Code No. 5154 / S

FACULTY OF ENGINEERING

B.E. 3/4 (CSE) I - Semester (Supplementary) Examination, June 2016

Subject : Software Engineering

Time : 3 hours **Max. Marks : 75**

Note: Answer all questions from Part-A. Answer any FIVE questions from Part-B.

PART – A (25 Marks)

1	Differentiate between personal and team process models.	3
2	What is an Agile process?	2
3	What is Effort? How effort is estimated?	2
4	What is “Collaborative requirements gathering”? Enlist the guidelines for it.	3
5	Differentiate between cardinality and modality.	3
6	What do you understand by the term “design quality”?	2
7	List and explain in brief the golden rules of user-interface design.	3
8	What is transform mapping?	2
9	What is equivalence class partitioning?	3
10	What is software verification? How is it different from validation?	2

PART – B (50 Marks)

11	a)	What is process framework? Explain.	5
	b)	Explain unified process model.	5
12	a)	What is Risk? Explain how risk is managed.	5
	b)	What are requirements Engineering tasks? Explain validating requirements.	5
13	a)	How to create a behavioural model? Explain about the state representations.	5
	b)	Explain design concepts.	5
14		Explain architectural styles and patterns in detail.	10
15		What are different levels of testing? Explain the usefulness of each level.	10

16 a) Explain about software project planning. 5

b) What is a metric? Explain the metrics for source code. 5

17 Write short notes on :

a) Evolutionary process models 4

b) Design evaluation 3

c) Debugging 3

******

1. Differentiate between personal and team process models

Ans. Personal software process: -

- It is a systematic application development method intended to help engineers understand and enhance their output by applying professionalism to the way they build software and monitoring their expected and actual code creation.

- It indicates developers how to control the value of their assets, what to draw up a sound plan as well as how to make promises. This also provides them the evidence to explain their proposals.

- The personal software method focuses on entities to enhance their results. It consists of methods, types and techniques that orient programmers in their technical work.

Team software process: -

- The goal of the TSP is to enhance the quality and efficiency of the entire team application development project, in addition to helping us help meet the expense and timeline obligations of creating software.

- TSP is designed for use in education environments, concentrating on the process of creating a project management team, creating team goals, assigning team tasks, and several other collaboration activities.

- Team software relies on a community of people and seeks to improve team performance.

2 What is an Agile process?

Ans. "**Agile process** model" refers to a **software development** approach based on iterative **development**. **Agile** methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the **development process**.

3. What is Effort? How effort is estimated?

Ans. In software development, effort estimation is the process of predicting the most realistic amount of effort (expressed in terms of person-hours or money) required to develop or maintain software based on incomplete, uncertain and noisy input. Effort estimates may be used as input to project plans, iteration plans, budgets, investment analyses, pricing processes and bidding rounds.

There are many ways of categorizing estimation approaches, see for example.[12][13] The top level categories are the following:

- Expert estimation: The quantification step, i.e., the step where the estimate is produced based on judgmental processes.
- Formal estimation model: The quantification step is based on mechanical processes, e.g., the use of a formula derived from historical data.
- Combination-based estimation: The quantification step is based on a judgmental and mechanical combination of estimates from different sources.

4. What is "Collaborative requirements gathering"? Enlist the guidelines for it.

Ans. Collaborative requirements gathering

Fix the **rules** for preparation and participation. The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

5. Differentiate between cardinality and modality.

Cardinality

- It tells about the maximum number of associations between rows of tables.
- There are different types: One-to-one, one-to-many, many-to-many.
- One to one is where the occurrence of object 'A' can relate to one and only one occurrence of object 'B', and vice-versa.
- One to many is where the occurrence of object 'A' can relate to multiple occurrences of object 'B', but object 'B' can relate to a single occurrence of object 'A'.
- Many to many is where multiple occurrences of object 'A' can relate to multiple occurrences of object 'B', and vice-versa.

Modality

- It tells about the minimum number of row associations in a table.
- There are different types: Nullable and Not nullable.
- Nullable columns accept an empty field.
- The non-nullable column doesn't accept null values.

6. What do you understand by the term "design quality"?

Ans. Design quality is the value of a design to customers. Design is the root of all quality including the quality of products, services, experiences, systems and processes. For example, a product with a poor design will be low quality even if quality control and quality assurance succeed in producing the design accurately.

