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.

1. 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.

1. 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

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

Portability

Interoperability

Reusability

Adaptability

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

Modularity

Maintainability

Flexibility

Scalability

1. 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.

1. 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.

1. 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.

1. 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.

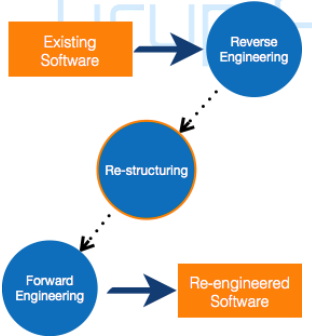
• Adaptive Maintenance - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.

• 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.

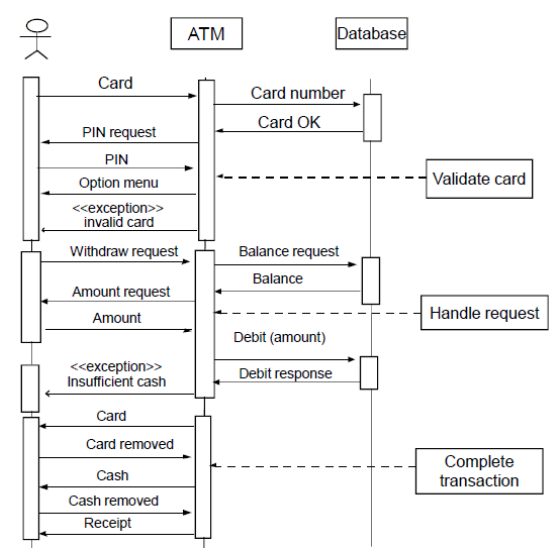
Types of requirement

• 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.

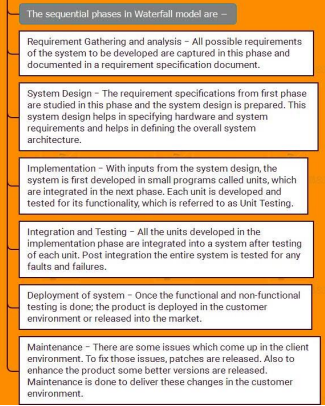
2. Roles: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.

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.

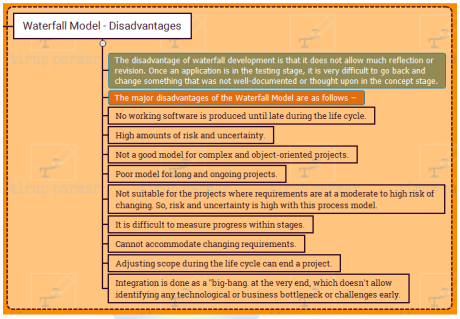
Software process is complex, it relies on making decisions. There’s no ideal process and most organizations have developed their own software process

15. What is waterfall model?









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

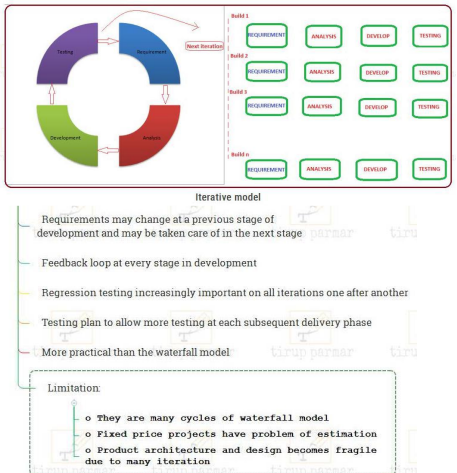
1. Establish objectives: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.

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.

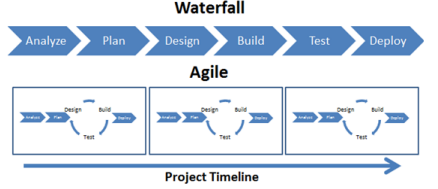
4. Evaluate the prototype: Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed. Prototyping is not a standalone, complete 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?



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.

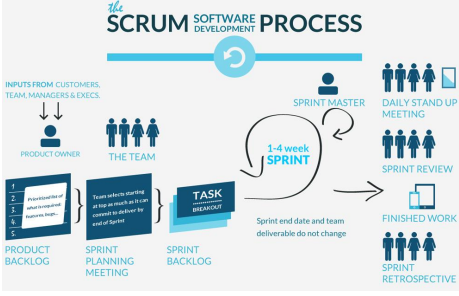
♣ New versions may be built several times per day;

♣ 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.



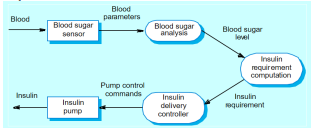
A scrum is an action associated with the game of rugby. It is enacted when players huddle together with the objective of moving the ball strategically towards the goal post. The SCRUM agile development methodology is derived from this action and draws from the rugby scrum some principles which are embedded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



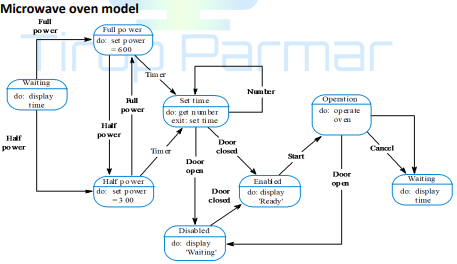
19. Explain data flow diagram.

