1. What is software engineering?

Ans:

The term software engineering is composed of two words, software and engineering. Software is more than just a program code.

A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations.

Software, when made for a specific requirement is called software product. Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods. So, we can define software engineering as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures.

The outcome of software engineering is an efficient and reliable software product. IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software. We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines.

A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.

2. What is need of software engineering?

Ans:

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

- Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one

. - Cost- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

- Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

- Quality Management- Better process of software development provides better and quality software product.

3. What are characteristics of good software?

A software product can be judged by what it offers and how well it can be used.

This software must satisfy on the following grounds:

Operational

Transitional

Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

1. Operational This tells us how well software works in operations.

It can be measured on:

Budget

Usability

Efficiency

Correctness

Functionality

Dependability

Security

Safety

2. Transitional This aspect is important when the software is moved from one platform to another:

Portability

Interoperability

Reusability

Adaptability

3. Maintenance This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

Modularity

Maintainability

Flexibility

Scalability

4. List and explain different types of applications.

1. Stand-alone applications These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.

2. Interactive transaction-based applications Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.

3. Embedded control systems These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.

4. Batch processing systems These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

5. Entertainment systems These are systems that are primarily for personal use and which are intended to entertain the user.

6. Systems for modeling and simulation These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.

7. Data collection systems These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.

8. Systems of systems These are systems that are composed of a number of other software systems.

5. Explain SDLC.

LIFE CYCLE MODEL

- A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle.

- A life cycle model represents all the activities required to make a software product transit through its life cycle phases.

- It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement.

- Different life cycle models may map the basic development activities to phases in different ways.

- Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out.

6. What are coding guidelines?

Writing an efficient software code requires a thorough knowledge of programming. This knowledge can be implemented by following a coding style which comprises several guidelines that help in writing the software code efficiently and with minimum errors.

These guidelines, known as coding guidelines, are used to implement individual programming language constructs, comments, formatting, and so on. These guidelines, if followed, help in preventing errors, controlling the complexity of the program, and increasing the readability and understandability of the program.

Some of the coding guidelines that are followed in a programming language are listed below.

- All the codes should be properly commented before being submitted to the review team.
- All curly braces should start from a new line.
- All class names should start with the abbreviation of each group. For example, AA and CM can be used instead of academic administration and course management, respectively.
- Errors should be mentioned in the following format: [error code]: [explanation]. For example, 0102: null pointer exception, where 0102 indicates the error code and null pointer exception is the name of the error.
- Every 'if statement should be followed by a curly braces even if there exists only a single statement.
- Every file should contain information about the author of the file, modification date, and version information. Similarly, some of the commonly used coding guidelines in a database (organized collection of information that is systematically organized for easy access and analysis) are listed below.
- Table names should start with TBL. For example, TBL_STUDENT.
- If table names contain one word, field names should start with the first three characters of the name of the table. For example, STU_FIRSTNAME.
- Every table should have a primary key.
- Long data type (or database equivalent) should be used for the primary key.

Advantages:

- Increased efficiency • Reduced costs: • Reduced complexity: • Reduced hidden costs• Code reuse • Automated error prevention.

7. Explain manual testing.

Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

- Manual - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager. Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

- Automated This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools. A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually. There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

8. What is unit testing? 9. Write a note on integration testing.

Testing Levels

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified. Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels –

Unit Testing : While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing : Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

System Testing : The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- Functionality testing - Tests all functionalities of the software against the requirement.

- Performance testing - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environment conditions.

• Security & Portability - These tests are done when the software is meant to work on various platforms and accessed by number of persons.

Acceptance Testing When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

• Alpha testing - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.

• Beta testing - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend

10. What is software maintenance? Types of software maintenance.

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

• Market Conditions - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.

• Client Requirements - Over the time, customer may ask for new features or functions in the software.

• Host Modifications - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.

• Organization Changes - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise. Types of maintenance In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

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

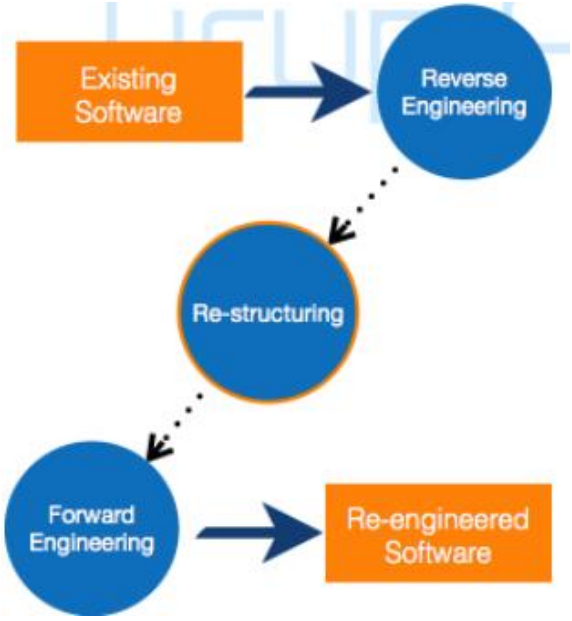
• Adaptive Maintenance - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

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

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

11. What is software re-engineering?

Software Re-engineering When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written. Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not. For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult. Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



Re-Engineering Process

• Decide what to re-engineer. Is it whole software or a part of it?

• Perform Reverse Engineering, in order to obtain specifications of existing software.

• Restructure Program if required. For example, changing function-oriented programs into object-oriented programs.

• Re-structure data as required.

• Apply Forward engineering concepts in order to get re-engineered software

12. Explain different types of requirements.

Software Requirements: Objectives

• To introduce the concepts of user requirements and system requirements

• To describe functional and non-functional requirements

• To explain how software requirements may be organised in a requirements document
Requirements engineering

• The process of finding out, analysing, documenting, and checking the services that the customer requires from a system and its operational constraints is called RE.

• Requirement may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

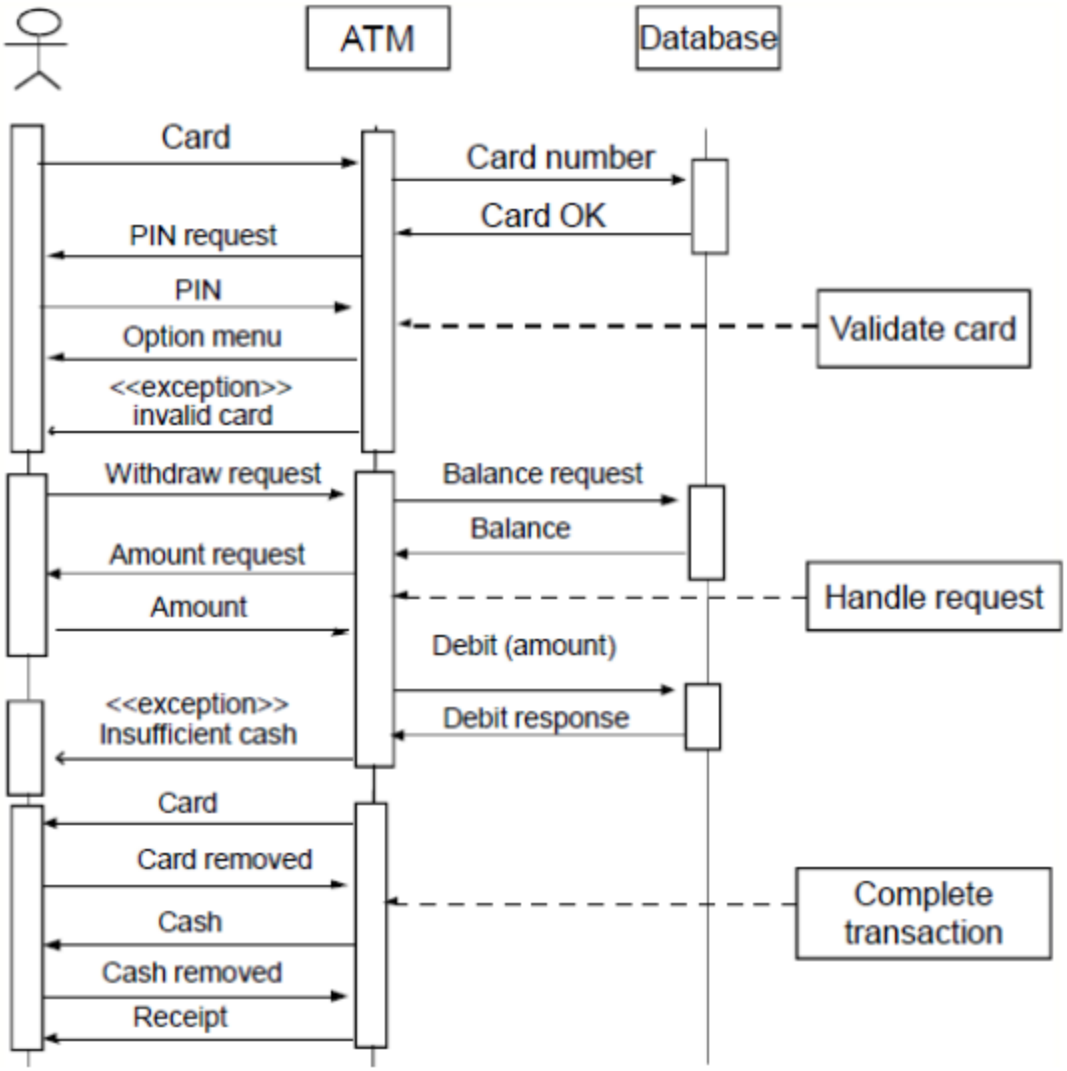
Types of requirement

• User requirements (high level abstract requirements) – Statements in natural language plus diagrams of what services the system provides and its operational constraints.Written for customers.

• System requirements (description of what system should do) – A structured document(also called functional specification) setting out detailed descriptions of the system's functions,services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor

13. Draw and explain sequence diagram of ATM withdrawal.

• Cash withdrawal from an ATM – Validate card : By checking the card number and user's PIN – Handle request : user requests are handled. Query database for withdrawal – Complete transaction: return the card and deliver cash & receipt. Sequence diagram of ATM withdrawal



14. Write a note on Software processes.

Software Processes:

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system. Any software process must include the following four activities:

1. Software specification (or requirements engineering): Define the main functionalities of the software and the constrains around them.

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

3. Software verification and validation: The software must conforms to it's specification and meets the customer needs.

4. Software evolution (software maintenance): The software is being modified to meet customer and market requirements changes. In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc.

There are also supporting activities such as configuration and change management, quality assurance, project management, user experience. Along with other activities aim to improve the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.

When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

1. Products: The outcomes of the an activity. For example, the outcome of architectural design maybe a model for the software architecture.

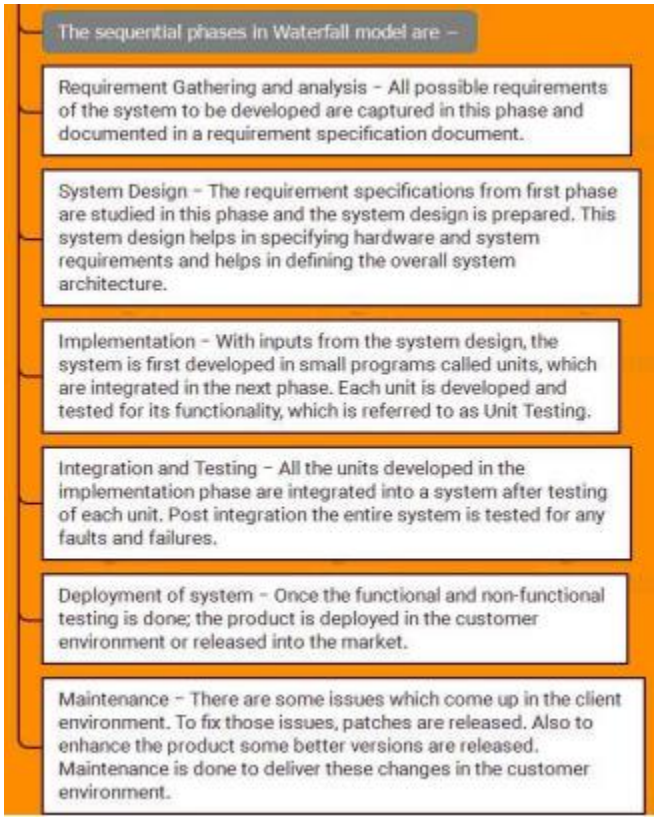
2. Roles: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.

