuration Management (SCM)
  - 1. Need of SCM
  - 2. Benefits of SCM
  - 3. SCM Repository-Functions and Features supported
- X. SCM Process- Change control and version Control
- I. Introduction to Software Project Management and its need.**

***(Question: explain the need of software project management- 4 Marks)***

Software projects have several properties that make them very different to other kinds of engineering project.

1. **The product is intangible:** It's hard to claim a bridge is 90% complete if there is not 90% of the bridge there. It is easy to claim that a software project is 90% complete, even if there are no visible outcomes.
2. **We don't have much experience.** Software engineering is a new discipline, and so we simply don't have much understanding of how to engineer large scale software projects.
3. **Large software projects are often "bespoke".** Most large software systems are one-off, with experience gained in one project being of little help in another.
4. **The technology changes very quickly.** Most large software projects employ new technology; for many projects.

## II. The Management Spectrum – 4 Ps and their Significance

***(Question: Explain the 4P's of management spectrum- 4 Marks)***

Effective software project management focuses on these items (in this order)

### 1. The people

III. Deals with the cultivation of motivated, highly skilled people

- i. Consists of the stakeholders, the team leaders, and the software team

### 2. The product

- i. Product objectives and scope should be established before a project can be planned

### 3. The process

- i. The software process provides the framework from which a comprehensive plan for software development can be established

### 4. The project

- i. Planning and controlling a software project is done for one primary reason...it is the only known way to manage complexity
- ii. In a 1998 survey, 26% of software projects failed outright, 46% experienced cost and schedule overruns

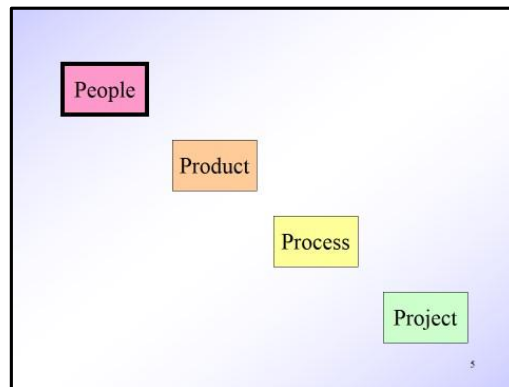


Figure 1: 4 P's of Project Management

## IV. Project Scheduling

### 1. Concept of Project Scheduling

- i. Changing customer requirements that are not reflected in schedule changes.
- ii. An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.
- iii. Predictable and/or unpredictable risks that were not considered when the project commenced.
- iv. Technical difficulties that could not have been foreseen in advance.

### 2. Factors that delay Project Schedule

Although there are many reasons why software is delivered late, most can be traced to one or more of the following root causes:

- i. An unrealistic deadline established by someone outside the software development group and forced on managers and practitioners within the group.
- ii. Changing customer requirements that are not reflected in schedule changes.
- iii. An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.
- iv. Predictable and/or unpredictable risks that were not considered when the project commenced.
- v. Technical difficulties that could not have been foreseen in advance.
- vi. Human difficulties that could not have been foreseen in advance.
- vii. Miscommunication among project staff that results in delays.
- viii. A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

### 3. Principles of Project Scheduling

- i. Compartmentalization: The project must be compartmentalized into a number of manageable activities and tasks.
- ii. Interdependency: The interdependency of each compartmentalized activity or task must be determined.
- iii. Time allocation: Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort).
- iv. Effort validation: the project manager must ensure that no more than the allocated number of people have been scheduled at any given time.
- v. Defined responsibilities: Every task that is scheduled should be assigned to a specific team member
- vi. Defined outcomes: Every task that is scheduled should have a defined outcome.
- vii. Defined milestones: Every task or group of tasks should be associated with a project milestone.
- viii. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

### 4. Project Scheduling Techniques- Concept of Gantt Chart, PERT, CPM 1. Gantt Chart

- i. A project control technique
- ii. Defined by Henry L. Gantt
- iii. Used for several purposes, including scheduling, budgeting, and resource planning.
- iv. When creating a software project schedule, the planner begins with a set of tasks.
- v. If automated tools are used, the work breakdown is input as a task network or task outline.
- vi. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.
- vii. As a consequence of this input, a timeline chart is generated also called Gantt chart.
- viii. A timeline chart can be developed for the entire project.
- ix. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