7. List and explain in brief the golden rules of user-interface design.

Ans. Golden Rules of User Interface Design

1. Place Users in Control
2. Reduce Users' Memory Load
3. Make the Interface Consistent

Place Users in Control

1. Use modes judiciously (modeless)
2. Allow users to use either the keyboard or mouse (flexible)
3. Allow users to change focus (interruptible)
4. Display descriptive messages and text(Helpful)
5. Provide immediate and reversible actions, and feedback (forgiving)

Reduce Users' Memory Load

1. Relieve short-term memory (remember)
2. Rely on recognition, not recall (recognition)
3. Provide visual cues (inform)
4. Provide defaults, undo, and redo (forgiving)
5. Provide interface shortcuts (frequency)

Make the Interface Consistent

1. Sustain the context of users' tasks (continuity)
2. Maintain consistency within and across products (experience)
3. Keep interaction results the same (expectations)
4. Provide aesthetic appeal and integrity (attitude)
5. Encourage exploration (predictable)

8. What is transform mapping?

Ans. Transform mapping is a technique in which data flow diagrams (DFDs) are mapped to a specific scenario. In other words, it's a data flow-oriented mapping technique that uses DFDs to map real life scenarios to a software architecture.

9 What is equivalence class partitioning?

Ans. Equivalence partitioning or equivalence class partitioning (ECP) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once.

10. What is software verification? How is it different from validation?

Ans. Software verification is a discipline of software engineering whose goal is to assure that software fully satisfies all the expected requirements.

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements.

Verification	**Validation**
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

11 a) What is process framework? Explain.

b) Explain unified process model.

Ans. Process Framework:

- A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of size or complexity.
- It also includes a set of umbrella activities that are applicable across the entire software process.
- Each framework activity is populated by a set of software engineering actions – a collection of related tasks that produces a major software engineering work product (e.g. design is a software engineering action).
- Each action is populated with individual work tasks that accomplish some part of the work implied by the action.
- The following generic process framework is applicable to the vast majority of software projects :

  1. Communication: This framework activity involves heavy communication and collaboration with the customer (and other stakeholders) and encompasses requirements gathering and other related activities.

  2. Planning: This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced and a work schedule.

  3. Modeling: This activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.

  4. Construction: This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

  5. Deployment: The software is delivered to the customer who evaluates the delivered product and provides feedback based on evaluation.

- Some most applicable framework activities are described below.

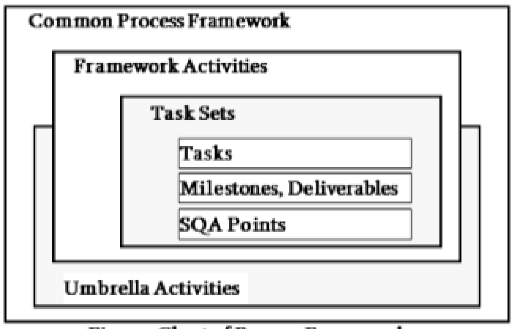
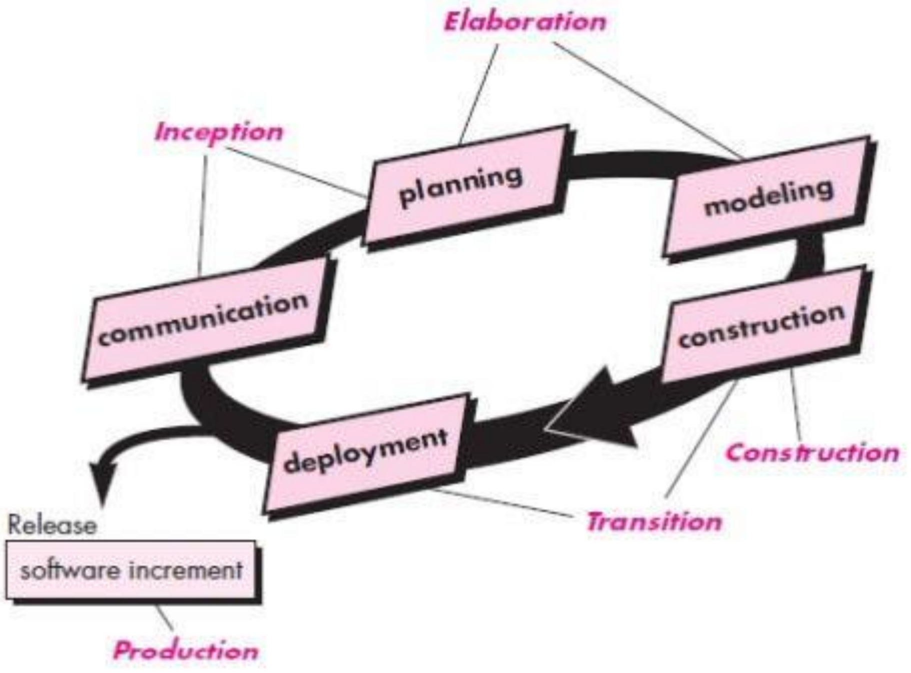