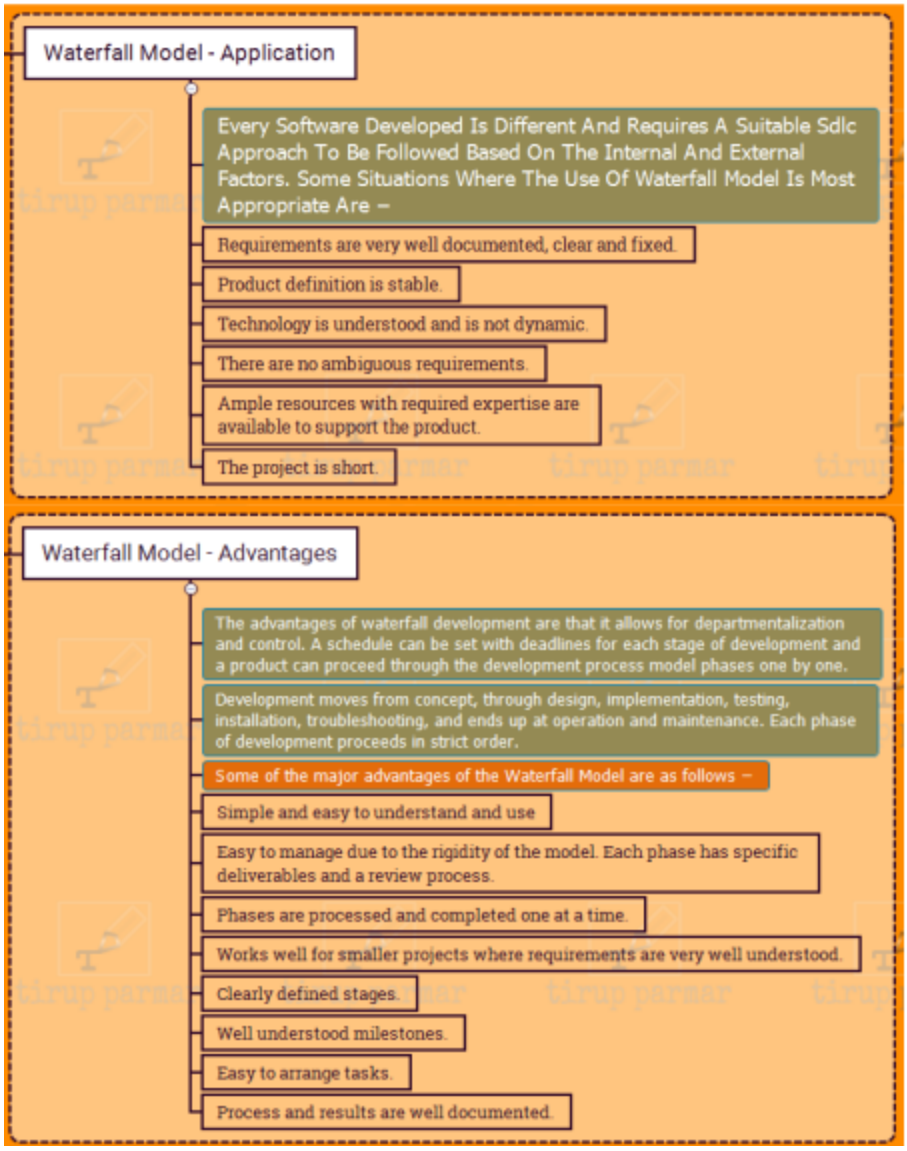
3. Pre and post conditions: The conditions that must be true before and after an activity. For example, the pre condition of the architectural design is the requirements have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

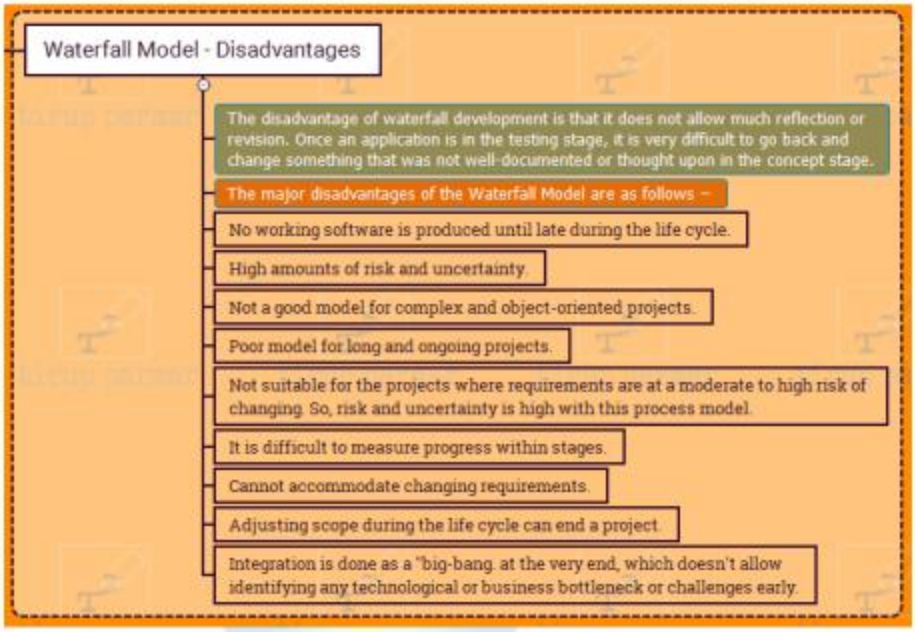
Software process is complex, it relies on making decisions. There’s no ideal process and most organizations have developed their own software process

15. What is waterfall model?

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.




Waterfall Model - Application
Every Software Developed Is Different And Requires A Suitable Sdlc Approach To Be Followed Based On The Internal And External Factors. Some Situations Where The Use Of Waterfall Model Is Most Appropriate Are –
Requirements are very well documented, clear and fixed.
Product definition is stable.
Technology is understood and is not dynamic.
There are no ambiguous requirements.
Ample resources with required expertise are available to support the product.
The project is short.
Waterfall Model - Advantages
The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.
Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.
Some of the major advantages of the Waterfall Model are as follows –
Simple and easy to understand and use
Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
Phases are processed and completed one at a time.
Works well for smaller projects where requirements are very well understood.
Clearly defined stages.
Well understood milestones.
Easy to arrange tasks.
Process and results are well documented.



16. What is prototype model?

PRTOTYPING MODEL Prototype A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system. So, a prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc. In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation. While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to working system.

A software prototype can be used:

[1] In the requirements engineering, a prototype can help with the elicitation and validation of system requirements. It allows the users to experiment with the system, and so, refine the requirements. They may get new ideas for requirements, and find areas of strength and weakness in the software. Furthermore, as the prototype is developed, it may reveal errors and in the requirements. The specification maybe then modified to reflect the changes.

[2] In the system design, a prototype can help to carry out deign experiments to check the feasibility of a proposed design. For example, a database design may be prototype-d and tested to check it supports efficient data access for the most common user queries. Other Needs for a prototype in software development There are several uses of a prototype. An important purpose is

to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs: how the screens might look like how the user interface would behave how the system would produce outputs Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product. A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

A prototype of the actual product is preferred in situations such as:

• User requirements are not complete

• Technical issues are not clear Fig. The process of prototype development

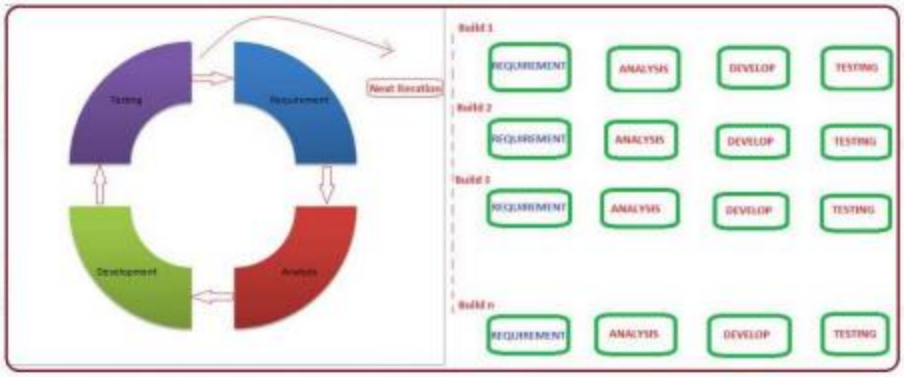
1. Establish objectives: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.

2. Define prototype functionality: Decide what are the inputs and the expected output from a prototype. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype.

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

4. Evaluate the prototype: Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed. Prototyping is not a standalone, complete development methodology, but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc.).

17. What is iterative model?

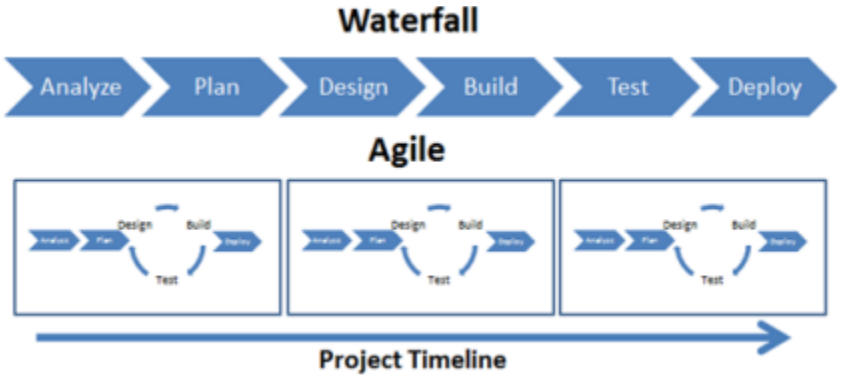


Iterative model

- Requirements may change at a previous stage of development and may be taken care of in the next stage
- Feedback loop at every stage in development
- Regression testing increasingly important on all iterations one after another
- Testing plan to allow more testing at each subsequent delivery phase
- More practical than the waterfall model
- Limitation:
  - o They are many cycles of waterfall model
  - o Fixed price projects have problem of estimation
  - o Product architecture and design becomes fragile due to many iteration

18. What is agile method?

Agility is flexibility, it is a state of dynamic, adapted to the specific circumstances. The agile methods refers to a group of software development models based on the incremental and iterative approach, in which the increments are small and typically, new releases of the system are created and made available to customers every few weeks



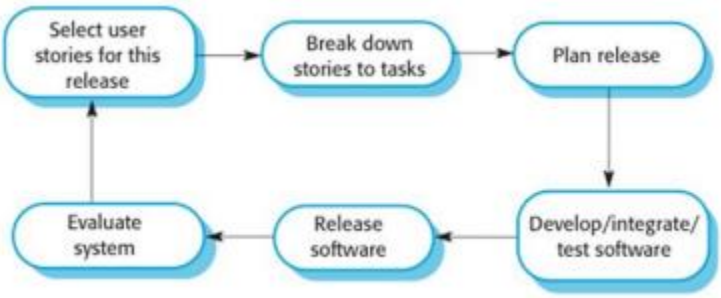
Extreme programming ″

Perhaps the best-known and most widely used agile method. ″ Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.

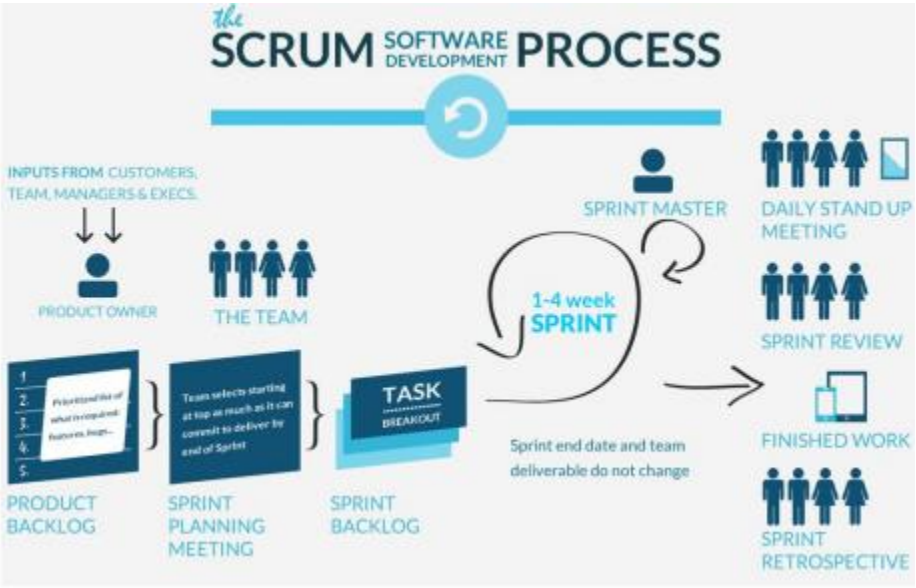
♣ New versions may be built several times per day;

♣ Increments are delivered to customers every 2 weeks

♣ All tests must be run for every build and the build is only accepted if tests run successfully. XP and agile principles ″ Incremental development is supported through small, frequent system releases. ″ Customer involvement means full-time customer engagement with the team. ″ People not process through pair programming, collective ownership and a process that avoids long working hours. ″ Change supported through regular system releases. ″ Maintaining simplicity through constant refactoring of code.