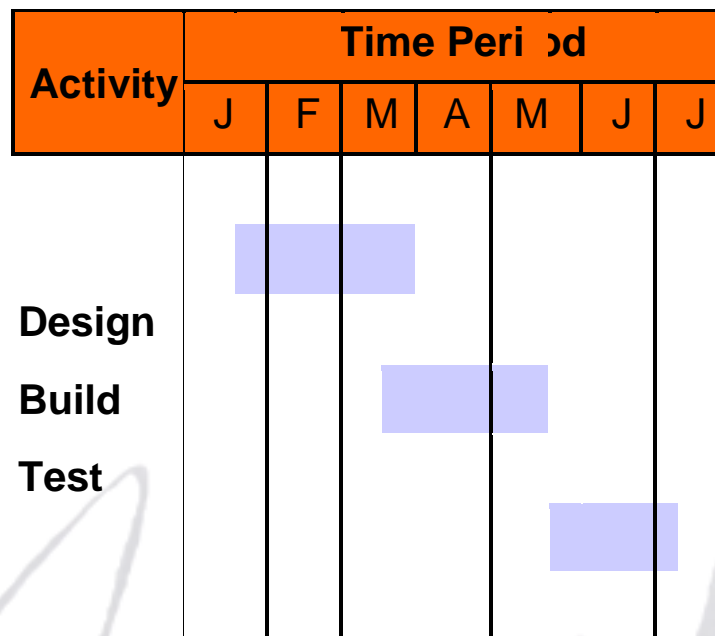


Figure 2: Gantt chart

### 2. Differentiate between Pert and CPM

- Network techniques
- Developed in 1950's
- CPM by DuPont for chemical plants
- PERT by U.S. Navy for Polaris missile
- Consider precedence relationships & interdependencies
- Each uses a different estimate of activity times

#### Benefits of PERT/CPM

- Useful at many stages of project management
- Mathematically simple
- Use graphical displays
- Give critical path & slack time
- Provide project documentation
- Useful in monitoring costs

#### Disadvantage of PERT and CPM

- Clearly defined, independent, & stable activities
- Specified precedence relationships
- Activity times (PERT) follow beta distribution



- iv. Subjective time estimates
- v. Over emphasis on critical path

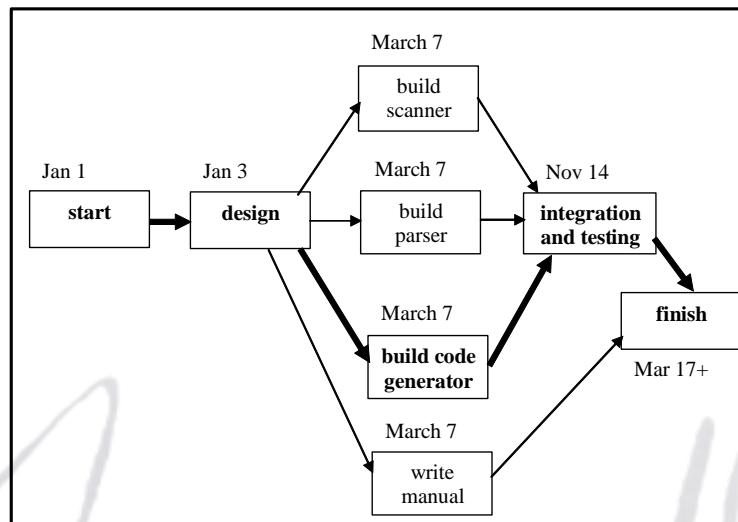


Figure 3: PERT

## V. Concept of Task Network

1. Individual tasks and subtasks have interdependencies based on their sequence.
2. A task network is a graphic representation of the task flow for a project.
3. A schematic network for a concept development project.
4. Critical path: The tasks on a critical path must be completed on schedule to make the whole project on schedule.

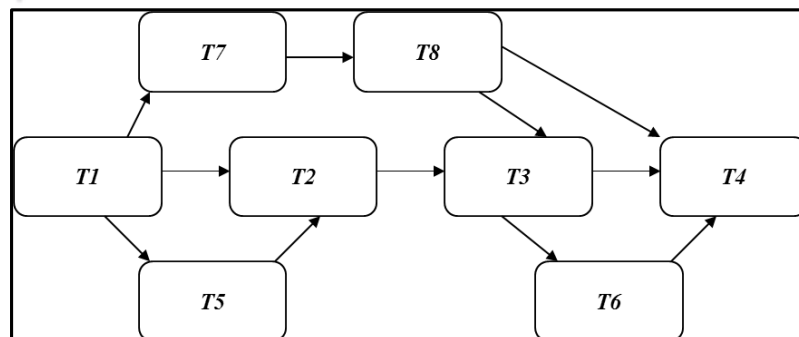


Figure 4: Task Network

## VI. Ways of Project Tracking

---

## Software Engineering

---

1. Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort.
2. Two project scheduling methods:
  - i. Program Evaluation and Review Technique (PERT)
  - ii. Critical Path Method (CPM)
3. Both methods are driven by information developed in earlier project planning activities:
  - i. Estimates of effort
  - ii. A decomposition of product function
  - iii. The selection of the appropriate process model
  - iv. The selection of project type and task set
4. Both methods allow a planner to do:
  - i. Determine the critical path
  - ii. Time estimation
  - iii. Calculate boundary times for each task
5. Boundary times:
  - i. The earliest time and latest time to begin a task
  - ii. The earliest time and latest time to complete a task
  - iii. The total float.