Figure: Chart of Process Framework

B. Explain unified process model.

The Unified Process in Software Engineering



Unified process (UP) is an architecture centric, use case driven, iterative and incremental development process. UP is also referred to as the unified software development process.

The Unified Process is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development. The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer’s view of a system. It emphasizes the important role of software architecture and “helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse” . It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Unified Process

This process divides the development process into five phases:

- Inception
- Elaboration
- Conception
- Transition
- Production

The inception phase of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.

The elaboration phase encompasses the communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software the use case model, the requirements model, the design model, the implementation model, and the deployment model. Elaboration creates an “executable architectural baseline” that represents a “first cut” executable system.

The construction phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users. To accomplish this, requirements and design models that were started during the elaboration phase are completed to reflect the final version of the software increment. All necessary and required features and functions for the software increment (the release) are then implemented in source code.

The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity. Software is given to end users for beta testing and user feedback reports both defects and necessary changes. At the conclusion of the transition phase, the software increment becomes a usable software release.

The production phase of the UP coincides with the deployment activity of the generic process. During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated. It is likely that at the same time the construction, transition, and production phases are being conducted, work may have already begun on the next software increment. This means that the five UP phases do not occur in a sequence, but rather with staggered concurrency.

12 a) What is Risk? Explain how risk is managed.

b) What are requirements Engineering tasks? Explain validating requirements.

Ans. Software risk encompasses the probability of occurrence for uncertain events and their potential for loss within an organization. Typically, software risk is viewed as a combination of robustness, performance efficiency, security and transactional risk propagated throughout the system. Risk management is the process of identifying, assessing and controlling threats to an organization's capital and earnings. Risk management allows organizations to attempt to prepare for the unexpected by minimizing risks and extra costs before they happen.

B. What are requirement engineering tasks? Explain validating requirements?

Ans. The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.

Requirement engineering constructs a bridge for design and construction.

Requirement engineering consists of seven different tasks as follow:

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Elicitation

Elicitation means to find the requirements from anybody.

The requirements are difficult because the **following problems occur in elicitation**.

Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.

Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project.

3. Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioural.
- The requirement is formalized in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.
- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.

checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation.

In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

13 a) How to create a behavioral model? Explain about the state representations.

b) Explain design concepts.

Ans. To create behavioural model following things can be considered:

- Evaluation of all use-cases to fully understand the sequence of interaction within the system.
- Identification of events that drive the interaction sequence and understand how these events relate to specific classes.
- Creating sequence for each use case.
- Building state diagram for the system.
- Reviewing the behavioural model to verify accuracy and consistency.

It describes interactions between objects. It shows how individual objects collaborate to achieve the behaviour of the system as a wholetime behaviour of a system is shown with the help of use case diagram, sequence diagram and activity diagram

- A use case focuses on the functionality of a system i.e. what a system does for users. It shows interaction between the system and outside actors': Student, librarians are actors, issue book use case.
- A sequence diagram shows the objects that interact and the time sequence of their interactions': Student, librarians are objects. Time sequence enquires for book check availability –with respect time.
- An activity diagram specifies important processing steps. It shows operations required for processing steps. It shows operations required for processing issue book, check availability does not show objects

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams.

b) Explain design concepts.

Ans. Design concepts

The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest-level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

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

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

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

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

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

14 Explain architectural styles and patterns in detail.

Architectural Style

The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design. Furthermore, when building the architectural style of our system we focus on layers and modules and how they are communicating with each other.

There are different types of architectural styles, and moreover, we can mix them and produce a hybrid style that consists of a mix between two and even more architectural styles. Below is a list of architectural styles and examples for each category:

- **Structure architectural styles:** such as layered, pipes and filters and component-based styles.
- **Messaging styles:** such as Implicit invocation, asynchronous messaging and publish-subscribe style.
- **Distributed systems:** such as service-oriented, peer to peer style, object request broker, and cloud computing styles.
- **Shared memory styles:** such as role-based, blackboard, database-centric styles.
- **Adaptive system styles:** such as microkernel style, reflection, domain-specific language styles.