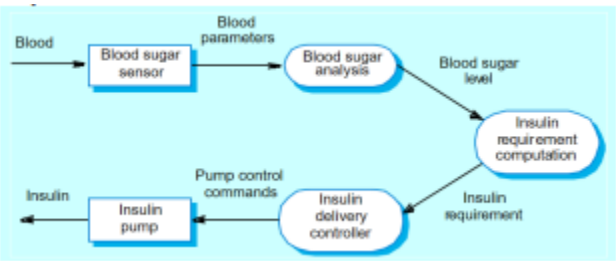
A scrum is an action associated with the game of rugby. It is enacted when players huddle together with the objective of moving the ball strategically towards the goal post. The SCRUM agile development methodology is derived from this action and draws from the rugby scrum some principles which are embedded throughout the life-cycle of a SCRUM enabled project.



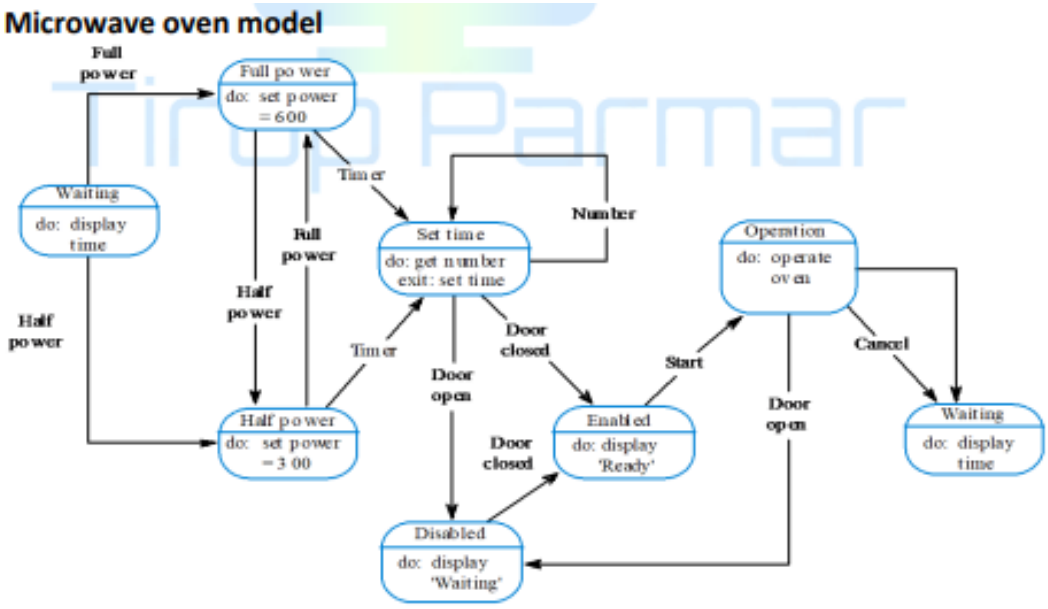
19. Explain data flow diagram.

Data flow diagrams ¬ DFDs model the system from a functional perspective. ¬ Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system. ¬ Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment

Insulin pump DFD



20. Draw and explain microwave oven model.



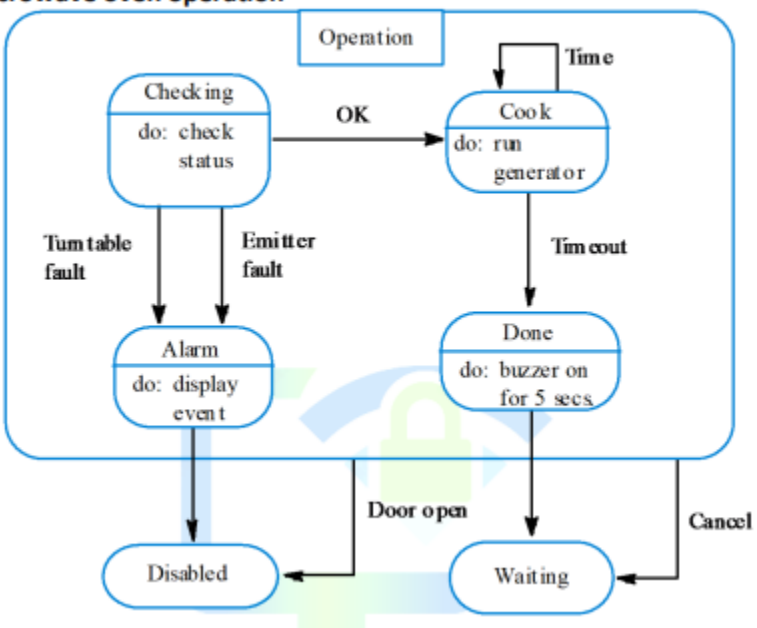
Microwave oven state description

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power'.
Full power	The oven power is set to 600 watts. The display shows 'Full power'.
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

Microwave oven stimuli

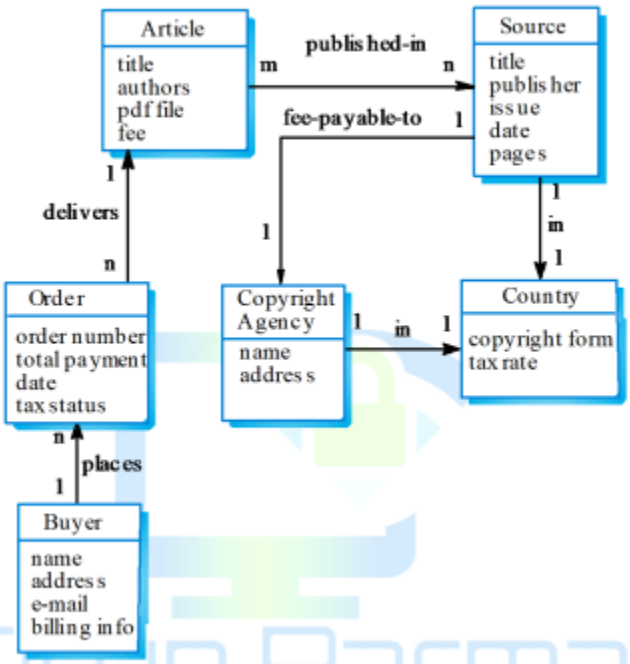
Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button
Timer	The user has pressed one of the timer buttons
Number	The user has pressed a numeric key
Door open	The oven door switch is not closed
Door closed	The oven door switch is closed
Start	The user has pressed the start button
Cancel	The user has pressed the cancel button

Microwave oven operation



21. Draw and explain library semantic model.

Library semantic model



Data dictionary entries

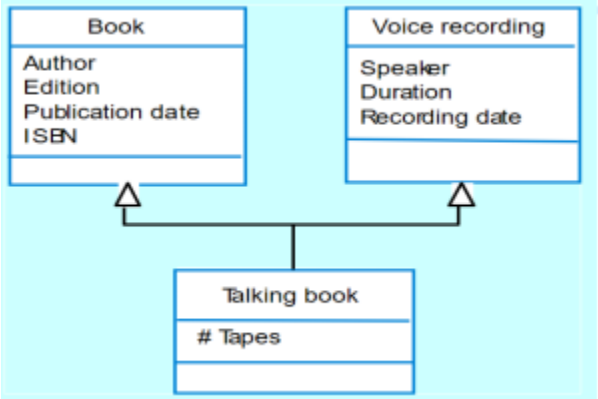
Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a copy of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

22. What is multiple inheritance model?

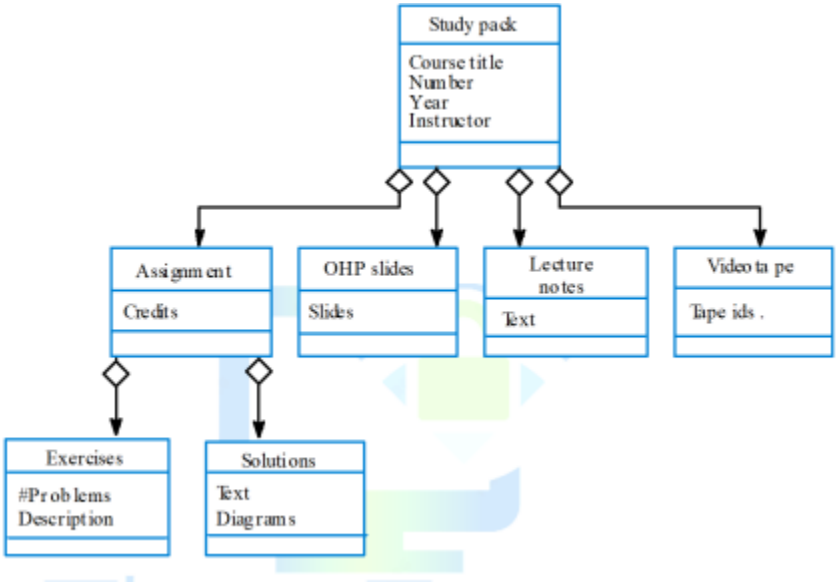
¬ Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super classes.

¬ This can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics.

¬ Multiple inheritance makes class hierarchy reorganisation more complex



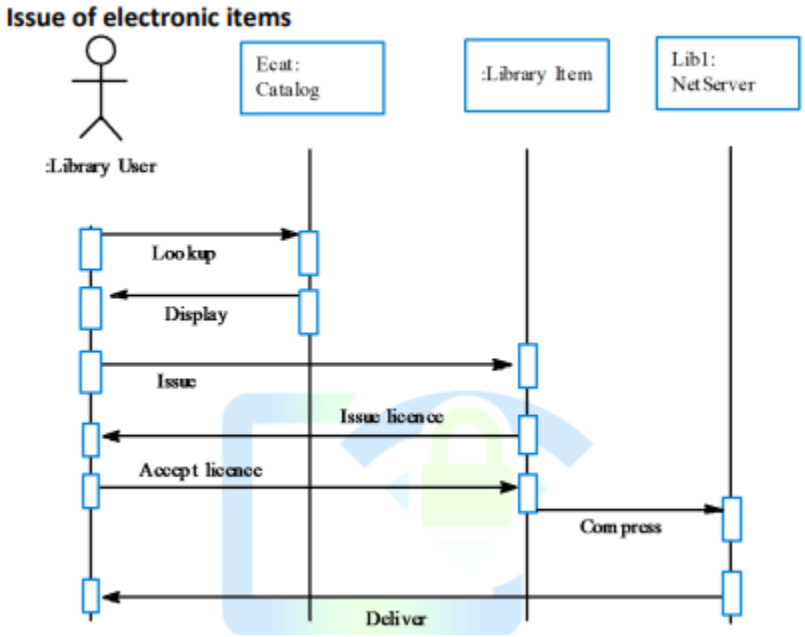
¬ An aggregation model shows how classes that are collections are composed of other classes. ¬ Aggregation models are similar to the part-of relationship in semantic data models.



23. Explain Object behavior modeling.

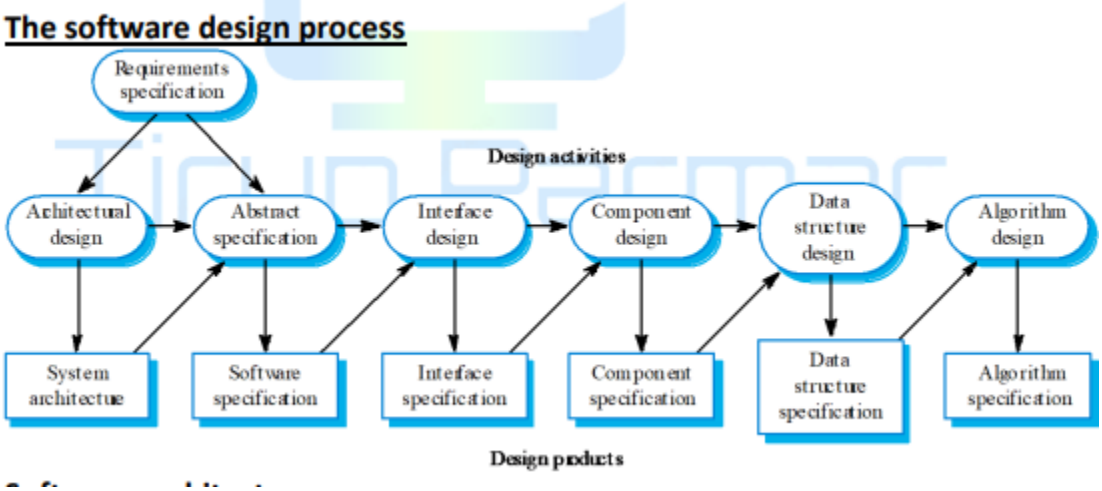
¬ A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case.

¬ Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects.



¬ Elicit requirements • Identify viewpoints • Conduct interviews and ethnographies • Draw up scenarios and use cases ¬ Refine use cases with functional decomposition and identify objects involved in interactions ¬ Build object model ¬ Create sequence diagrams for each use case ¬ Create state diagrams for each object ¬ Group related objects into subsystems

24. Write a note on software design process.



25. What is sub system decomposition?

Subsystem decomposition

- Concerned with decomposing the system into interacting sub-systems.
- The architectural design is normally expressed as a block diagram presenting an overview of the system structure.
- More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed.