## VII. Risk Management

### 1. What is Software Risk?

*(Question: Explain the concept of risks in software- 4 marks)*

- i. Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.
- ii. CTC (condition-transition-consequence) format may be a good representation for the detailed risks (e.g. given that <condition> then there is a concern that (possibly) <consequence>).
- iii. This general condition can be refined in the following manner:
  - a. Sub condition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.
  - b. Sub condition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
  - c. Sub condition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.
  - iv. The

consequences associated with these refined sub conditions remains the same (i.e., 30 percent of software components must be customer engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

### **2. Concept of Proactive and Reactive risk strategies**

*(Question: What are reactive and proactive risk strategies? 4 Marks, 2 Marks each)*

#### **1. Reactive risk strategies**

- i. Reactive risk strategies follows that the risks have to be tackled at the time of their occurrence.
- ii. No precautions are to be taken as per this strategy.
- iii. They are meant for risks with relatively smaller impact.

#### **2. Proactive risk strategies**

- i. Proactive risk strategies follows that the risks have to be identified before start of the project.
- ii. They have to be analyzed by assessing their probability of occurrence, their impact after occurrence, and steps to be followed for its precaution.
- iii. They are meant for risks with relatively higher impact.

### **3. Types of Software Risks 1. Software Requirement Risk**

- i. Lack of analysis for change of requirements.
- ii. Change extension of requirements.
- iii. Lack of report for requirements.
- iv. Poor definition of requirements.
- v. Ambiguity of requirements.
- vi. Change of requirements.
- vii. Inadequate of requirements.
- viii. Impossible requirements.
- ix. Invalid requirements.

### 2. Software Cost Risks

- i. Lack of good estimation in projects
- ii. Unrealistic schedule
- iii. The hardware does not work well
- iv. Human errors
- v. Lack of testing
- vi. Lack of monitoring
- vii. Complexity of architecture
- viii. Large size of architecture
- ix. Extension of requirements change
- x. The tools does not work well
- xi. Personnel change, Management change, technology change, and environment change
- xii. Lack of reassessment of management cycle

### 3. Software Scheduling Risks

- i. Inadequate budget
- ii. Change of requirements and extension of requirements
- iii. Human errors
- iv. Inadequate knowledge about tools and techniques
- v. Long-term training for personnel
- vi. Lack of employment of manager experience
- vii. Lack of enough skill
- viii. Lack of good estimation in projects

### 4. Software Quality Risks

- i. Inadequate documentation
- ii. Lack of project standard
- iii. Lack of design documentation
- iv. Inadequate budget
- v. Human errors
- vi. Unrealistic schedule
- vii. Extension of requirements change
- viii. Poor definition of requirements
- ix. Lack of enough skill

- x. Lack of testing and good estimation in projects

### **VIII. Risk Assessment**

***(Question: What is the concept of risk Assessment? - 4 Marks)***

1. Risk assessment is another important case that integrates risk management and risk analysis.
2. There are many risk assessment methodologies that focus on different types of risks. Risk assessment requires correct explanations of the target system and all security features.
3. It is important that a risk referent levels like performance, cost, support and schedule must be defined properly for risk assessment to be useful.

### **IX. Risk Analysis**

***(Question: What is the concept of risk analysis? - 4 Marks)***

1. There are quite different types of risk analysis that can be used.
2. Risk analysis is used to identify the high risk elements of a project in software engineering.
3. It provides ways of detailing the impact of risk mitigation strategies.
4. Risk analysis has also been found to be most important in the software design phase to evaluate criticality of the system, where risks are analyzed and necessary counter measures are introduced.
5. The main purpose of risk analysis is to understand risks in better ways and to verify and correct attributes.
6. A successful risk analysis includes important elements like problem definition, problem formulation, data collection.