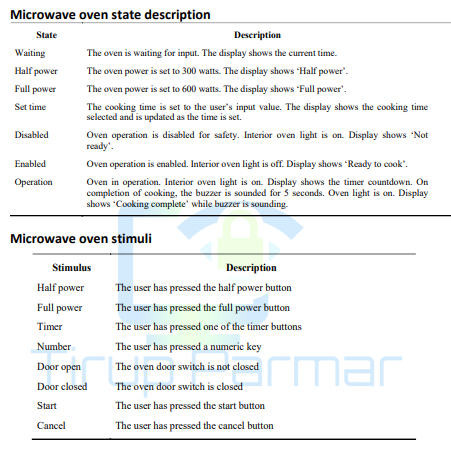
Data flow diagrams ¬ DFDs model the system from a functional perspective. ¬ Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system. ¬ Data flow diagrams may also be used in showing the data exchange between a system and other systems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

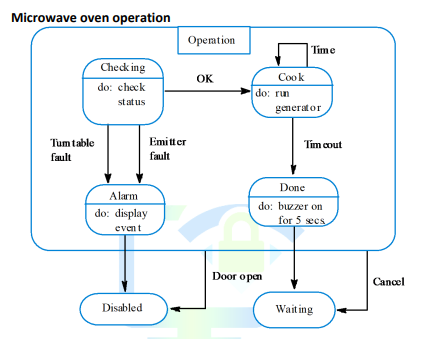
Insulin pump DFD



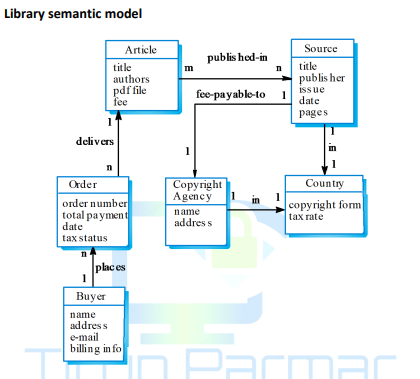
20. Draw and explain microwave oven model.

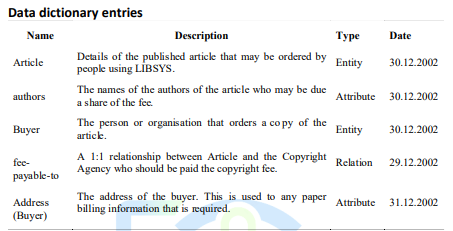






21. Draw and explain library semantic model.



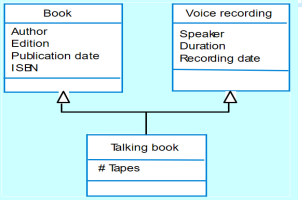


22. What is multiple inheritance model?

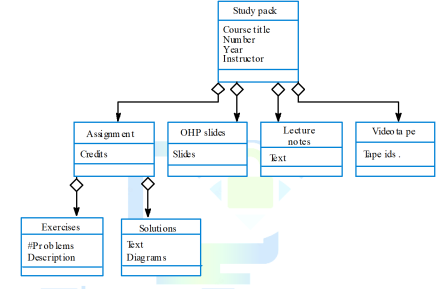
¬ Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several superclasses.

¬ 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



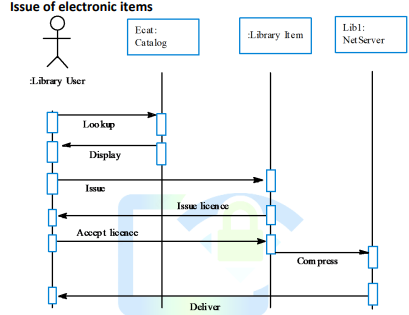
¬ An aggregation model shows how classes that are collections are composed of other classes. ¬ Aggregation models are similar to the part-of relationship in semantic data models.



23. Explain Object behavior modeling.

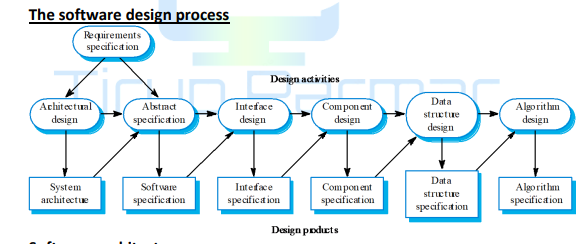
¬ A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case.

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

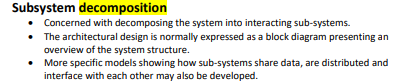


¬ Elicit requirements • Identify viewpoints • Conduct interviews and ethnographies • Draw up scenarios and use cases ¬ Refine use cases with functional decomposition and identify objects involved in interactions ¬ Build object model ¬ Create sequence diagrams for each use case ¬ Create state diagrams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

24. Write a note on software design process.



25. What is sub system decomposition?



26. What is repository model?

The repository model

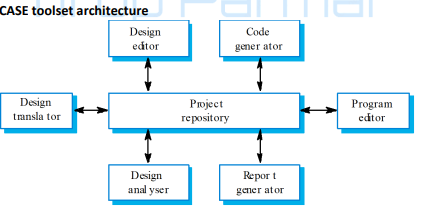
• Sub-systems must exchange data.

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.



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.

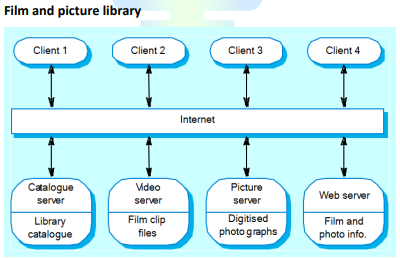
27. What is client server model?

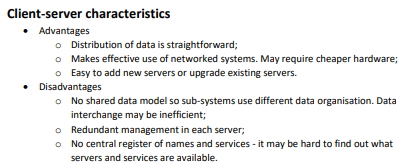
Client-server model

• 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.





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 subsystems or the system’s environment.

**Centralised control**

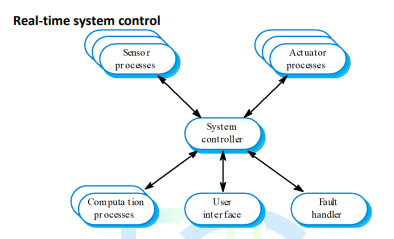
• A control sub-system takes responsibility for managing the execution of other subsystems.