26. What is repository model?

The repository model

• Sub-systems must exchange data.

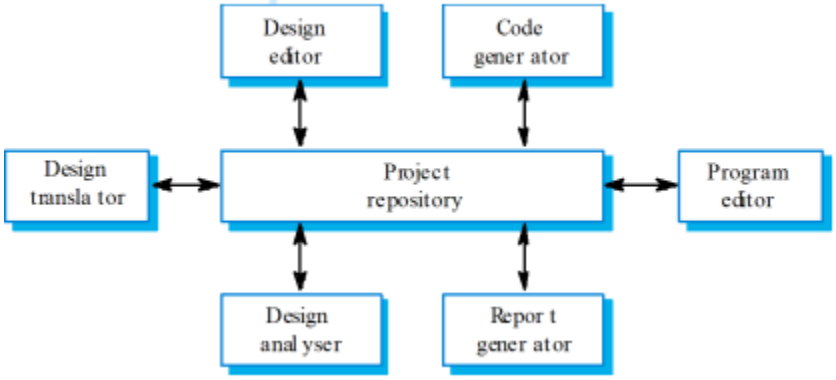
This may be done in two ways:

o Shared data is held in a central database or repository and may be accessed by all sub-systems;

o Each sub-system maintains its own database and passes data explicitly to other sub-systems.

• When large amounts of data are to be shared, the repository model of sharing is most commonly used.

CASE toolset architecture



Repository model characteristics

- Advantages

o Efficient way to share large amounts of data;

o Sub-systems need not be concerned with how data is produced Centralised management e.g. backup, security, etc.

o Sharing model is published as the repository schema.

- Disadvantages

o Sub-systems must agree on a repository data model. Inevitably a compromise;

o Data evolution is difficult and expensive;

o No scope for specific management policies;

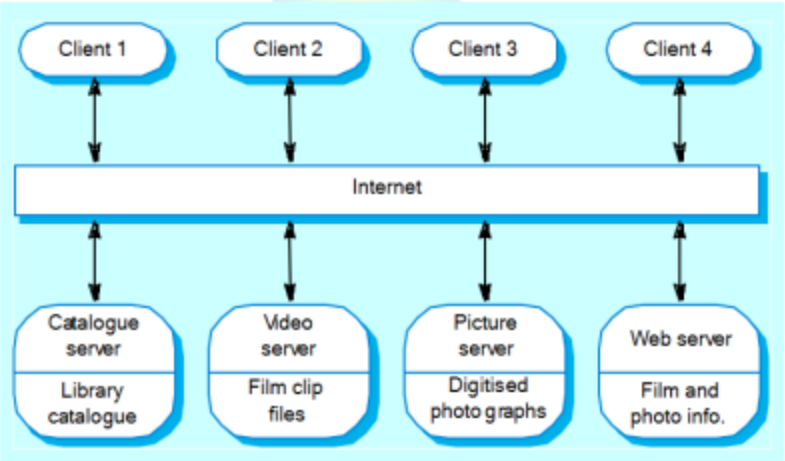
o Difficult to distribute efficiently.

27. What is client server model?

Client-server model

- Distributed system model which shows how data and processing is distributed across a range of components.
- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

Film and picture library



Client-server characteristics

- Advantages
  - Distribution of data is straightforward;
  - Makes effective use of networked systems. May require cheaper hardware;
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;
  - Redundant management in each server;
  - No central register of names and services - it may be hard to find out what servers and services are available.

28. What are different decomposition styles?

Modular decomposition styles

• Styles of decomposing sub-systems into modules.

• No rigid distinction between system organisation and modular decomposition. Sub-systems and modules

• A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems

. • A module is a system component that provides services to other components but would not normally be considered as a separate system.

Modular decomposition

• Another structural level where sub-systems are decomposed into modules.

• Two modular decomposition models covered

o An object model where the system is decomposed into interacting object;

o A pipeline or data-flow model where the system is decomposed into functional modules which transform inputs to outputs.

- If possible, decisions about concurrency should be delayed until modules are implemented.

Object models

- Structure the system into a set of loosely coupled objects with well-defined interfaces.
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- When implemented, objects are created from these classes and some control model used to coordinate object operations.

29. Write a note on control styles.

- Are concerned with the control flow between sub-systems. Distinct from the system decomposition model

- **Centralised control**

o One sub-system has overall responsibility for control and starts and stops other sub-systems.

- **Event-based control**

o Each sub-system can respond to externally generated events from other sub systems or the system's environment.

Centralised control

- A control sub-system takes responsibility for managing the execution of other sub systems.

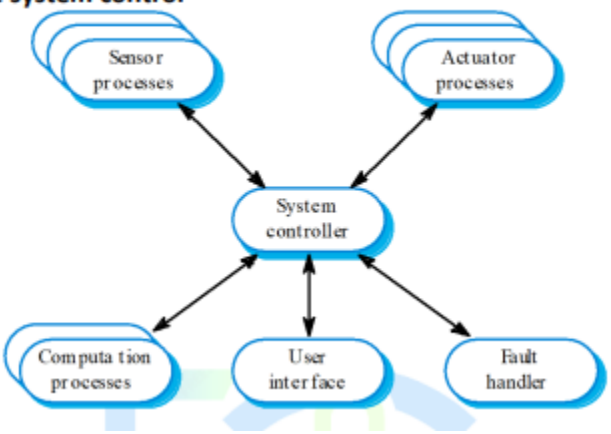
- **Call-return model**

o Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems.

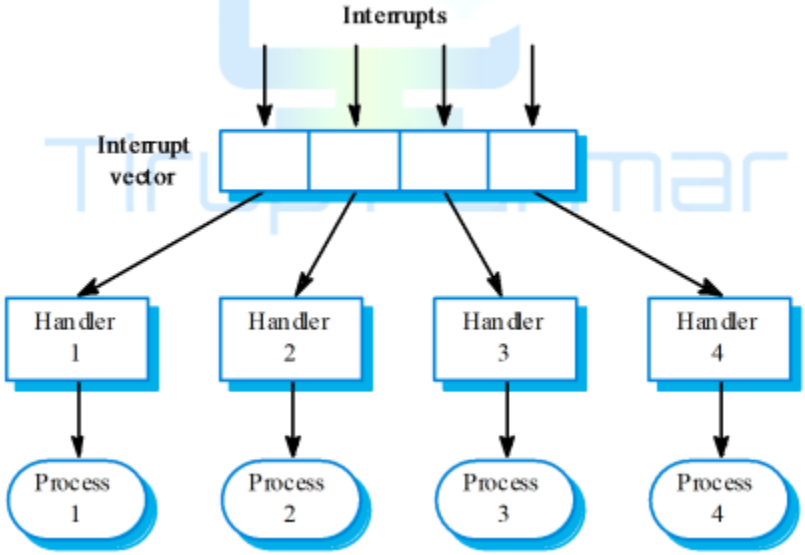
- **Manager model**

o Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement.

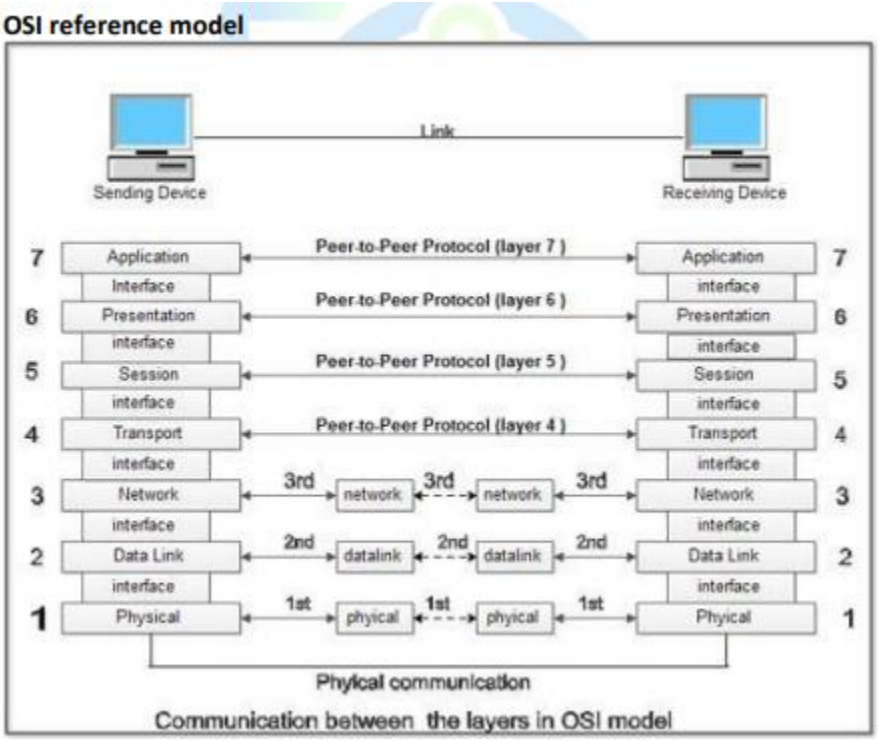
Real-time system control



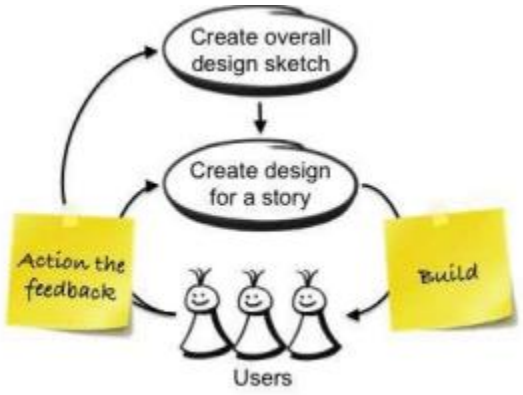
Interrupt-driven control



30. Explain OSI reference model.



31. What is user interface design process? What are 3 golden rules of user interface design?

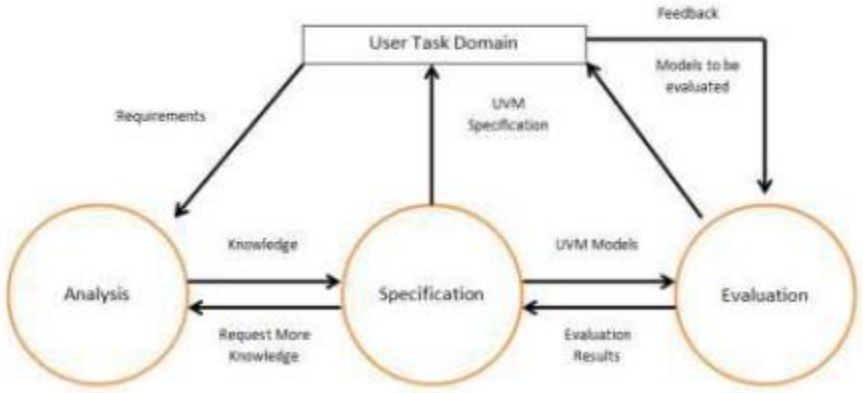


- Interface inconsistency can cost a big company millions of dollars in lost productivity and increased support costs.
- The software becomes more popular if its user interface is:
  - Attractive
  - Simple to use
  - Responsive in short time
  - Clear to understand
  - Consistent on all interfacing screens

Background

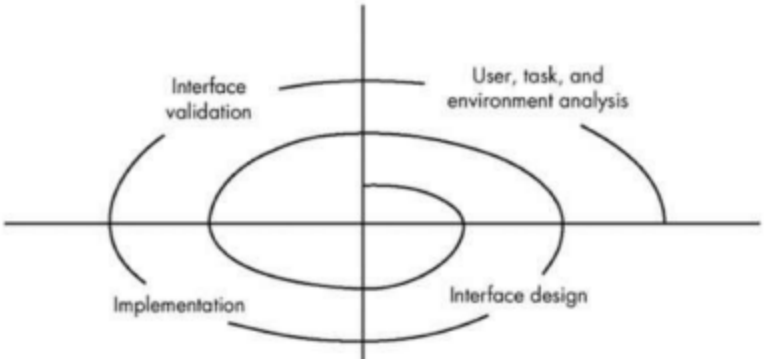
- Interface design focuses on the following
  - The design of interfaces between software components
  - The design of interfaces between the software and other nonhuman producers and consumers of information
  - The design of the interface between a human and the computer
- Graphical user interfaces (GUIs) have helped to eliminate many of the most horrific interface problems
- However, some are still difficult to learn, hard to use, confusing, counterintuitive, unforgiving, and frustrating
- User interface analysis and design has to do with the study of people and how they relate to technology

Interface Design Process



- User interface development follows a spiral process
  - Interface analysis (user, task, and environment analysis)
    - Focuses on the profile of the users who will interact with the system
    - Concentrates on users, tasks, content and work environment
    - Studies different models of system function (as perceived from the outside)

  - Delineates the human- and computer-oriented tasks that are required to achieve system function
  - Interface design
    - Defines a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system
  - Interface construction
    - Begins with a prototype that enables usage scenarios to be evaluated
    - Continues with development tools to complete the construction
  - Interface validation, focuses on
    - The ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements
    - The degree to which the interface is easy to use and easy to learn
    - The users' acceptance of the interface as a useful tool in their work



The Golden Rules of User Interface Design

The three areas of user interface design principles are:

- **1.** Place users in control of the interface
- **2.** Reduce users' memory load
- **3.** Make the user interface consistent.