### **X. Risk control- Need, RMMM strategy**

***(Question: Explain RMMM and its need in software project management- 8 Marks)***

An effective strategy for dealing with risk must consider three issues

1. Risk mitigation (i.e., avoidance)
2. Risk monitoring

## Software Engineering

---

### 3. Risk management and contingency planning

#### **Need for RMMM**

- i. Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market)
- ii. Mitigate those causes that are under our control before the project starts
- iii. Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave
- iv. Organize project teams so that information about each development activity is widely dispersed
- v. Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner
- vi. Conduct peer reviews of all work (so that more than one person is "up to speed")
- vii. Assign a backup staff member for every critical technologist.
- viii. During risk monitoring, the project manager monitors factors that may provide an indication of whether a risk is becoming more or less likely
- ix. Risk management and contingency planning assume that mitigation efforts have failed and that the risk has become a reality
- x. RMMM steps incur additional project cost
- xi. Large projects may have identified 30 – 40 risks
- xii. Risk is not limited to the software project itself
- xiii. Risks can occur after the software has been delivered to the user

## **XI. Software Configuration Management (SCM)**

### **1. Need of SCM**

***(Question: Give the need of SCM in software engineering- 4 Marks)***

- i. Also called software configuration management (SCM)
- ii. It is an umbrella activity that is applied throughout the software process
- iii. It's goal is to maximize productivity by minimizing mistakes caused by confusion when coordinating software development
- iv. SCM identifies, organizes, and controls modifications to the software being built by a software development team

- v. SCM activities are formulated to identify change, control change, ensure that change is being properly implemented, and report changes to others who may have an interest

### **2. Benefits of SCM**

***(Question: Explain the benefits of SCM- 4 marks)***

- i. Identify all items that collectively define the software configuration
- ii. Manage changes to one or more of these items
- iii. Facilitate construction of different versions of an application
- iv. Ensure the software quality is maintained as the configuration evolves over time
- v. Provide information on changes that have occurred

### **3. SCM Repository-Functions and Features supported**

***(Question: Explain SCM Repository with Diagram- 4 Marks, 2 marks each)***

Automated SCM Repository

- i. It is a set of mechanisms and data structures that allow a software team to manage change in an effective manner
- ii. It acts as the center for both accumulation and storage of software engineering information
- iii. Software engineers use tools integrated with the repository to interact with it