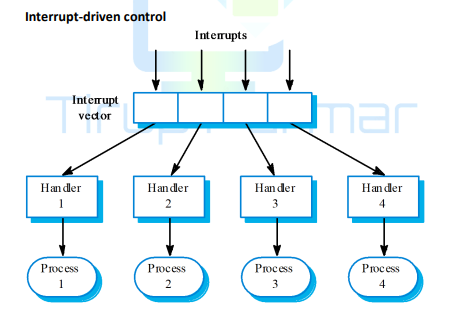
**• Call-return model**

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

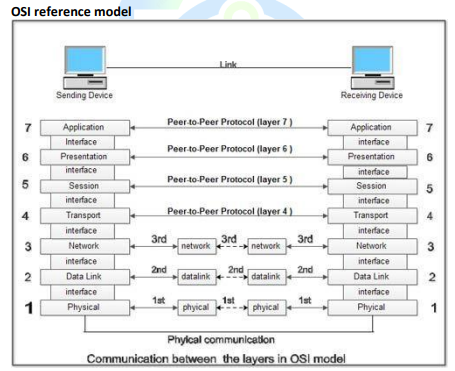
**• Manager model**

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.

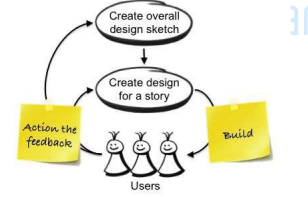


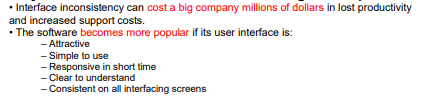


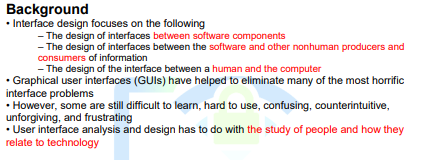
30. Explain OSI reference model.

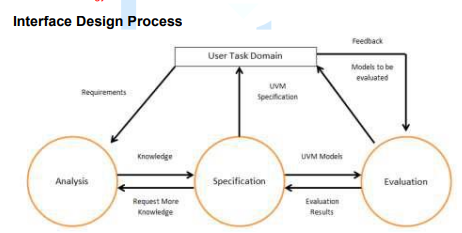


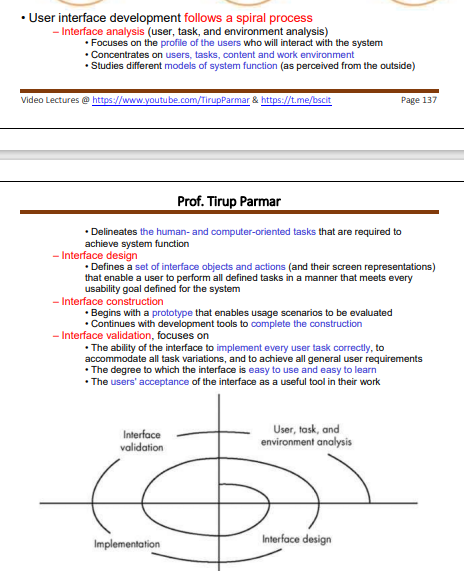
31. What is user interface design process? What are 3 golden rules of user interface design?