1. Place the User in Control

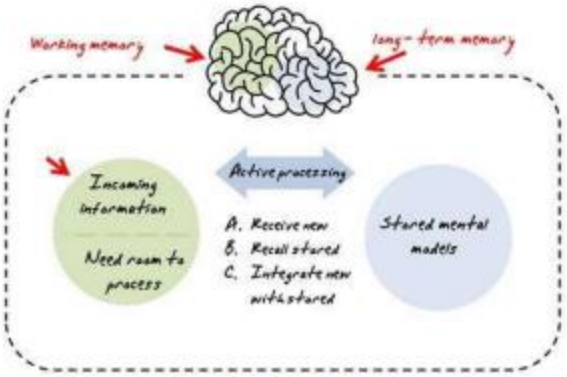
- Define interaction modes in a way that does not force a user into unnecessary or undesired actions
  - The user shall be able to enter and exit a mode with little or no effort (e.g., spell check → edit text → spell check)
- Provide for flexible interaction

  - The user shall be able to perform the same action via keyboard commands, mouse movement, or voice recognition
- Allow user interaction to be interruptible and "undo"able
  - The user shall be able to easily interrupt a sequence of actions to do something else (without losing the work that has been done so far)
  - The user shall be able to "undo" any action
- Streamline interaction as skill levels advance and allow the interaction to be customized
  - The user shall be able to use a macro mechanism to perform a sequence of repeated interactions and to customize the interface
- Hide technical internals from the casual user
  - The user shall not be required to directly use operating system, file management, networking. etc., commands to perform any actions. Instead, these operations shall be hidden from the user and performed "behind the scenes" in the form of a real-world abstraction
- Design for direct interaction with objects that appear on the screen

  - The user shall be able to manipulate objects on the screen in a manner similar to what would occur if the object were a physical thing (e.g., stretch a rectangle, press a button, move a slider)

2. Reduce the User's Memory Load

- Reduce demand on short-term memory
  - The interface shall reduce the user's requirement to remember past actions and results by providing visual cues of such actions
- Establish meaningful defaults
  - The system shall provide the user with default values that make sense to the average user but allow the user to change these defaults
  - The user shall be able to easily reset any value to its original default value
- Define shortcuts that are intuitive
  - The user shall be provided mnemonics (i.e., control or alt combinations) that tie easily to the action in a way that is easy to remember such as the first letter



- The visual layout of the interface should be based on a real world metaphor
  - The screen layout of the user interface shall contain well-understood visual cues that the user can relate to real-world actions
- Disclose information in a progressive fashion
  - When interacting with a task, an object or some behavior, the interface shall be organized hierarchically by moving the user progressively in a step-wise fashion from an abstract concept to a concrete action (e.g., text format options → format dialog box)

3.Make the Interface Consistent

- The interface should present and acquire information in a consistent fashion
  - All visual information shall be organized according to a design standard that is maintained throughout all screen displays
  - Input mechanisms shall be constrained to a limited set that is used consistently throughout the application
  - Mechanisms for navigating from task to task shall be consistently defined and implemented
- Allow the user to put the current task into a meaningful context
  - The interface shall provide indicators (e.g., window titles, consistent color coding) that enable the user to know the context of the work at hand
  - The user shall be able to determine where he has come from and what alternatives exist for a transition to a new task

Summary: Golden Rules

- *Place User in Control*
  - Define interaction in such a way that the user is not forced into performing unnecessary or undesired actions
  - Provide for flexible interaction (users have varying preferences)
  - Allow user interaction to be interruptible and reversible
  - Streamline interaction as skill level increases and allow customization of interaction
  - Hide technical internals from the casual user
  - Design for direct interaction with objects that appear on the screen
- *Reduce User Cognitive (Memory) Load*
  - Reduce demands on user's short-term memory
  - Establish meaningful defaults
  - Define intuitive short-cuts
  - Visual layout of user interface should be based on a familiar real world metaphor
  - Disclose information in a progressive fashion
- *Make Interface Consistent*
  - Allow user to put the current task into a meaningful context
  - Maintain consistency across a family of applications
  - If past interaction models have created user expectations, do not make changes unless there is a good reason to do so

---

Norman's *Seven Stages of Action* that explain how people do things:

1. Form a goal
2. Form the intention
3. Specify an action
4. Execute the action
5. Perceive the state of the world
6. Interpret the state of the world
7. Evaluate the outcome

---

32. What questions should be answered during user analysis process?

1) Are the users trained professionals, technicians, clerical or manufacturing workers?
2) What level of formal education does the average user have?
3) Are the users capable of learning on their own from written materials or have they expressed a desire for classroom training?
4) Are the users expert typists or are they keyboard phobic?
5) What is the age range of the user community?
6) Will the users be represented predominately by one gender?
7) How are users compensated for the work they perform or are they volunteers?
8) Do users work normal office hours, or do they work whenever the job is required?
9) Is the software to be an integral part of the work users do, or will it be used only occasionally?
10) What is the primary spoken language among users?
11) What are the consequences if a user makes a mistake using the system?
12) Are users experts in the subject matter that is addressed by the system?
13) Do users want to know about the technology that sits behind the interface?

33. What is error message?

What are guidelines for error messages?

Guidelines for Error Messages

An effective error message philosophy can do much to improve the quality of an interactive system and will significantly reduce user frustration when problems do occur

- The message should describe the problem in plain language that a typical user can understand
- The message should provide constructive advice for recovering from the error
- The message should indicate any negative consequences of the error (e.g., potentially corrupted data files) so that the user can check to ensure that they have not occurred (or correct them if they have)
- The message should be accompanied by an audible or visual cue such as a beep, momentary flashing, or a special error color
- The message should be non-judgmental
  - The message should never place blame on the user

34. What is software project management?

Software project management

- Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Success criteria

- Deliver the software to the customer at the agreed time.
- Keep overall costs within budget.
- Deliver software that meets the customer's expectations.
- Maintain a happy and well-functioning development team.

Software management distinctions

- The product is intangible.
  - Software cannot be seen or touched. Software project managers cannot see progress by simply looking at the artefact that is being constructed.
- Many software projects are 'one-off' projects.
  - Large software projects are usually different in some ways from previous projects. Even managers who have lots of previous experience may find it difficult to anticipate problems.
- Software processes are variable and organization specific.
  - We still cannot reliably predict when a particular software process is likely to lead to development problems.

35. What are different management activities?

Management activities

- *Project planning*
  - Project managers are responsible for planning. estimating and scheduling project development and assigning people to tasks.
- *Reporting*
  - Project managers are usually responsible for reporting on the progress of a project to customers and to the managers of the company developing the software.
- *Risk management*
  - Project managers assess the risks that may affect a project, monitor these risks and take action when problems arise.
- *People management*
  - Project managers have to choose people for their team and establish ways of working that leads to effective team performance

- *Proposal writing*
  - The first stage in a software project may involve writing a proposal to win a contract to carry out an item of work. The proposal describes the objectives of the project and how it will be carried out.

36. Explain risk management.

Risk management

✧ Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.

✧ A risk is a probability that some adverse circumstance will occur

- Project risks affect schedule or resources;
- Product risks affect the quality or performance of the software being developed;
- Business risks affect the organisation developing or procuring the software.

Examples of common project, product, and business risks

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organizational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool underperformance	Product	CASE tools, which support the project, do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The risk management process

✧ Risk identification
- Identify project, product and business risks;

✧ Risk analysis
- Assess the likelihood and consequences of these risks;

✧ Risk planning
- Draw up plans to avoid or minimise the effects of the risk;

✧ Risk monitoring
- Monitor the risks throughout the project;

Risk identification

✧ May be a team activities or based on the individual project manager's experience.

✧ A checklist of common risks may be used to identify risks in a project
- Technology risks.
- People risks.
- Organisational risks.
- Requirements risks.
- Estimation risks.

Risk analysis

- Assess probability and seriousness of each risk.
- Probability may be very low, low, moderate, high or very high.

- Risk consequences might be catastrophic, serious, tolerable or insignificant.

Risk planning

- Consider each risk and develop a strategy to manage that risk.
- Avoidance strategies
  - The probability that the risk will arise is reduced;
- Minimisation strategies
  - The impact of the risk on the project or product will be reduced;
- Contingency plans
  - If the risk arises, contingency plans are plans to deal with that risk;

Risk monitoring

- Assess each identified risks regularly to decide whether or not it is becoming less or more probable.
- Also assess whether the effects of the risk have changed.
- Each key risk should be discussed at management progress meetings.

Risk indicators

Risk type	Potential indicators
Technology	Late delivery of hardware or support software; many reported technology problems.
People	Poor staff morale; poor relationships amongst team members; high staff turnover.
Organizational	Organizational gossip; lack of action by senior management.
Tools	Reluctance by team members to use tools; complaints about CASE tools; demands for higher-powered workstations.
Requirements	Many requirements change requests; customer complaints.
Estimation	Failure to meet agreed schedule; failure to clear reported defects.

37. How to motivate people in an organization?

Motivating people

- An important role of a manager is to motivate the people working on a project.
- Motivation means organizing the work and the working environment to encourage people to work effectively.
  - If people are not motivated, they will not be interested in the work they are doing. They will work slowly, be more likely to make mistakes and will not contribute to the broader goals of the team or the organization.
- Motivation is a complex issue but it appears that their are different types of motivation based on:
  - Basic needs (e.g. food, sleep, etc.);
  - Personal needs (e.g. respect, self-esteem);
  - Social needs (e.g. to be accepted as part of a group).

Need satisfaction

- In software development groups, basic physiological and safety needs are not an issue.
- Social
  - Provide communal facilities;
  - Allow informal communications e.g. via social networking
- Esteem
  - Recognition of achievements;
  - Appropriate rewards.
- Self-realization
  - Training - people want to learn more;
  - Responsibility.

Personality types

- The needs hierarchy is almost certainly an oversimplification of motivation in practice.
- Motivation should also take into account different personality types:
  - Task-oriented;
  - Self-oriented;
  - Interaction-oriented.
- Task-oriented.
  - The motivation for doing the work is the work itself;
- Self-oriented.
  - The work is a means to an end which is the achievement of individual goals - e.g. to get rich, to play tennis, to travel etc.;
- Interaction-oriented
  - The principal motivation is the presence and actions of co-workers. People go to work because they like to go to work.

Motivation balance

- Individual motivations are made up of elements of each class.
- The balance can change depending on personal circumstances and external events.
- However, people are not just motivated by personal factors but also by being part of a group and culture.
- People go to work because they are motivated by the people that they work with.

38. How team spirit is important?

Teamwork

- Most software engineering is a group activity
  - The development schedule for most non-trivial software projects is such that they cannot be completed by one person working alone.
- A good group is cohesive and has a team spirit. The people involved are motivated by the success of the group as well as by their own personal goals.
- Group interaction is a key determinant of group performance.
- Flexibility in group composition is limited
  - Managers must do the best they can with available people.

Group cohesiveness

- In a cohesive group, members consider the group to be more important than any individual in it.
- The advantages of a cohesive group are:
  - Group quality standards can be developed by the group members.
  - Team members learn from each other and get to know each other's work; Inhibitions caused by ignorance are reduced.
  - Knowledge is shared. Continuity can be maintained if a group member leaves.
  - Refactoring and continual improvement is encouraged. Group members work collectively to deliver high quality results and fix problems, irrespective of the individuals who originally created the design or program.

39. What is software quality? What are quality factors to be considered?

Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product.
- Involves defining appropriate quality standards and procedures and ensuring that these are followed.
- Should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility.

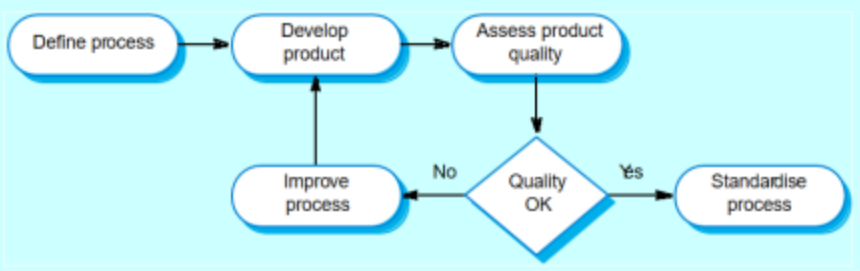
What is quality?

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.
- Software quality management procedures cannot rely on having perfect specifications.