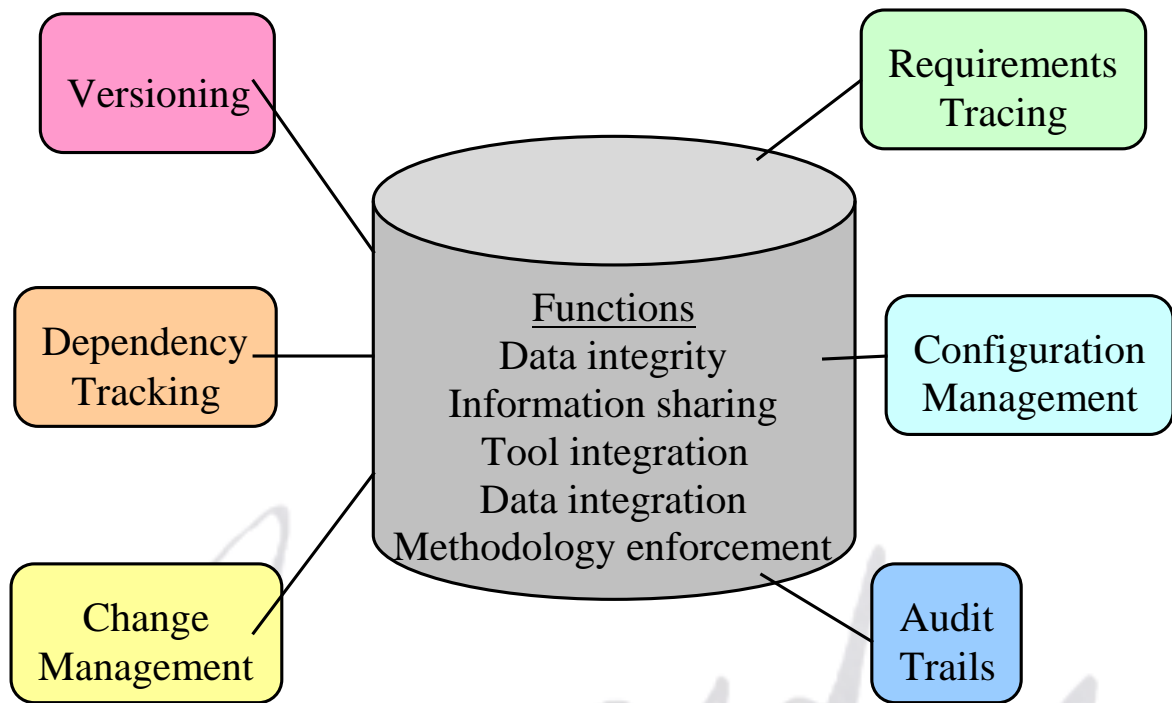


Figure 5: SCM Repository

### Functions of SCM repository

*(Question: List and explain the functions of SCM Repository- 6 Marks)*

1. **Data integrity** :Validates entries, ensures consistency, cascades modifications
2. **Information sharing** :Shares information among developers and tools, manages and controls multi-user access
3. **Tool integration** : Establishes a data model that can be accessed by many software engineering tools, controls access to the data
4. **Data integration** :Allows various SCM tasks to be performed on one or more CSCIs
5. **Methodology enforcement** :Defines an entity-relationship model for the repository that implies a specific process model for software engineering
6. **Document standardization** :Defines objects in the repository to guarantee a standard approach for creation of software engineering documents



Software Engineering

---

Anuradha

---

Anuradha Bhatia

## **XII. SCM Process- Change control and version Control**

*(Question: What is version control and change control in SCM Process.- 8 Marks)*

1. **Requirements Management.** Every requirement for the system should be tracked from end to end. This prevents development time being spent on non-essential features; it also allows QA to do a better job of testing. Should changes be introduced mid-development, their impact can be measured. Additionally, capturing requirements facilitates communicating them in a coordinated manner.
2. **Design.** The architecture and implementation blueprints of the system can be captured, clearly communicated, and easily distributed among developers, managers, and other decision-makers.
3. **Version Control.** As the software is developed, changes in the source are tracked. This is useful to developers, especially for development and bug fixing; it is useful for build automation, QA reporting, providing information to help desks, obtaining past releases, and managing patches.
4. **Build Tools.** The build process can be made fairly automated, easily constructing different platforms and releases as desired. Some tools provide the ability to track the interface versions that make up a release.
5. **Defect Tracking.** Allows problem reports to be directed to the development team, with reporting to management of problem areas and identifying trends. Developers need to be able to communicate problems among themselves.
6. **Automated Testing.** Provides assurance for regression testing and identifies problems during stress testing. Automation can rigorously beat on software and run through permutations faster than a human.
7. **Release Management.** Captures the contents of a particular release and information to construct a consistent build environment. This includes software patches for post-releases. Certain releases may have their own histories or preinstallation requirements.

8. **Distribution Management.** Captures how a release is distributed, whether by floppies, CD-ROM, WWW,FTP, etc. Certain distributions and platforms need their own install instructions.
9. **Installation Tools.** Some packages assist or construct the install process, or provide options for what gets installed, including licensing.
10. **Configuration Management.** Captures the hardware and software configuration at the installation site. Some customers may have different platforms, licenses, or patch releases applied.
11. **Help Desk.** Customers need a point of contact where they can go to report a problem, check the status of a fix, ask for a new feature, or to learn about the product.
12. **Impact Analysis.** When a new requirement is levied on the system, the impact from development, to QA, to documentation, to distribution, and so on, should be measured. Sometimes it isn't cost effective to do something just because you can.
13. **Process Management.** The product's lifecycle can be defined, communicated, sometimes enforced,<sup>1</sup> tracked, measured, and improved.
14. **Document Tracking.** Documentation consistency, changes, distribution, and store-housing are all managed under this branch.

# Software Quality Management

## CONTENTS

### I. Basic Quality Concepts

### II. Software Quality Assurance (SQA)

1. Definition of SQA
2. SQA Activities

### III. Quality Evaluation Standards

1. Six sigma for software
2. ISO 9000 for software - concept and major considerations

IV. CMMI- CMMI Levels, Process Areas considered. V. CMMI Vs ISO.

### VI. McCall's Quality factors.

## **I. Basic Quality Concepts**

1. Quality means that a product satisfies the demands of its specifications
2. It also means achieving a high level of customer satisfaction with the product
3. In software systems this is difficult
  - i. Customer quality requirements (e.g. efficiency or reliability) often conflict with developer quality requirements (e.g. maintainability or reusability)
  - ii. Software specifications are often incomplete, inconsistent, or ambiguous

## **II. Software Quality Assurance (SQA)**

### **1. Definition of SQA**

***(Question: Explain SQA with its activities. - 4 Marks)***

- i. Conformance to software requirements is the foundation from which software quality is measured.
- ii. Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered.
- iii. Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

### **2. SQA Activities**

These activities are performed (or facilitated) by an independent SQA group that:

- i. Prepares an SQA plan for a project.