Design Patterns

Design patterns are accumulative best practices and experiences that software professionals used over the years to solve the general problem by – trial and error – they faced during software development. The Gang of Four (GOF, refers to Eric Gamma, Richard Helm, Ralf Johnson, and John Vlissides) wrote a book in 1994 titled with "Design Pattern – Elements of reusable object-oriented software" in which they suggested that design patterns are based on two main principles of object-oriented design:

- Develop to an interface, not to an implementation.
- Favor object composition over inheritance.

Also, they presented that the design patterns set contains 23 patterns and categorized into three main sets:

1. Creational design patterns:
Provide a way to create objects while hiding the creation logic. Thus, the object creation is to be done without instantiating objects directly with the "New" keyword to gives the flexibility to decide which objects need to be created for a given use case. The creational design patterns are:

- **Abstract factory pattern:** provide an interface to create objects without specifying the classes.
- **Singleton pattern:** provide only a single instance of the calls and global access to this instance.
- **Builder Pattern:** Separate the construction from representation and allows the same construction to create multiple representations.
- **Prototype pattern:** creating duplicate without affecting the performance and memory. So, the duplicate object is built from the skeleton of an existing object.

2. Structural patterns:
Concerned with class and object composition. The Structural design patterns are:

- **Adapter pattern:** it works as a bridge between two incompatible interfaces and companies their capabilities.
- **Bridge pattern:** provide a way to decouple the abstraction from its implementation.
- **Filter pattern:** Also known as criteria pattern, it provides a way to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations.
- **Composite pattern:** provide a way to treat a group of objects in a similar way as a single object. It composes objects in term of a tree structure to represent part as well as a whole hierarchy
- **Decorator pattern:** allows adding new functionality to an existing object without altering its structure.
- **Façade pattern:** provide a unified interface to a set of interfaces.it hides the complexities of the system and provides an interface to the client using which the client can access the system.
- **Flyweight pattern:** reduce the number of objects created and to decrease memory footprint and increase performance. It helps in reusing already existing similar kind objects by storing them and creates a new object when no matching object is found.
- **Proxy pattern:** provides a place holder to another object to control access to it. The object has an original object to interface its functionality to the outer world.

3. Behavioural patterns:
Behavioural patterns are concerned with communications between objects. The following is the list of behavioural patterns:

- **Responsibility pattern:** creates a chain of receiver objects for a request. This pattern decouples the sender and receiver of a request based on the type of request.
- **Command pattern:** it's a data-driven pattern in which A request is wrapped under an object as command and passed to an invoker object.
- **Interpreter pattern:** provides a way to evaluate language grammar or expression. It involves implementing an expression interface that tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine, etc.
- **Iterator pattern:** provides a way to access the elements of a collection object in a sequential manner without any need to know its underlying representation.
- **Mediator pattern:** used to reduce communication complexity between multiple objects or classes. It provides a mediator class that normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling.
- **Memento pattern:** used to restore the state of an object to a previous state.
- **Observer pattern:** used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.
- **State pattern:** is used to change the class behavior based on its state.
- **Null object pattern** helps to avoid null references by having a default object.
- **Strategy pattern:** provides a way to change class behavior or its algorithm at run time.
- **Template pattern:** an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need, but the invocation is to be in the same way as defined by an abstract class.
- **Visitor pattern:** used to change the executing algorithm of an element class.

15 What are different levels of testing? Explain the usefulness of each level.

Ans. Levels of Testing

There are mainly four **Levels of Testing** in software testing:

1. **Unit Testing**: checks if software components are fulfilling functionalities or not.
2. **Integration Testing**: checks the data flow from one module to other modules.
3. **System Testing**: evaluates both functional and non-functional needs for the testing.
4. **Acceptance Testing**: checks the requirements of a specification or contract are met as per its delivery.

1) **Unit testing:**

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

2) **Integration testing:**

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

3) **System testing:**

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

4) **Acceptance testing:**

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

Other Types of Testing:

- Regression Testing
- Buddy Testing

- Alpha Testing
- Beta Testing

A level of software testing is a process where every unit or component of a software/system is tested.

The primary goal of system testing is to evaluate the system's compliance with the specified needs.

In Software Engineering, four main levels of testing are Unit Testing, Integration Testing, System Testing and Acceptance Testing.

16 a) Explain about software project planning.