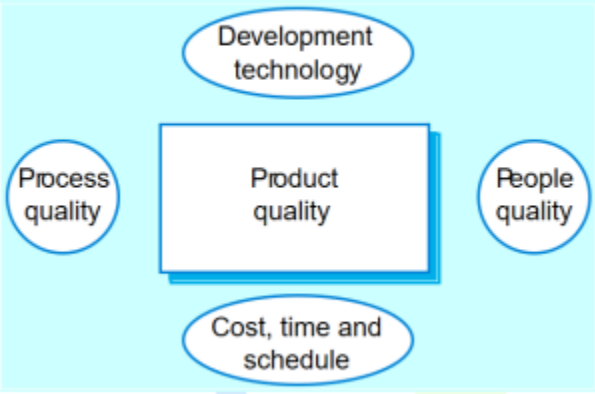
Process and product quality

- The quality of a developed product is influenced by the quality of the production process.
- A good process is usually required to produce a good product.
  - For manufactured goods, this link is straightforward.
  - But for software, this link is more complex and poorly understood.

Process-based product quality



Principal product quality factors



40. What are different quality management activities? Explain any one.

- **Quality management activities**
  - Quality assurance and standards
  - Quality planning
  - Quality control

Prof. Tirup Parmar

Quality assurance and standards

- Standards are important for effective quality management.
  - Encapsulation of best practice – avoids repetition of past mistakes.
  - They are a framework for quality assurance processes – they involve checking compliance to standards.
  - They provide continuity – new staff can understand the organisation by understanding the standards that are used.

Quality standards

- Standards may be international, national, organizational or project standards.
- **Product standards** define characteristics that all components should exhibit e.g. a common programming style.
- **Process standards** define how the software process should be enacted.

Product and process standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of documents to CM
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with standards

- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious manual work is often involved to maintain the documentation associated with the standards.

Standards development

- Involve practitioners in development. Engineers should understand the rationale underlying a standard.

- Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have associated tool support. Excessive clerical work is the most significant complaint against standards.

ISO 9000

- An international set of standards for quality management.
- Applicable to a range of organisations from manufacturing to service industries.
- ISO 9001 applicable to organisations which design, develop and maintain products.
- ISO 9001 is a generic model of the quality process that must be instantiated for each organisation using the standard.

ISO 9001

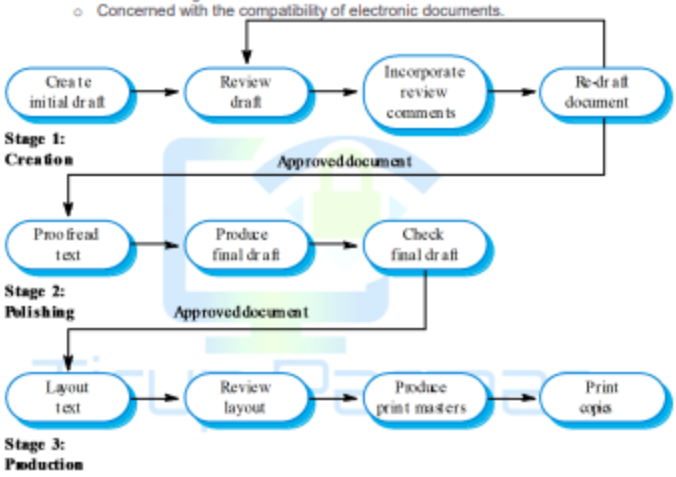
Management responsibility	Quality system
Control of non-conforming products	Design control
Handling, storage, packaging and delivery	Purchasing
Purchaser-supplied products	Product identification and traceability
Process control	Inspection and testing
Inspection and test equipment	Inspection and test status
Contract review	Corrective action
Document control	Quality records
Internal quality audits	Training
Servicing	Statistical techniques

ISO 9000 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognised.

Documentation standards

- Particularly important – documents are the tangible manifestation of the software.
- Documentation process standards
  - Concerned with how documents should be developed, validated and maintained.
- Document standards
  - Concerned with document contents, structure, and appearance.
- Document interchange standards
  - Concerned with the compatibility of electronic documents.



Document standards

- Document identification standards
  - How documents are uniquely identified.
- Document structure standards
  - Standard structure for project documents.
- Document presentation standards
  - Define fonts and styles, use of logos, etc.
- Document update standards
  - Define how changes from previous versions are reflected in a document.

Quality planning

- The process of developing a quality plan for a project.
- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.

Quality control

- This involves checking the software development process to ensure that procedures and standards are being followed.
- There are two approaches to quality control
  - Quality reviews;
  - Objective software assessment and software measurement.

Quality reviews

- This is the principal method of validating the quality of a process or of a product.
- A group examines part or all of a process or system and its documentation to find potential problems.
- There are different types of review with different objectives
  - Inspections for defect removal (product);
  - Reviews for progress assessment (product and process);
  - Quality reviews (product and standards).

41. Write a note on Software measurement and metrics.

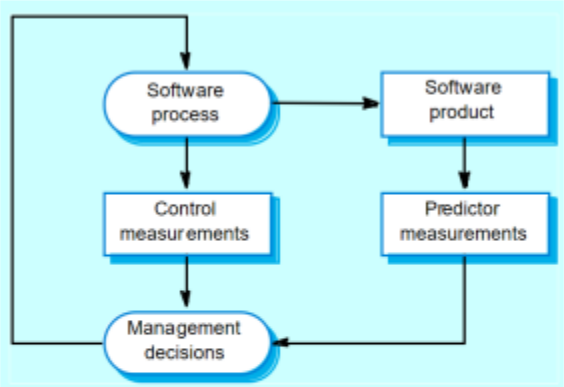
Software measurement and metrics

- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don't make systematic use of software measurement.
- There are few established standards in this area.

Software metric

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.

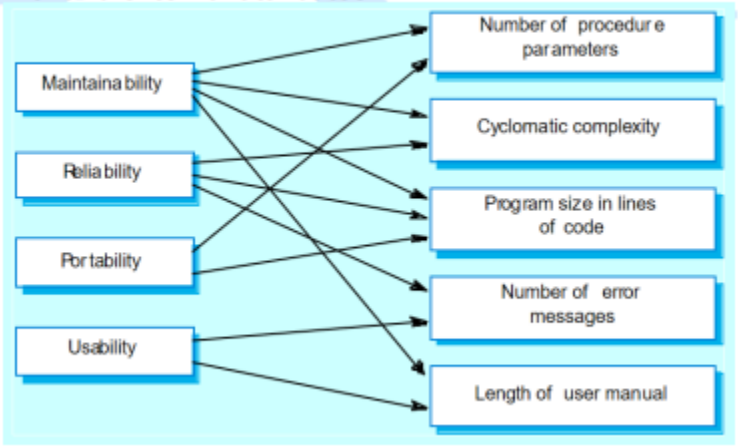
Predictor and control metrics



Metrics assumptions

- A software property can be measured.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

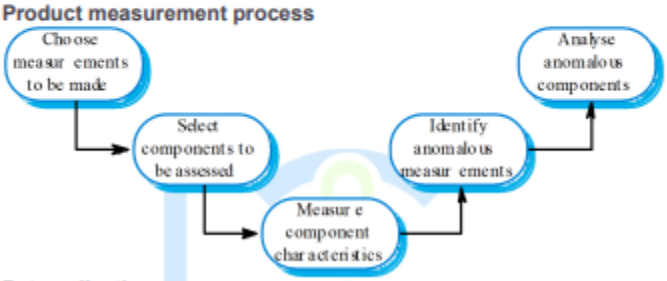
Internal and external attributes



The measurement process

- A software measurement process may be part of a quality control process.
- Data collected during this process should be maintained as an organisational resource.
- Once a measurement database has been established, comparisons across projects become possible.

Product measurement process



Data collection

Software product metrics

Software metric	Description
Fan in/Fan-out	Fan-in is a measure of the number of functions or methods that call some other function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss how to compute cyclomatic complexity in Chapter 22.
Length of identifiers	This is a measure of the average length of distinct identifiers in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document is to understand.

42. Write a note on process improvement activities.

- **Process improvement activities**
  - Process measurement
  - Process analysis and modelling
  - Process change

Process improvement

- Understanding existing processes and introducing process changes to improve product quality, reduce costs or accelerate schedules.
- Most process improvement work so far has focused on defect reduction. This reflects the increasing attention paid by industry to quality.
- However, other process attributes can also be the focus of improvement

Process classification

Process measurement

- Wherever possible, quantitative process data should be collected
  - However, where organisations do not have clearly defined process standards this is very difficult as you don't know what to measure. A process may have to be defined before any measurement is possible.
- Process measurements should be used to assess process improvements
  - But this does not mean that measurements should drive the improvements. The improvement driver should be the organizational objectives.

Process attributes

Process characteristic	Description
Understandability	To what extent is the process explicitly defined and how easy is it to understand the process definition?
Visibility	Do the process activities culminate in clear results so that the progress of the process is externally visible?
Supportability	To what extent can CASE tools be used to support the process activities?
Acceptability	Is the defined process acceptable to and usable by the engineers responsible for producing the software product?
Reliability	Is the process designed in such a way that process errors are avoided or trapped before they result in product errors?
Robustness	Can the process continue in spite of unexpected problems?

43. What is process analysis?

Process analysis and modelling

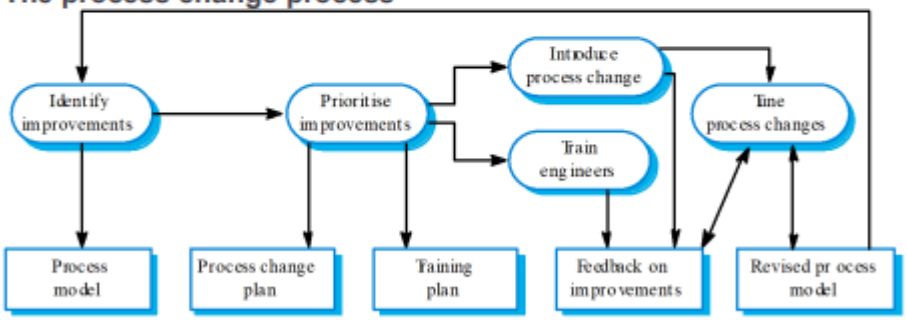
- Process analysis
  - The study of existing processes to understand the relationships between parts of the process and to compare them with other processes.
- Process modelling
  - The documentation of a process which records the tasks, the roles and the entities used;
  - Process models may be presented from different perspectives.
- Study an existing process to understand its activities.
- Produce an abstract model of the process. You should normally represent this graphically. Several different views (e.g. activities, deliverables, etc.) may be required.
- Analyse the model to discover process problems. This involves discussing process activities with stakeholders and discovering problems and possible process changes.

44. What is process change process?

Process change

- Involves making modifications to existing processes.
- This may involve:
  - Introducing new practices, methods or processes;
  - Changing the ordering of process activities;
  - Introducing or removing deliverables;
  - Introducing new roles or responsibilities.
- Change should be driven by measurable goals.

The process change process



Process change stages

- Improvement identification.
- Improvement prioritisation.
- Process change introduction.
- Process change training.
- Change tuning.

45. Differentiate between verification and validation.

Verification and Validation

Verification	Validation
Are we building the system right?	Are we building the right system?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	**Validation** is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.
Following activities are involved in **Verification**: Reviews, Meetings and Inspections.	Following activities are involved in **Validation**: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check whether implementation software is as per specification document or not.	**Validation** is carried out by testing team.
Execution of code is not comes under **Verification**.	Execution of code is comes under **Validation**.
Verification process explains whether the outputs are according to inputs or not.	**Validation** process describes whether the software is accepted by the user or not.
Verification is carried out before the Validation.	**Validation** activity is carried out just after the Verification.
Following items are evaluated during **Verification**: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during **Validation**: Actual product or Software under test.

46. Write a note on testing and debugging.

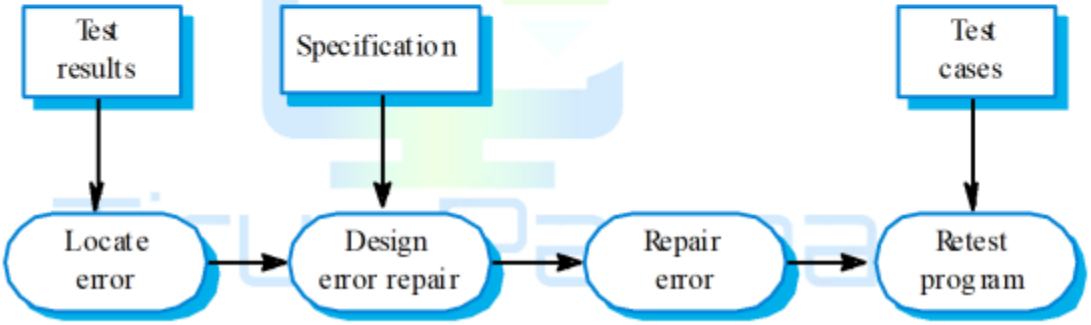
Types of testing

- **Defect testing**
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
- **Validation testing**
  - Intended to show that the software meets its requirements.
  - A successful test is one that shows that a requirements has been properly implemented.