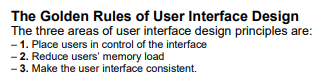


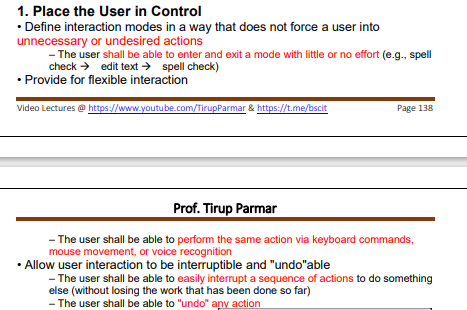


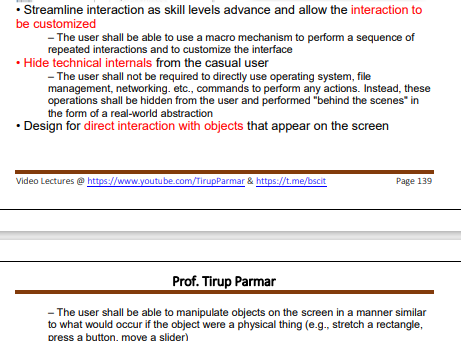


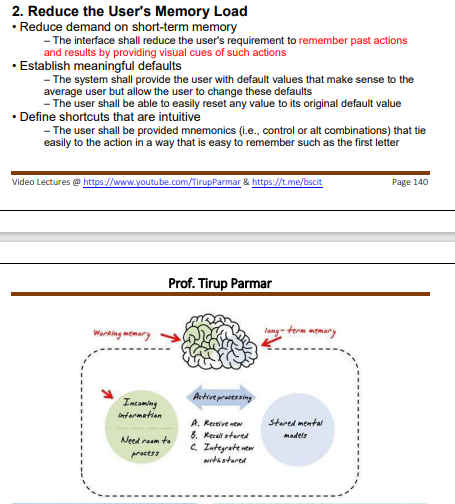


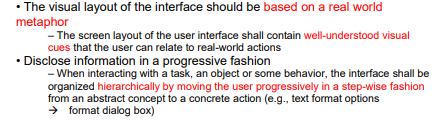


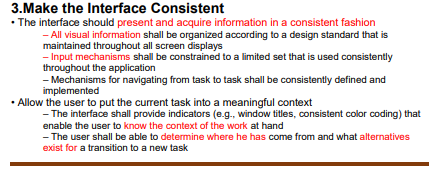


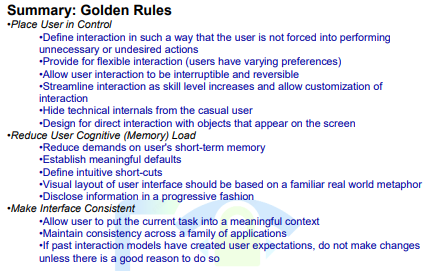


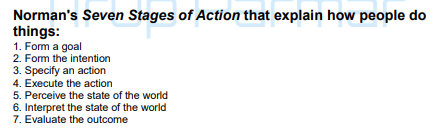




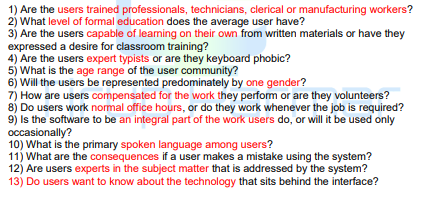






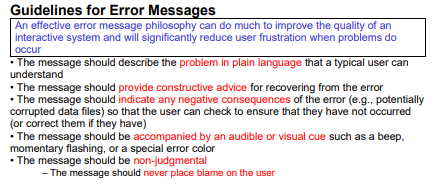


32. What questions should be answered during user analysis process?

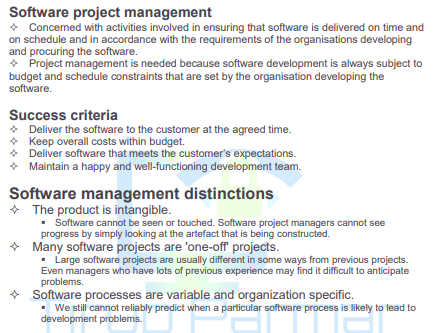


33. What is error message?

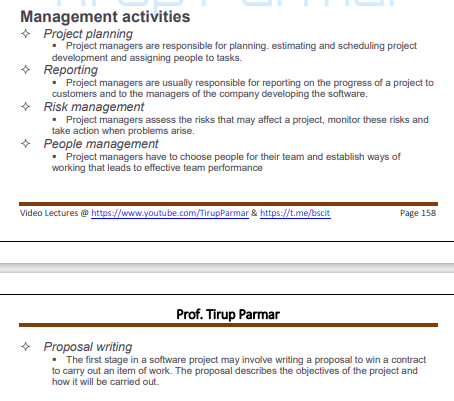
What are guidelines for error messages?



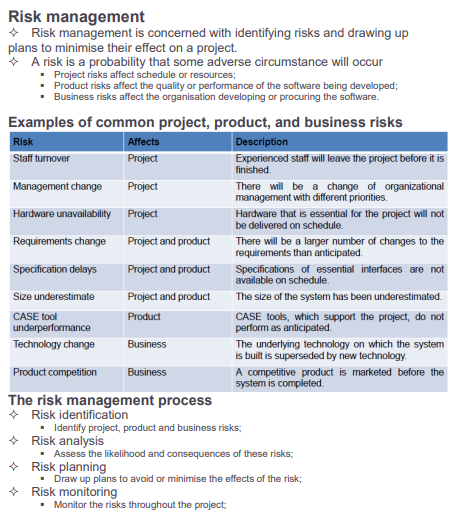
34. What is software project management?

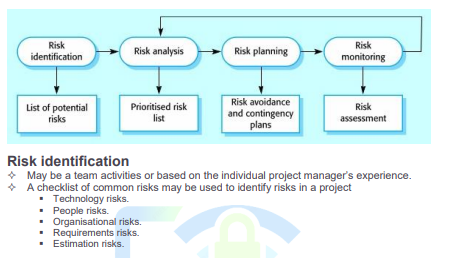


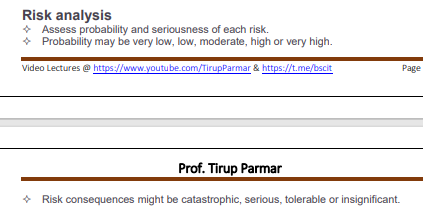
35. What are different management activities?

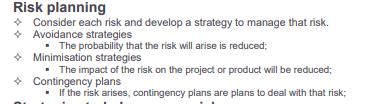


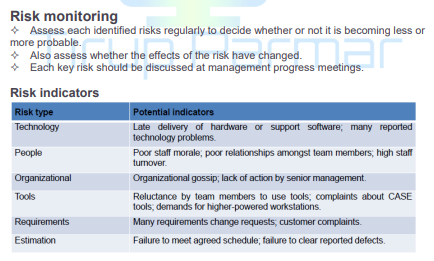
36. Explain risk management.