## Software Engineering

---

- ii. Participates in the development of the project's software process description.
- iii. Reviews software engineering activities to verify compliance with the defined software process.
- iv. Audits designated software work products to verify compliance with those defined as part of the software process.
- v. Ensures that deviations in software work and work products are documented and handled according to a documented procedure.
- vi. Records any noncompliance and reports to senior management.

### III. Quality Evaluation Standards

#### 1. Six sigma for software - Concept of DMAIC and DMDAV Approach

*(Question: Explain six sigma for project development. - 6 Marks)*

- i. Six Sigma through the correct application of statistical tools can reap a company enormous rewards that will have a positive effect for years
- ii. Six Sigma can be a dismal failure if not used correctly.
- iii. A disciplined quantitative approach for improvement of defined metrics iv. Can be applied to all business processes, manufacturing, finance and services



Figure 1: Six Sigma Roadmap

- v. The failure rate of any project is represented as the six sigma with failure rate as shown in Figure 2.

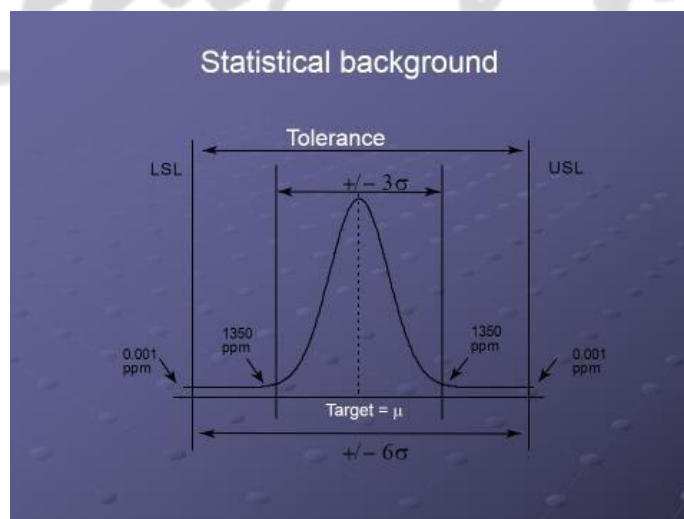


Figure 2: Six sigma Failure rate

## 2. ISO 9000 for software - concept and major considerations

*(Question: Explain the ISO 9000 standards for software development. – 4 Marks)*

## Software Engineering

---

- i. International set of standards for quality management
- ii. Quality standards and procedures must be documented in an organizational quality manual
- iii. An external body is often used to certify that the quality manual conforms to ISO 9000 standards
- iv. Many customers are demanding that suppliers are ISO 9000 certified