Testing and debugging

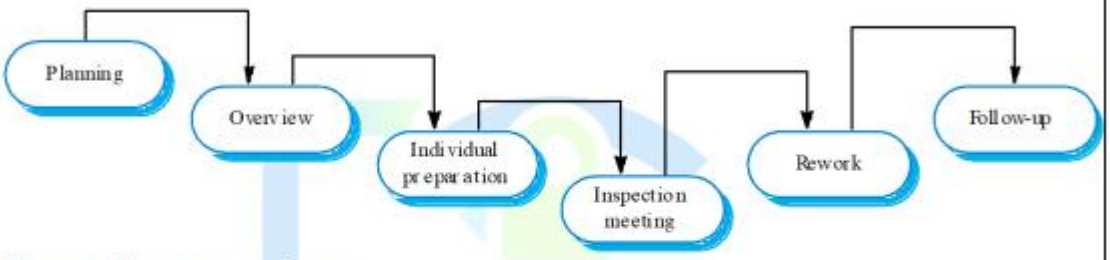
- Defect testing and debugging are distinct processes.
- Verification and validation is concerned with establishing the existence of defects in a program.
- Debugging is concerned with locating and repairing these errors.
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error.

The debugging process



47. What do you mean by inspection process, procedure and roles.

The inspection process



Inspection procedure

- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance.
- Inspection takes place and discovered errors are noted.
- Modifications are made to repair discovered errors.
- Re-inspection may or may not be required.

Inspection roles

Author or owner	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

48. Explain software testing life cycle.

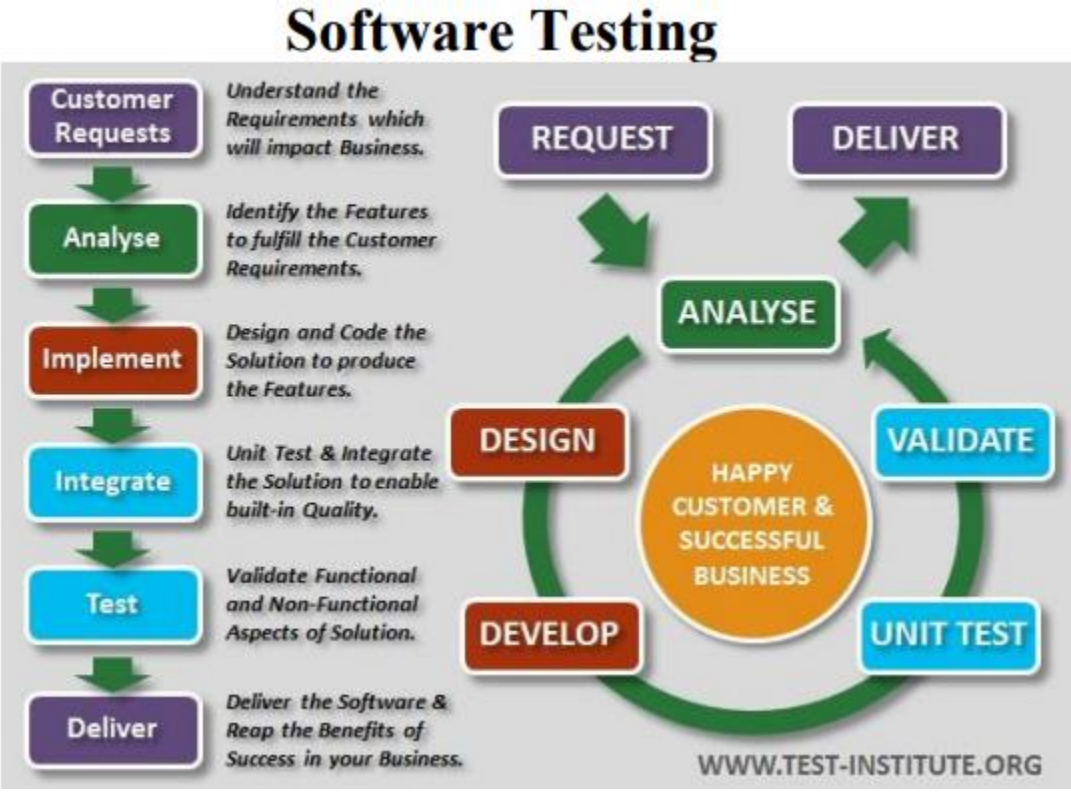


Fig. ***Software Testing Methodology in Software Engineering***

- Software testing is nothing but an art of investigating software to ensure that its quality under test is in line with the requirement of the client.
- Software testing is carried out in a systematic manner with the intent of finding defects in a system. It is required for evaluating the system.
- As the technology is advancing we see that everything is getting digitized. You can access your bank online, you can shop from the comfort of your home, and the options are endless.
- Have you ever wondered what would happen if these systems turn out to be defective?One small defect can cause a lot of financial loss.
- It is for this reason that software testing is now emerging as a very powerful field in IT.
- Although like other products software never suffers from any kind of wear or tear or corrosion but yes, design errors can definitely make your life difficult if they go undetected.
- Regular testing ensures that the software is developed as per the requirement of the client.
- However, if the software is shipped with bugs embedded in it, you never know when they can create a problem and then it will be very difficult to rectify defect because scanning hundreds and thousands of lines of code and fixing a bug is not an easy task.

49. What is top-down estimation? 50. What is bottom-up estimation?

Top-down and bottom-up estimation

- Any of these approaches may be used top-down or bottom-up
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems
- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate

Top-down estimation

- Usable without knowledge of the system architecture and the components that might be part of the system
- Takes into account costs such as integration, configuration management and documentation
- Can underestimate the cost of solving difficult low-level technical problems

Bottom-up estimation

- Usable when the architecture of the system is known and components identified
- Accurate method if the system has been designed in detail
- May underestimate costs of system level activities such as integration and documentation

51. Explain cocomo model.

The COCOMO model

- An empirical model based on project experience
- Well-documented, 'independent' model which is not tied to a specific software vendor
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 81

Project complexity	Formula	Description
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Ωελλ-υνδερστοοδ απλιχατιονσ δεωελοπεδ βψ σμαλλτεαμσ.
Μοδερατε	$PM = 3.0\ (KΔΣI)^{1.12} \times M$	Μορε χομπλεξ προφεχτωηερε τεαμ μεμβερσ μαψ ηαπε λιμιτεδ εξπεριενχε οφ ρελατεδ σψστεμσ.
Εμβεδδεδ	$PM = 3.6\ (KΔΣI)^{1.20} \times M$	Χομπλεξ προφεχτωηερε τηε σοφτωαρε ισ περφορμινγ χουπλεδ χομπλεξ οτηερ σοφτωαρε, ρεγυλατιονσανδ οπερατιοναλπροχεδυρεσ

COCOMO 2 levels

- COCOMO 2 is a 3 level model that allows increasingly detailed estimates to be prepared as development progresses
- Early prototyping level
  - Estimates based on object points and a simple formula is used for effort estimation
- Early design level
  - Estimates based on function points that are then translated to LOC
- Post-architecture level
  - Estimates based on lines of source code

52. What is object point estimation?

Object point estimation

- Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and 3GL modules
- They can therefore be estimated at an early point in the development process. At this stage, it is very difficult to estimate the number of lines of code in a system

53. What is productivity estimates?

Productivity estimates

- Real-time embedded systems, 40-160 LOC/P-month
- Systems programs , 150-400 LOC/P-month
- Commercial applications, 200-800 LOC/P-month
- In object points, productivity has been measured between 4 and 50 object points/month depending on tool support and developer capability

Factors affecting productivity

Factor	Description
Application domain experience	Knowledge of the application domain is essential for effective software development. Engineers whoalready understand a domain are likely to be the most productive.
Process quality	The development process used can have a significant effect on productivity. This is covered in Chapter 31.
Project size	The larger a project, the more time required for team communications. Less time is available for development so individual productivity is reduced.
Technology support	Good support technology such as CASE tools, supportive configuration management systems, etc. can improve productivity.
Working environment	As discussed in Chapter 28, a quiet working environment with private work areas contributes to improved productivity.

54. Write a note on web service.

Web services

- A web service is an instance of a more general notion of a service:

  "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production".

- The essence of a service, therefore, is that the provision of the service is independent of the application using the service.
- Service providers can develop specialized services and offer these to a range of service users from different organizations.

55. Describe service oriented approach. 56. Write advantages of service oriented approach.

Benefits of service-oriented apprach

- Services can be offered by any service provider inside or outside of an organisation so organizations can create applications by integrating services from a range of providers.
- The service provider makes information about the service public so that any authorised user can use the service.
- Applications can delay the binding of services until they are deployed or until execution. This means that applications can be reactive and adapt their operation to cope with changes to their execution environment.

Benefits of a service-oriented approach

- Opportunistic construction of new services is possible. A service provider may recognise new services that can be created by linking existing services in innovative ways.
- Service users can pay for services according to their use rather than their provision. Instead of buying a rarely-used component, the application developers can use an external service that will be paid for only when required.
- Applications can be made smaller, which is particularly important for mobile devices with limited processing and memory capabilities. Computationally-intensive processing can be offloaded to external services.

57. What is service oriented architecture?

Service-oriented architectures

- A means of developing distributed systems where the components are stand-alone services
- Services may execute on different computers from different service providers
- Standard protocols have been developed to support service communication and information exchange

Service-oriented architecture

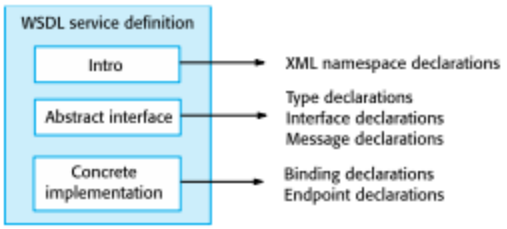
58. What are benefits of service oriented architecture?

Benefits of SOA

- Services can be provided locally or outsourced to external providers
- Services are language-independent
- Investment in legacy systems can be preserved
- Inter-organisational computing is facilitated through simplified information exchange

59. Write a note on Web service description language.

Organization of a WSDL specification



WSDL specification components

- The 'what' part of a WSDL document, called an interface, specifies what operations the service supports, and defines the format of the messages that are sent and received by the service.
- The 'how' part of a WSDL document, called a binding, maps the abstract interface to a concrete set of protocols. The binding specifies the technical details of how to communicate with a Web service.
- The 'where' part of a WSDL document describes the location of a specific Web service implementation (its endpoint).

Part of a WSDL description for a web service

Define some of the types used. Assume that the namespace prefixes 'ws' refers to the namespace URI for XML schemas and the namespace prefix associated with this definition is weathns.

```
<types>
  <xs: schema targetNameSpace = "http://.../weathns"
    xmlns: weathns = "http://.../weathns" >
    <xs:element name = "PlaceAndDate" type = "pdrec" />
    <xs:element name = "MaxMinTemp" type = "mmtrec" />
    <xs: element name = "InDataFault" type = "errmess" />

    <xs: complexType name = "pdrec"
    <xs: sequence>
    <xs:element name = "town" type = "xs:string"/>
    <xs:element name = "country" type = "xs:string"/>
    <xs:element name = "day" type = "xs:date" />
    </xs:complexType>
    Definitions of MaxMinType and InDataFault here
  </schema>
</types>
```


Part of a WSDL description for a web service

Now define the interface and its operations. In this case, there is only a single operation to return maximum and minimum temperatures.

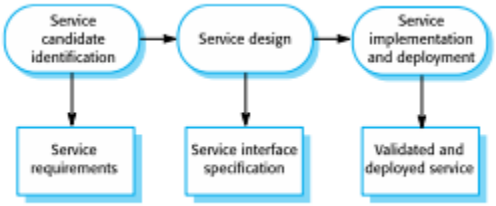
```
<interface name = "weatherInfo" >
  <operation name = "getMaxMinTemps" pattern = "wsdlns: in-out">
    <input messageLabel = "In" element = "weathns: PlaceAndDate" />
    <output messageLabel = "Out" element = "weathns:MaxMinTemp" />
    <outfault messageLabel = "Out" element = "weathns:InDataFault" />
  </operation>
</interface>
```

60. What is service engineering?

Service engineering

- The process of developing services for reuse in service-oriented applications
- The service has to be designed as a reusable abstraction that can be used in different systems.
- Generally useful functionality associated with that abstraction must be designed and the service must be robust and reliable.
- The service must be documented so that it can be discovered and understood by potential users.