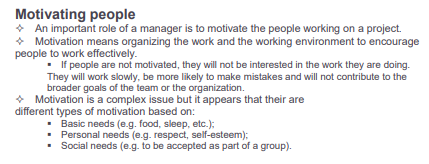


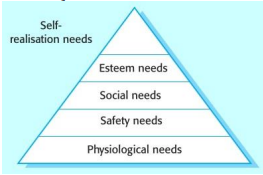


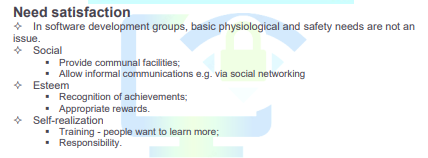




37. How to motivate people in an organization?

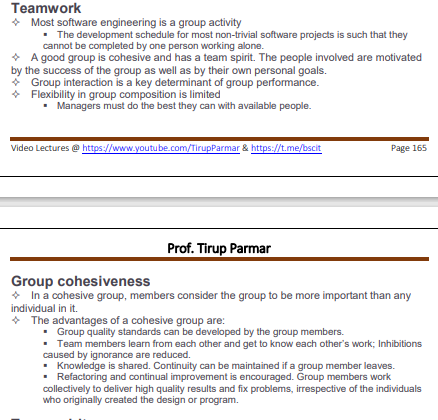




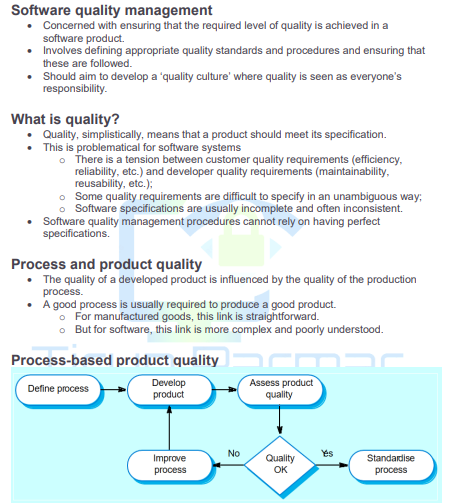


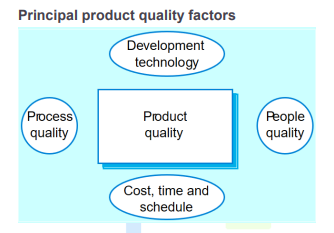


38. How team spirit is important?

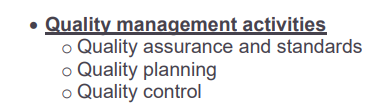


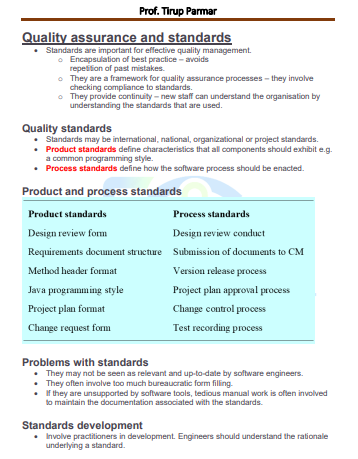
39. What is software quality? What are quality factors to be considered?



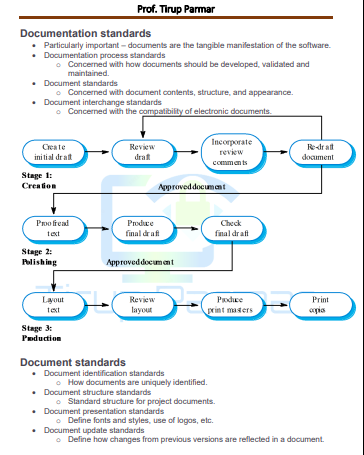


40. What are different quality management activities? Explain any one.





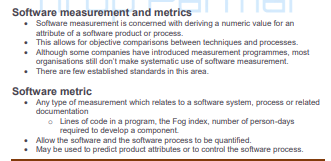


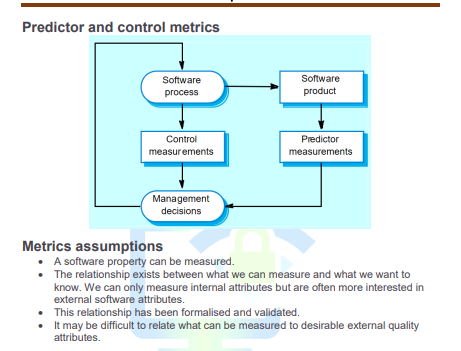


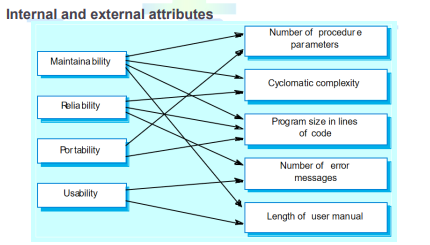


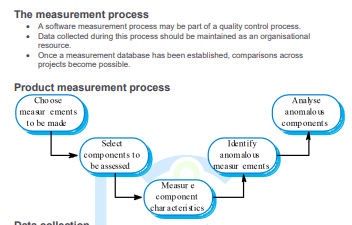


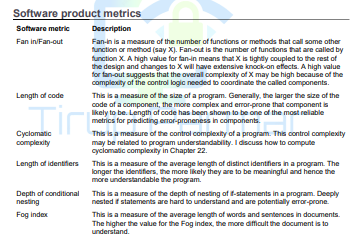
41. Write a note on Software measurement and metrics.