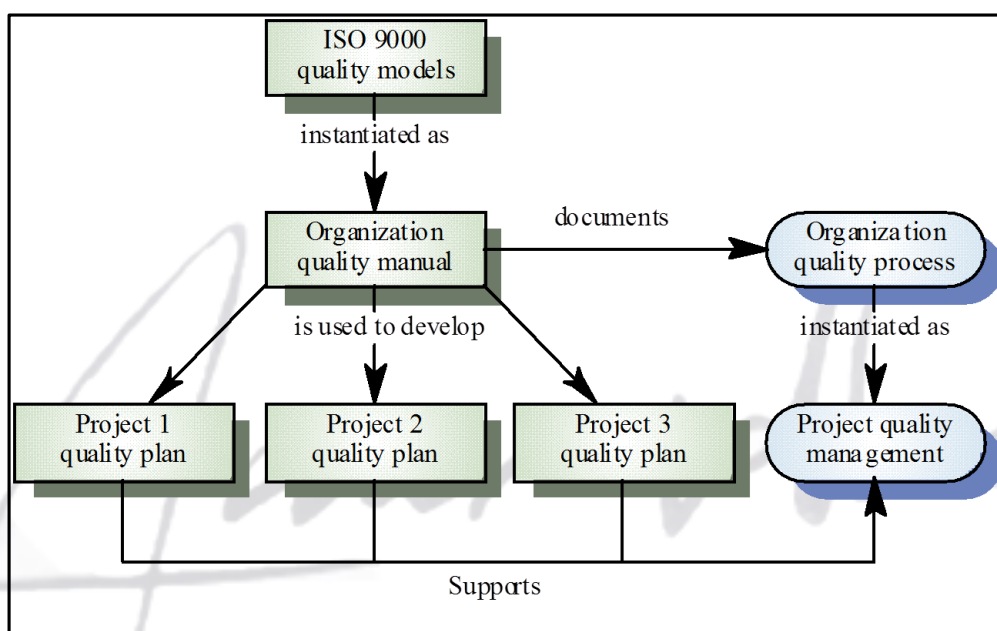


Figure 3: ISO model

## IV. CMMI- CMMI Levels, Process Areas considered.

*(Question: Explain CMMI models with diagram. – 6 Marks)*

1. CMMI (Capability Maturity Model Integration) is a proven industry framework to improve product quality and development efficiency for both hardware and software
  - i. Sponsored by US Department of Defence in cooperation with Carnegie Mellon University and the Software Engineering Institute (SEI)
  - ii. Many companies have been involved in CMMI definition such as Motorola and Ericsson
  - iii. CMMI has been established as a model to improve business results
2. CMMI, staged, uses 5 levels to describe the maturity of the organization, same as predecessor CMM

- i. Vastly improved version of the CMM
- ii. Emphasis on business needs, integration and institutionalization

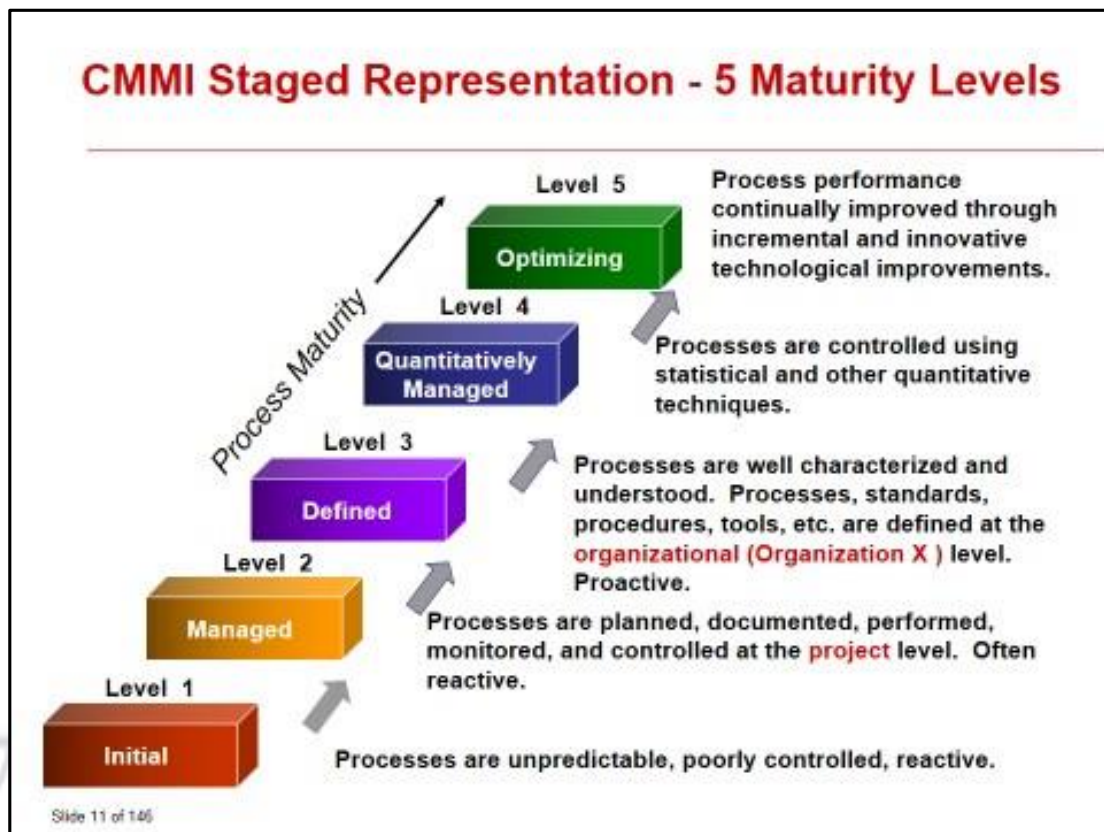


Figure 4: CMMI maturity model

## V. CMMI Vs ISO.

- The comparison of CMMI vs ISO reveals that while CMMI is more focused, complex, and aligned with business objectives, ISO is flexible, wider in scope and not directly linked to business objectives.
- The attainment of either a CMMI ranking or ISO certification nevertheless help organizations establish a quality management system and focus on continuous improvement.

### 1. CMMI vs ISO: Conceptual Difference

- i. The fundamental difference between CMMI vs ISO is conceptual. CMMI is a process model and ISO is an audit standard.