What is a metric? Explain the metrics for source code.

Ans. Once a project is found to be possible, computer code project managers undertake project designing. Project designing is undertaken and completed even before any development activity starts. Project designing consists of subsequent essential activities:

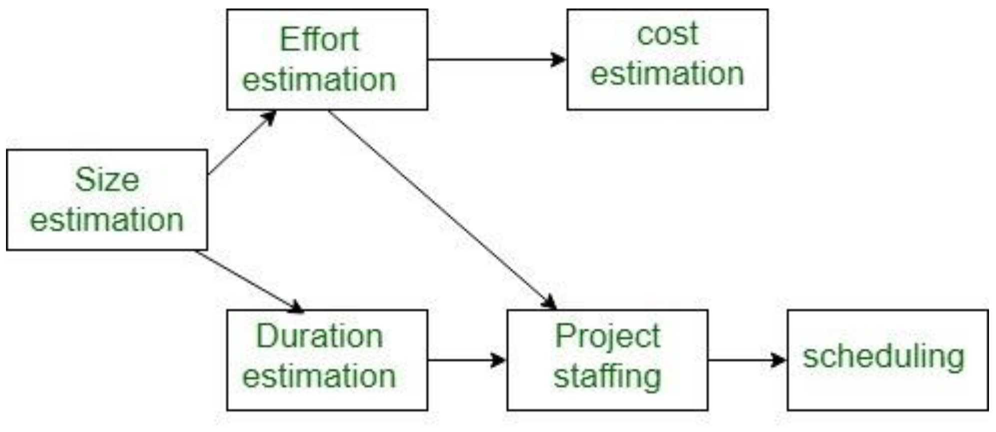
Estimating the subsequent attributes of the project:

- **Project size:**
  What's going to be downside quality in terms of the trouble and time needed to develop the product?
- **Cost:**
  What proportion is it reaching to value to develop the project?
- **Duration:**
  However long is it reaching to want complete development?
- **Effort:**
  What proportion effort would be required?

The effectiveness of the following designing activities relies on the accuracy of those estimations.

- planning force and alternative resources
- workers organization and staffing plans
- Risk identification, analysis, and abatement designing
- Miscellaneous arranges like quality assurance plan, configuration, management arrange, etc.

Precedence ordering among project planning activities:
The different project connected estimates done by a project manager have already been mentioned. The below diagram shows the order during which vital project coming up with activities is also undertaken. It may be simply discovered that size estimation is that the 1st activity. It's conjointly the foremost basic parameter supported that all alternative coming up with activities square measure dispensed, alternative estimations like the estimation of effort, cost, resource, and project length also are vital elements of the project coming up with.



Precedence ordering among planning activities

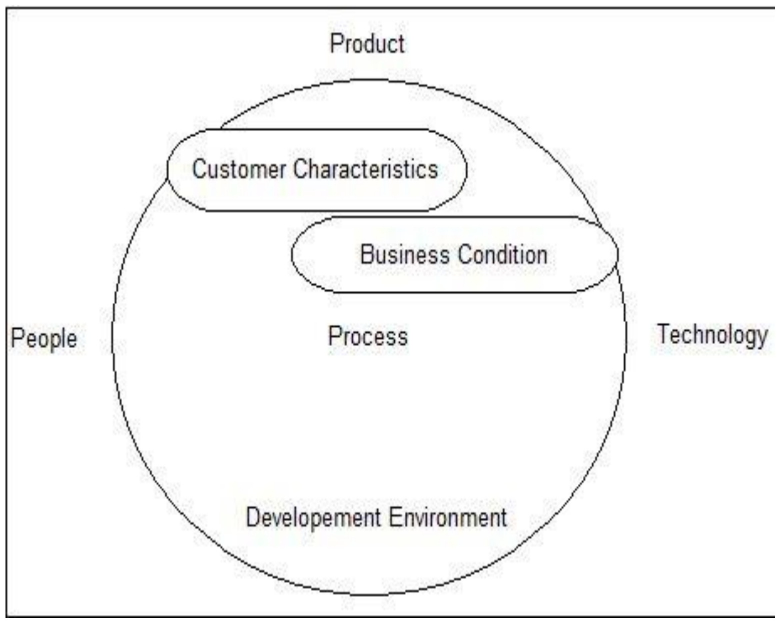
B. What is a metric? Explain the metrics for source code.