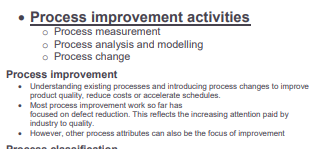


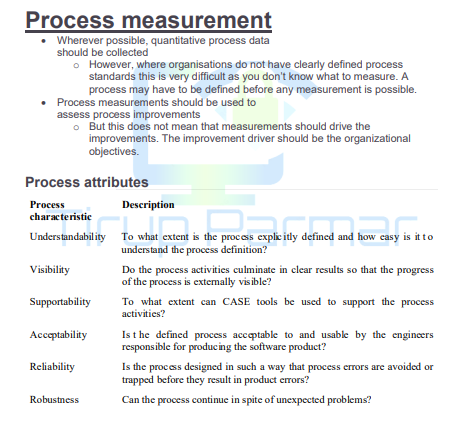




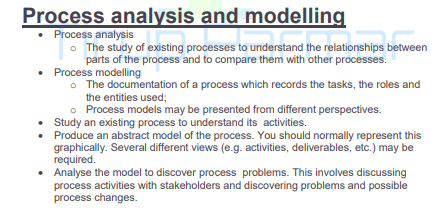


42. Write a note on process improvement activities.

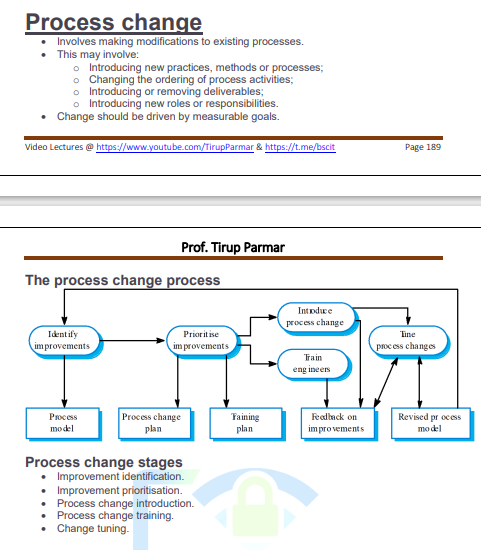




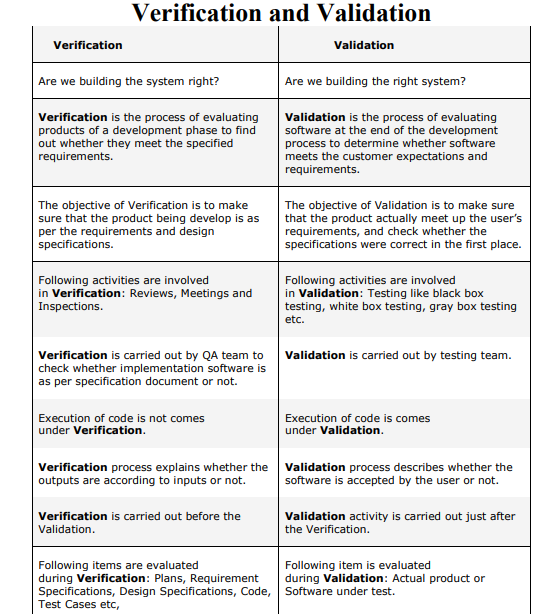
43. What is process analysis?



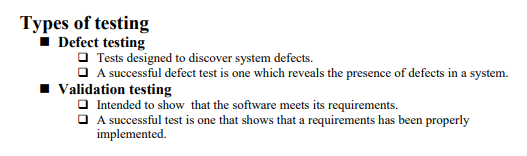
44. What is process change process?

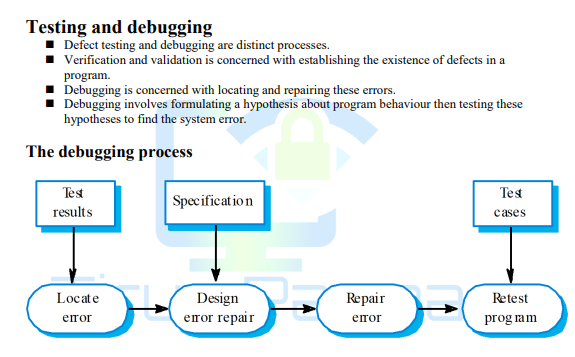


45. Differentiate between verification and validation.

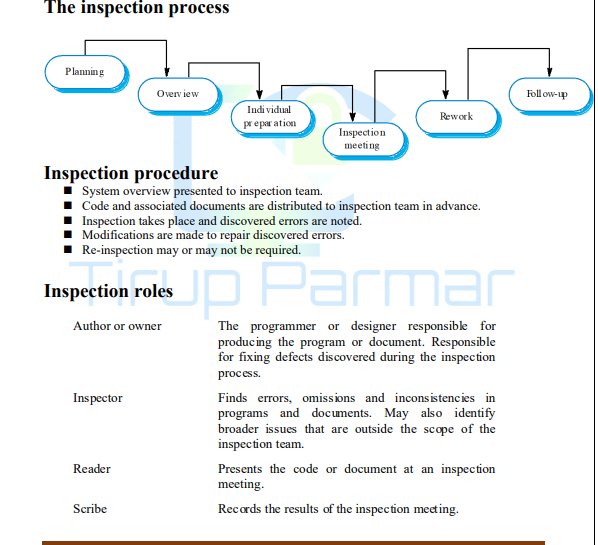


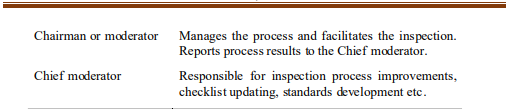
46. Write a note on testing and debugging.



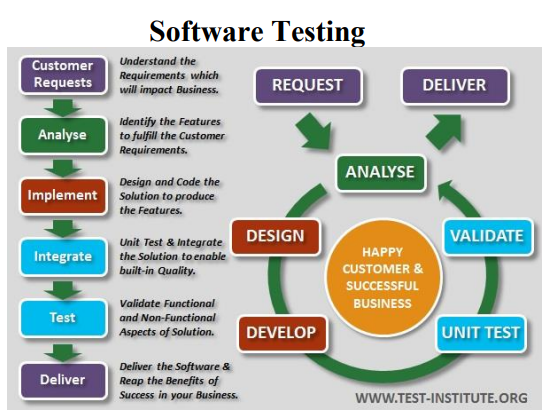


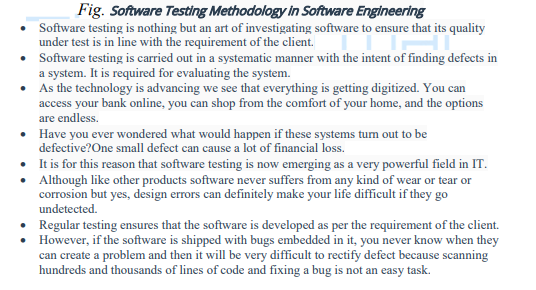
47. What do you mean by inspection process, procedure and roles.



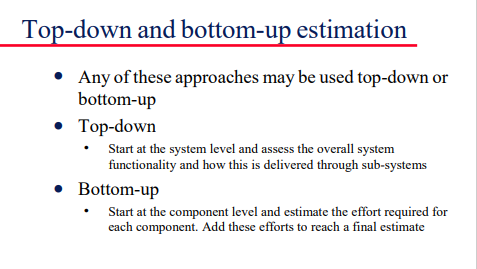


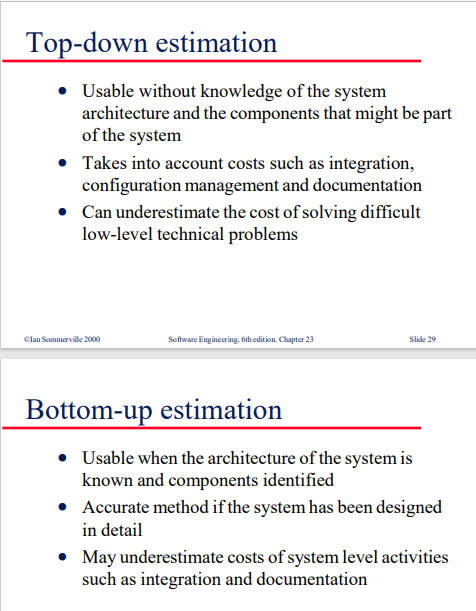
48. Explain software testing life cycle.