61. What is service engineering process?

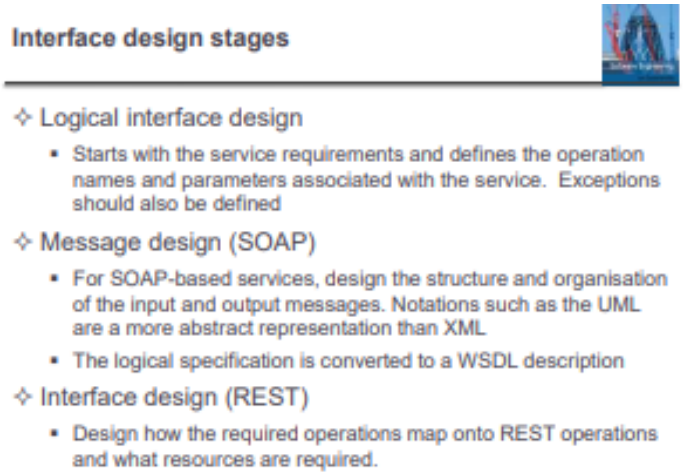


62. What are stages of service engineering?

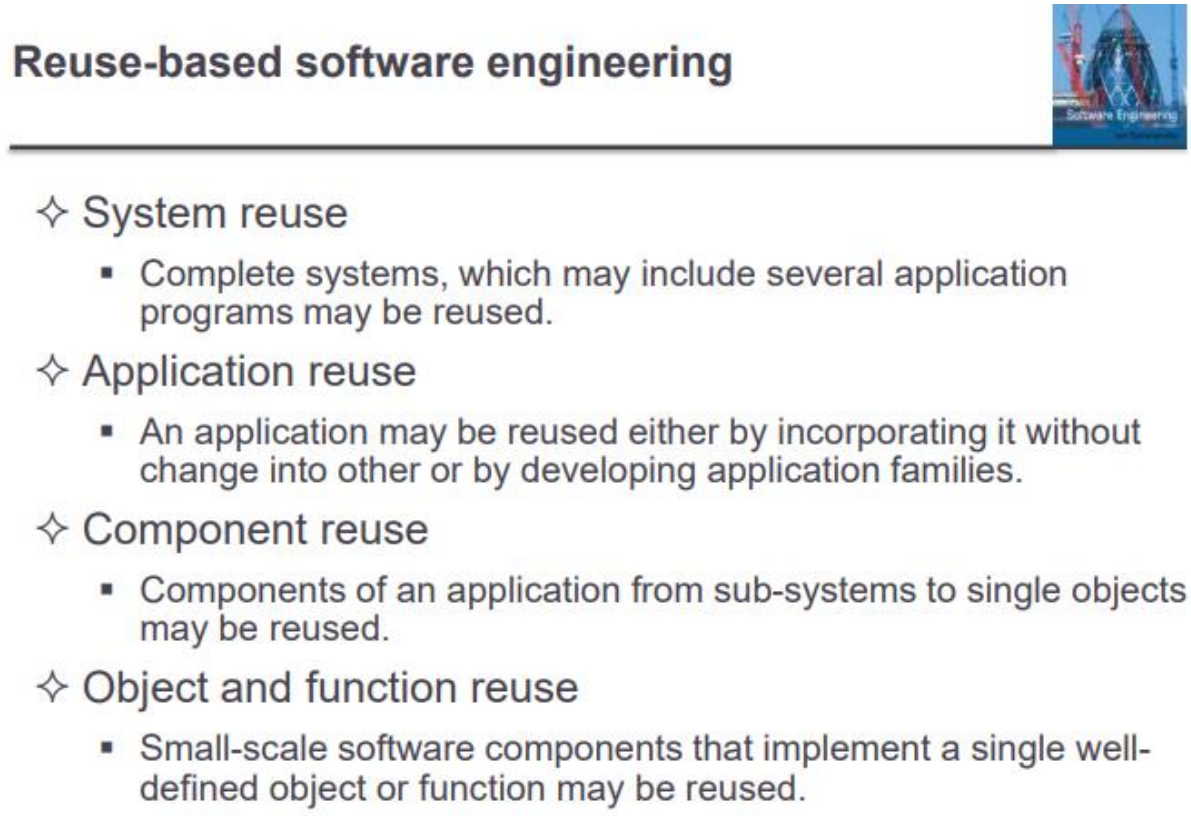
Stages of service engineering

- Service candidate identification, where you identify possible services that might be implemented and define the service requirements.
- Service design, where you design the logical service interface and its implementation interfaces (SOAP and/or RESTful)
- Service implementation and deployment, where you implement and test the service and make it available for use.

63. List and explain different interface design and stages



64. What is re-use based software engineering?



65. What are benefits of software re-use?

Benefit	Explanation
Accelerated development	Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.
Effective use of specialists	Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.
Increased dependability	Reused software, which has been tried and tested in working systems, should be more dependable than new software. Its design and implementation faults should have been found and fixed.

Benefits of software reuse

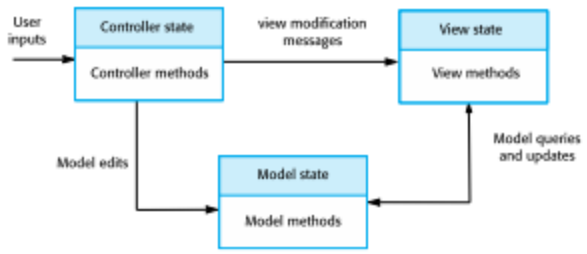
Benefit	Explanation
Lower development costs	Development costs are proportional to the size of the software being developed. Reusing software means that fewer lines of code have to be written.
Reduced process risk	The cost of existing software is already known, whereas the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as subsystems are reused.
Standards compliance	Some standards, such as user interface standards, can be implemented as a set of reusable components. For example, if menus in a user interface are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.

66. Explain model-view controller.

Model-view controller

- System infrastructure framework for GUI design.
- Allows for multiple presentations of an object and separate interactions with these presentations.
- MVC framework involves the instantiation of a number of patterns (as discussed in Chapter 7).

The Model-View-Controller pattern

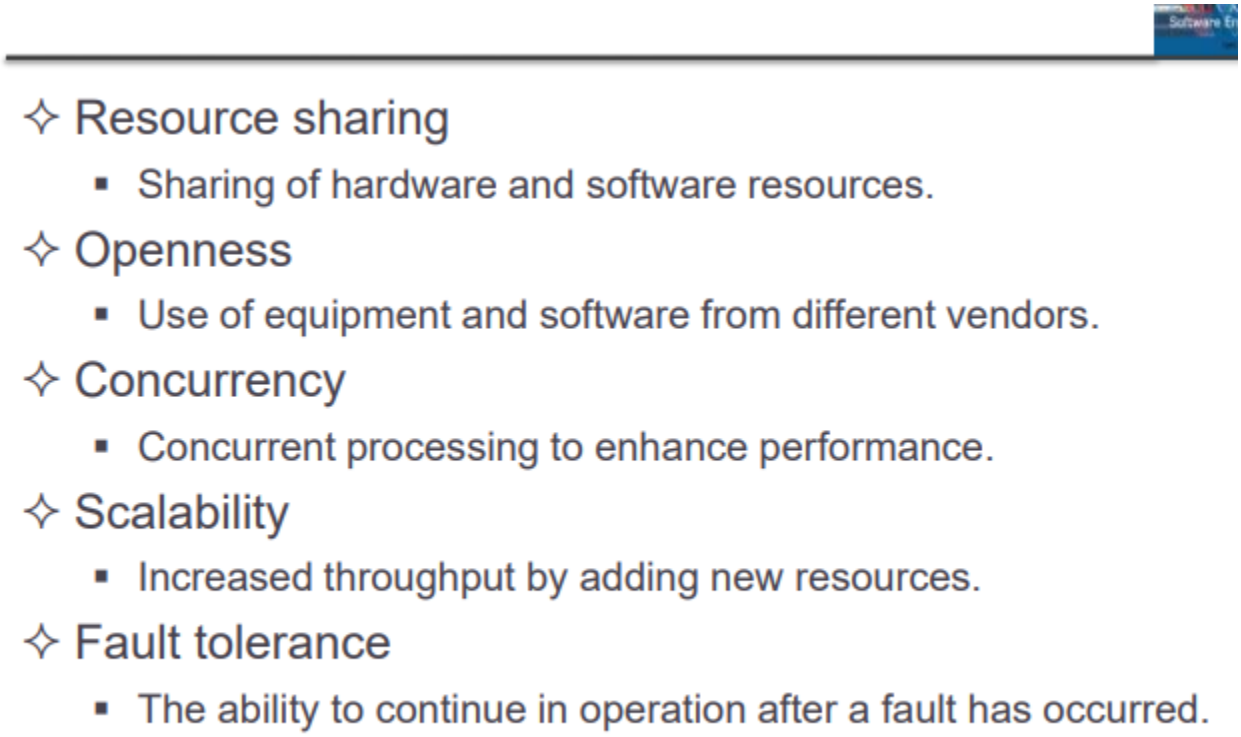


67. What is distributed system?

Distributed systems

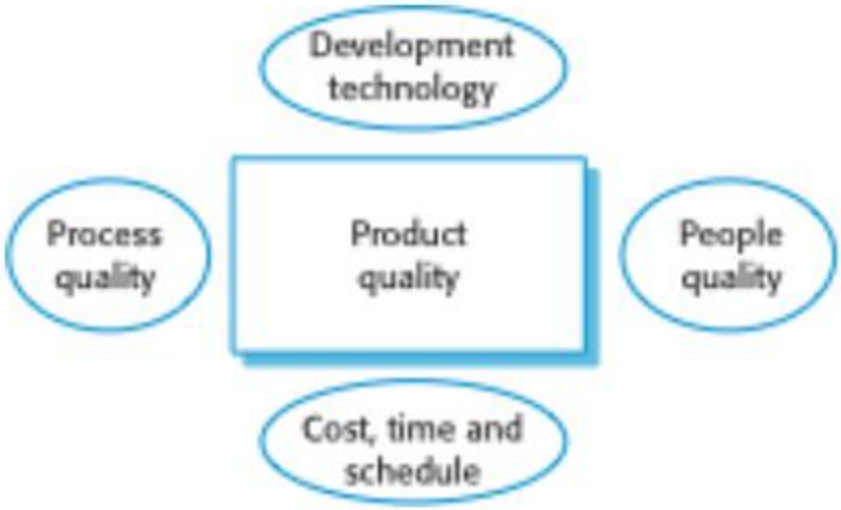
- Virtually all large computer-based systems are now distributed systems.
  - "… a collection of independent computers that appears to the user as a single coherent system."
- Information processing is distributed over several computers rather than confined to a single machine.
- Distributed software engineering is therefore very important for enterprise computing systems.

68. What are characteristics / benefits of distributed system?



69. Different factors affecting software product quality.

Factors affecting software product quality



70. Write a note on CMMI model

The CMMI process improvement framework

- The CMMI framework is the current stage of work on process assessment and improvement that started at the Software Engineering Institute in the 1980s.
- The SEI's mission is to promote software technology transfer particularly to US defence contractors.
- It has had a profound influence on process improvement
  - Capability Maturity Model introduced in the early 1990s.
  - Revised maturity framework (CMMI) introduced in 2001.

The CMMI model

- An integrated capability model that includes software and systems engineering capability assessment.
- The model has two instantiations
  - Staged where the model is expressed in terms of capability levels;
  - Continuous where a capability rating is computed.

CMMI model components

- Process areas
  - 24 process areas that are relevant to process capability and improvement are identified. These are organised into 4 groups.
- Goals
  - Goals are descriptions of desirable organisational states. Each process area has associated goals.
- Practices
  - Practices are ways of achieving a goal - however, they are advisory and other approaches to achieve the goal may be used.

Process areas in the CMMI

Category	Process area
Process management	Organizational process definition (OPD)
	Organizational process focus (OPF)
	Organizational training (OT)
	Organizational process performance (OPP)
	Organizational innovation and deployment (OID)
Project management	Project planning (PP)
	Project monitoring and control (PMC)
	Supplier agreement management (SAM)
	Integrated project management (IPM)
	Risk management (RSKM)
	Quantitative project management (QPM)

Process areas in the CMMI

Category	Process area
Engineering	Requirements management (REQM)
	Requirements development (RD)
	Technical solution (TS)
	Product integration (PI)
	Verification (VER)
	Validation (VAL)
Support	Configuration management (CM)
	Process and product quality management (PPQA)

CMMI assessment

- Examines the processes used in an organisation and assesses their maturity in each process area.
- Based on a 6-point scale:
  - Not performed;
  - Performed;
  - Managed;
  - Defined;
  - Quantitatively managed;
  - Optimizing.

The staged CMMI model

- Comparable with the software CMM.
- Each maturity level has process areas and goals. For example, the process area associated with the managed level include:
  - Requirements management;
  - Project planning;
  - Project monitoring and control;
  - Supplier agreement management;
  - Measurement and analysis;
  - Process and product quality assurance.

The CMMI staged maturity model

Key points

- The CMMI process maturity model is an integrated process improvement model that supports both staged and continuous process improvement.
- Process improvement in the CMMI model is based on reaching a set of goals related to good software engineering practice and describing, standardizing and controlling the practices used to achieve these goals.
- The CMMI model includes recommended practices that may be used, but these are not obligatory.