Ans. A **software metric** is a standard of measure of a degree to which a **software** system or process possesses some property. Even if a **metric** is not a measurement (**metrics** are functions, while measurements are the numbers obtained by the application of **metrics**), often the two terms are used as synonyms.

Process Metrics

To improve any process, it is necessary to measure its specified attributes, develop a set of meaningful metrics based on these attributes, and then use these metrics to obtain indicators in order to derive a strategy for process improvement.

Using software process metrics, software engineers are able to assess the efficiency of the software process that is performed using the process as a framework. Process is placed at the centre of the triangle connecting three factors (product, people, and technology), which have an important influence on software quality and organization performance. The skill and motivation of the people, the complexity of the product and the level of technology used in the software development have an important influence on the quality and team performance. The process triangle exists within the circle of environmental conditions, which includes development environment, business conditions, and customer /user characteristics.



To measure the efficiency and effectiveness of the software process, a set of metrics is formulated based on the outcomes derived from the process. These outcomes are listed below.

- Number of errors found before the software release
- Defect detected and reported by the user after delivery of the software
- Time spent in fixing errors
- Work products delivered
- Human effort used
- Time expended
- Conformity to schedule
- Wait time
- Number of contract modifications
- Estimated cost compared to actual cost.

Product Metrics

In software development process, a working product is developed at the end of each successful phase. Each product can be measured at any stage of its development. Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements. If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Product metrics help software engineer to detect and correct potential problems before they result in catastrophic defects. In addition, product metrics assess the internal product attributes in order to know the efficiency of the following.

- Analysis, design, and code model
- Potency of test cases
- Overall quality of the software under development.

Various metrics formulated for products in the development process are listed below.

- **Metrics for analysis model:** These address various aspects of the analysis model such as system functionality, system size, and so on.
- **Metrics for design model:** These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.
- **Metrics for source code:** These assess source code complexity, maintainability, and other characteristics.
- **Metrics for testing:** These help to design efficient and effective test cases and also evaluate the effectiveness of testing.
- **Metrics for maintenance:** These assess the stability of the software product.

17 Write short notes on :

a) Evolutionary process models

b) Design evaluation

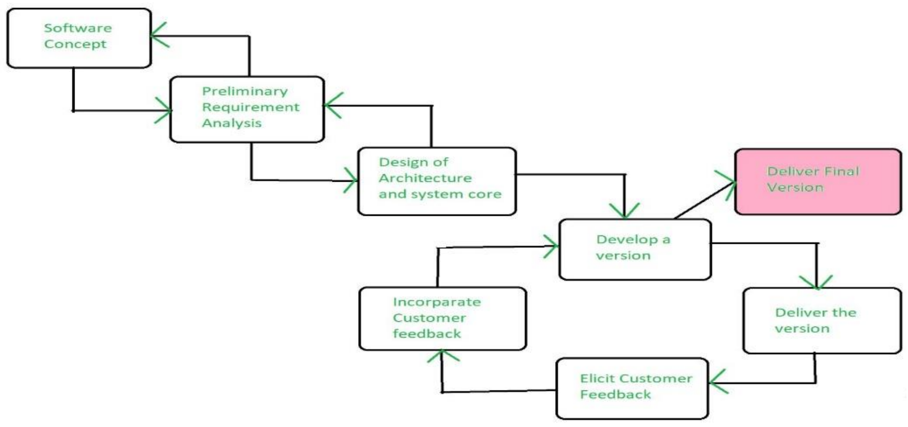
c) Debugging

Ans.

a) Evolutionary process models

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.

It is better for software products that have their feature sets redefined during development because of user feedback and other factors. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system core
Deliver Final Version
Develop a version
Incorparate Customer feedback
Deliver the version
Elicit Customer Feedback

Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer. The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements. The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

b) Design evaluation

Software design and estimation play the key role for software development process. Different methods are used for architecture design and detailed design evaluation. For architectural design stage a technique that allows selecting and evaluating suite of architectural patterns is proposed. It allows us to consistently evaluate the impact of specific patterns to software characteristics with a given functionality. Also the criterion of efficiency metric is proposed which helps us to evaluate architectural patterns for specified software. During detailed design stage we are interested in the selection of the optimal metric suits which takes into account the characteristics of required system. The proposed technique contains a number a steps where at each step a specific criterion should be used to make a selection from the available metric suites. In the end we can perform the selected metric suite improvement.

c) Debugging

Debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Debugging Process: Steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to software engineer to the defect to verify that it is genuine.
- Defect Analysis using modeling, documentations, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.