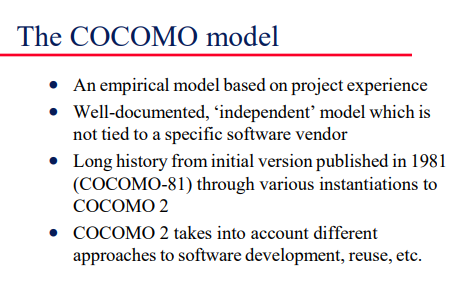


49. What is top-down estimation? 50. What is bottom-up estimation?

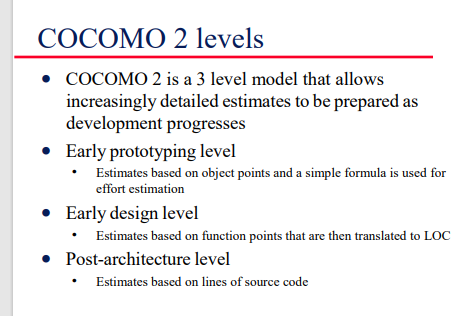




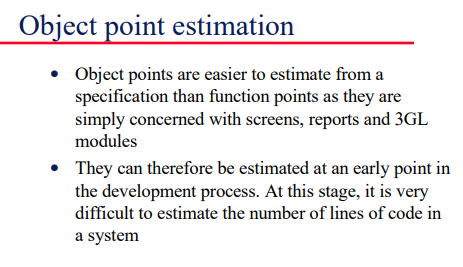
51. Explain cocomo model.



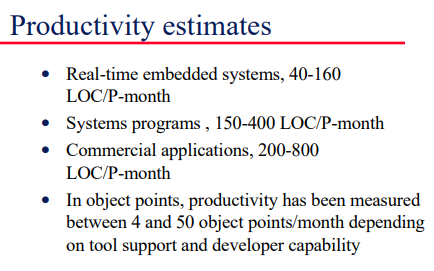


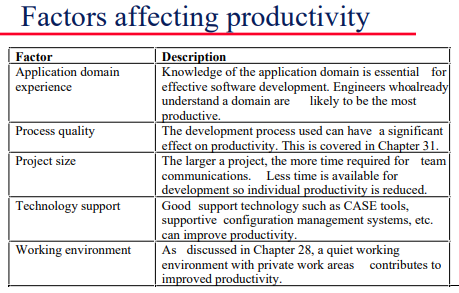


52. What is object point estimation?

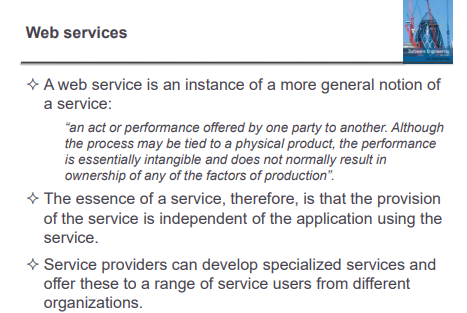


53. What is productivity estimates?

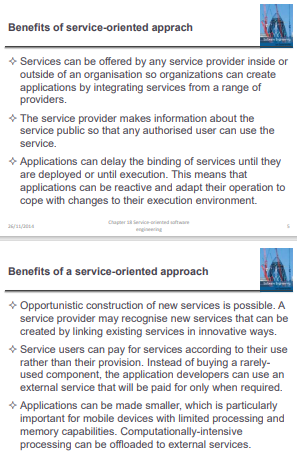




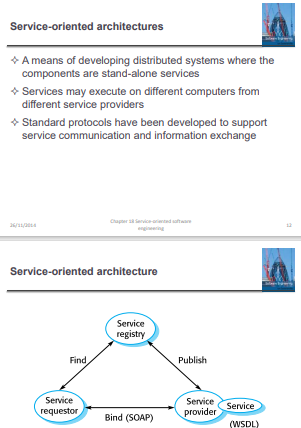
54. Write a note on web service.



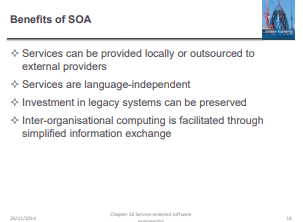
55. Describe service oriented approach. 56. Write advantages of service oriented approach.



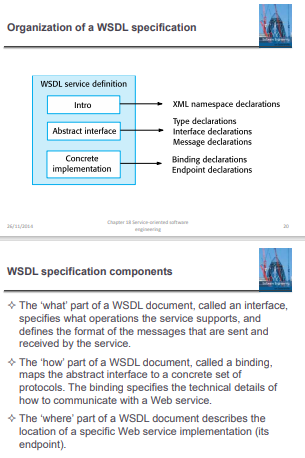
57. What is service oriented architecture?

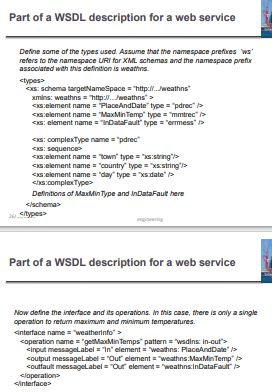


58. What are benefits of service oriented architecture?

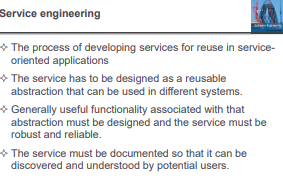


59. Write a note on Web service description language.

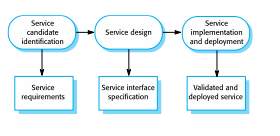




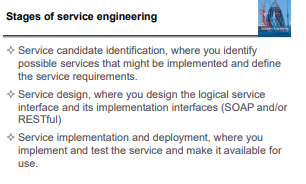
60. What is service engineering?