## Software Engineering

---

- ii. CMMI is a set of related "best practices" derived from industry leaders and relates to product engineering and software development.
- iii. Businesses receive CMMI ratings from Level 1 to Level 5 depending upon the extent of compliance to key performance areas specified in the selected CMMI process area.
- iv. ISO is a certification tool that certifies businesses whose processes conform to the laid down standards.

### **2. CMMI vs ISO: Scope**

- i. CMMI is rigid and extends only to businesses developing software intensive systems. ISO is flexible and applicable to all manufacturing industries.
- ii. CMMI focuses on engineering and project management processes whereas ISO's focus is generic in nature.
- iii. CMMI mandates generic and specific practices and businesses have a choice of selecting the model relevant to their business needs from 22 developed process areas.
- iv. ISO requirements are same for all companies, industries, and disciplines.

### **3. CMMI vs ISO: Approach**

- i. CMMI requires ingraining processes into business needs so that such processes become part of corporate culture and do not break down under the pressure of deadlines.
- ii. ISO specifies to conformance and remains oblivious as to whether such conformance is of strategic business value or not.
- iii. CMMI approaches risk management as an organized and technical discipline by identifying risk factors, quantifying such risk factors, and tracking them throughout the project life cycle.
- iv. ISO was until recently neutral on risk management. ISO 31000:2009 now provides generic guidelines for the design, implementation, and maintenance of risk management processes throughout an organization.
- v. Although CMMI focuses on linkage of processes to business goals, customer satisfaction is not a factor in the ranking whereas customer satisfaction is an important part of ISO requirements.

### **4. CMMI vs ISO Implementation**

- i. Neither CMMI nor ISO requires the establishment of new processes.

- ii. CMMI compares the existing processes to industry best practices whereas ISO requires adjustment of existing processes to confirm to the specific ISO requirements.
- iii. In practice, some organizations tend to rely on extensive documentation while implementing both CMMI and ISO.
- iv. Most organizations tend to constitute in-house teams, or rely on external auditors to see through the implementation process.

## VI. McCall's Quality factors

**(Question: Explain McCall's Quality factor with diagram. - 8 Marks)**

The factors that affect S/W quality can be categorized in two broad groups:

- 1. factors that can be directly measured (defects uncovered during testing)
- 2. factors that can be measured only indirectly (Usability and maintainability)

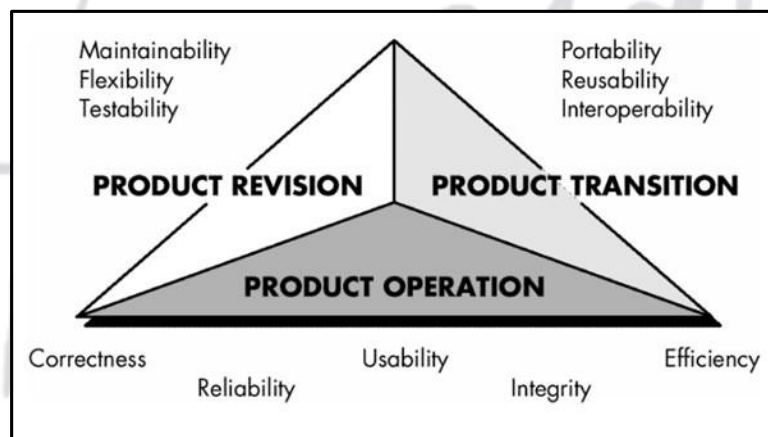


Figure 5: McCall's Quality Factor

The S/W quality factors shown above focus on three important aspects of a S/W product:

- i. Its operational characteristics
- ii. Its ability to undergo change
- iii. Its adaptability to new environments

**The various factors of quality are:**

## Software Engineering

---

- i. **Correctness:** The extent to which a program satisfies its specs and fulfills the customer's mission objectives.
- ii. **Reliability:** The extent to which a program can be expected to perform its intended function with required precision.
- iii. **Efficiency:** The amount of computing resources and code required to perform is function. iv. **Integrity:** The extent to which access to S/W or data by unauthorized persons can be controlled.
- v. **Usability:** The effort required to learn, operate, prepare input for, and interpret output of a program.
- vi. **Maintainability:** The effort required to locate and fix errors in a program.
- vii. **Flexibility:** The effort required to modify an operational program.
- viii. **Testability:** The effort required to test a program to ensure that it performs its intended function.
- ix. **Portability:** The effort required to transfer the program from one hardware and/or software system environment to another.
- x. **Reusability:** The extent to which a program can be reused in other applications-related to the packaging and scope f the functions that the program performs.
- xi. **Interoperability:** The effort required to couple one system to another.