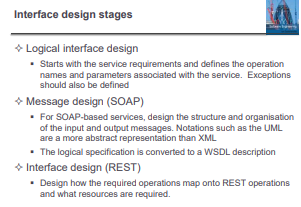
61. What is service engineering process?



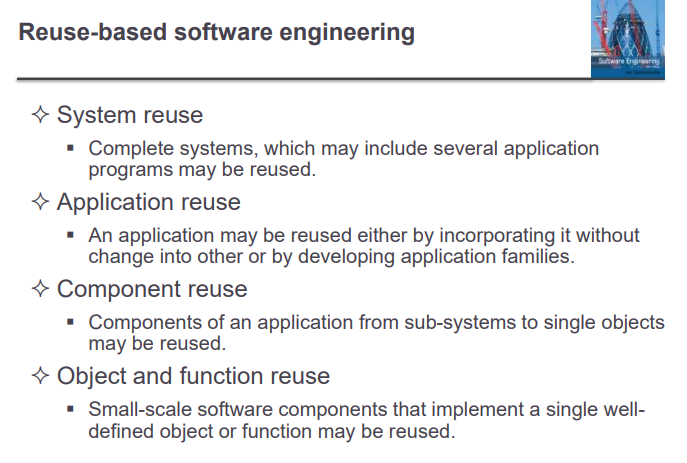
62. What are stages of service engineering?



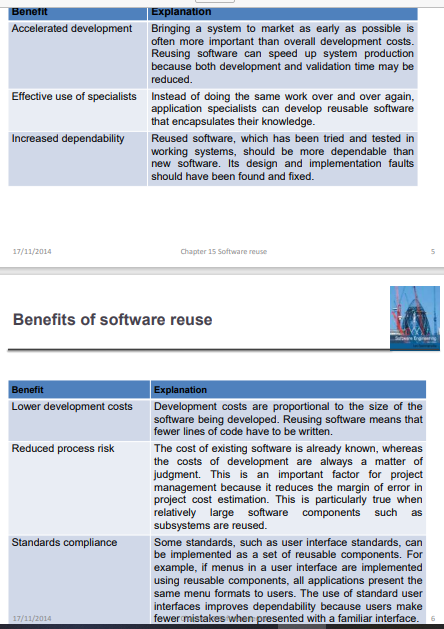
63. List and explain different interface design and stages



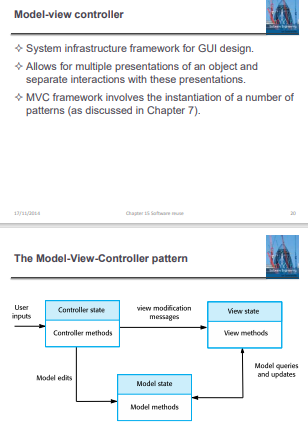
64. What is re-use based software engineering?



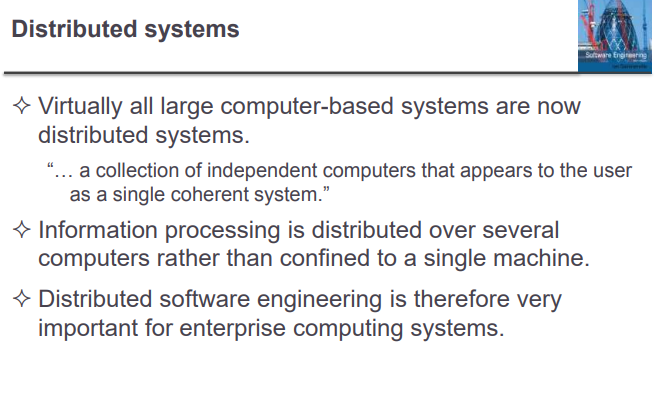
65. What are benefits of software re-use?



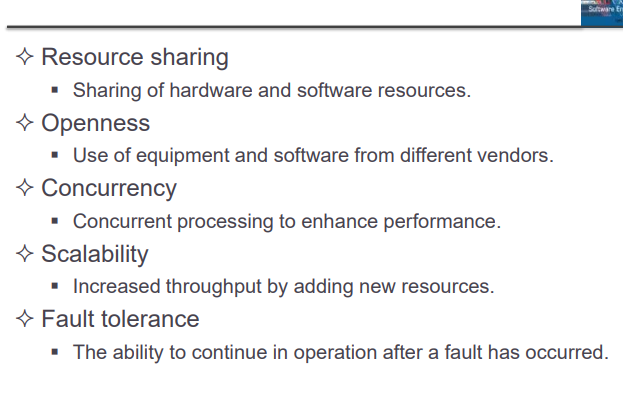
66. Explain model-view controller.



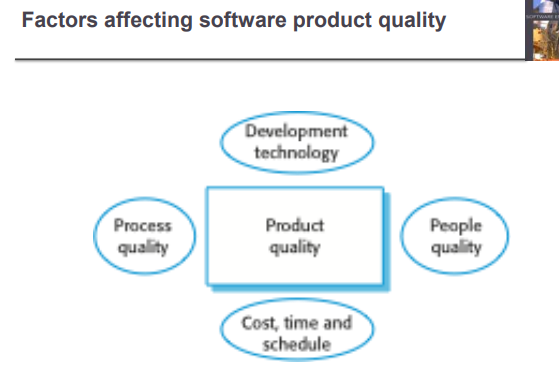
67. What is distributed system?



68. What are characteristics / benefits of distributed system?



69. Different factors affecting software product quality.



70. Write a note on CMMI model

