III/IV B.Tech (Regular) DEGREE EXAMINATION
COMPUTER SCIENCE AND ENGINEERING
Software Engineering Schema

I. Answer the following. **(12*1=12M)**

a. Define the system software.
System software is a collection of programs that are written to service other programs.
Ex. Compilers, Editors, File Management utilities.

b. What is prototyping?
Software prototyping is the activity of creating prototypes of software applications i.e., incomplete versions of the software program being developed.

c. Define Scrum.
Scrum is an Agile software development method. It is designed for teams of three to nine developers who break their work into actions that can be completed within fixed duration cycles (called "sprints"), track progress and re-plan in daily 15-minute stand-up meetings, and collaborate to deliver workable software in every sprint.

d. What is the essence of software engineering practice?
1. Understand the plan.(or)Communication and Analysis.
2. Plan a solution. (or)Modeling of software design.
3. Carry out the plan. .or)Code generation.
4. Examine the result. (or) Testing and Quality Assurance.

e. What is Verification and Validation?

Verification:	Are we building the product right?
Validation:	Are we building the right product?

f. What is a data object?
A Data object can be defined in terms of set of attributes or properties.
Eg.External Entities,things,Occurences,an event,a role,a place,etc..

g. What is abstraction?
The process of abstraction hides all but the relevant data about an object in order to reduce complexity and increase efficiency.

h. Define Persistence.
Persistence refers to object and process characteristics that continue to exist even after the process that created it ceases or the machine it is running on is powered off.

i. What is Control Coupling?
It occurs when operation A() invokes operation B() and passes control fag to B.The control flag then directs logical flow within B.

j. Define LOC.
Lines of code (LOC), is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.

k. What is alpha and beta testing?
Alpha Testing is a type of testing conducted by a team of highly skilled testers at development site whereas Beta Testing is done by customers or end users at their own site.

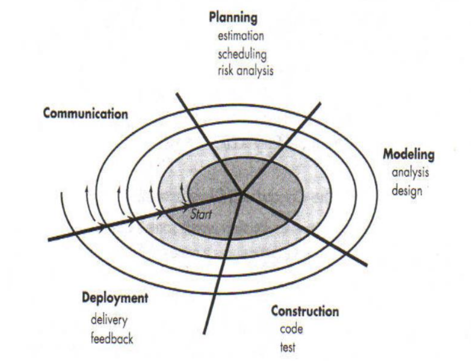
i. Define Unit Testing.
Unit testing is a level of software testing where individual units/ components of software are tested.

UNIT I

2.a)What is process model? Discuss Spiral Model. (6M)

Software Process: A software process model is a simplified representation of software process .These models is abstractions of the process that can be used to explain different approaches to the software development.

Spiral Model:



It is an Evolutionary process model that couples the iterative nature of prototyping and waterfall model. Using the Spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a paper model or prototype, during later iterations, more complete versions of models are produced. A spiral model is divided into set of framework activities. Risk is considered as each revolution is made. Each pass through the planning region results in the adjustment of project plan, cost and schedule.

Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex.

2.b. Discuss about concurrent engineering framework activities (6M)

The concurrent development model, sometimes called concurrent engineering, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states. The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.

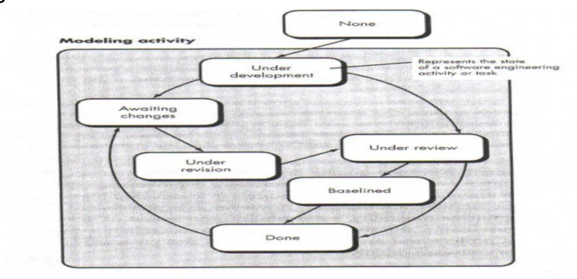


Figure above provides a schematic representation of one Software engineering task within the modeling activity for the concurrent process model. The activity – modeling may be in any one of the states noted at any given time. All activities exist concurrently but reside in different states. For example, early in the project the communication activity has completed its first iteration and exists in the awaiting changes state. The modeling activity which existed in none state while initial communication was completed now makes a transition into underdevelopment state. If, however, the customer indicates the changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state. The concurrent process model defines a series of events that will trigger transitions from state to state for each of the Software engineering activities, actions, or tasks.

3.a)Explain different phases of unified process. (4M)

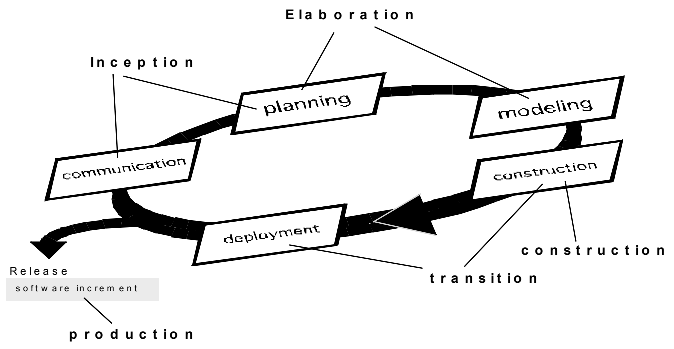
Unified process model is a use-case driven, architecture-centric, iterative and incremental software process closely aligned with the Unified Modeling Language (UML).

Phases of unified process:

The figure below depicts the phases of the UP and relates them to the generic activities.

The **Inception phase** of the UP encompasses both customer communication and planning activities.

By collaborating with the customer and end-users, 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.



A use-case describes a sequence of actions that are performed by an *actor* (person, machine, another system) as the actor interacts with the Software.

The ***elaboration*** **phase** encompasses the customer 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 analysis model, the design model, the implementation model, and the deployment model.

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.

The ***transition*** **phase** of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment 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 "user manuals, trouble-shooting guides, and installation procedures.)

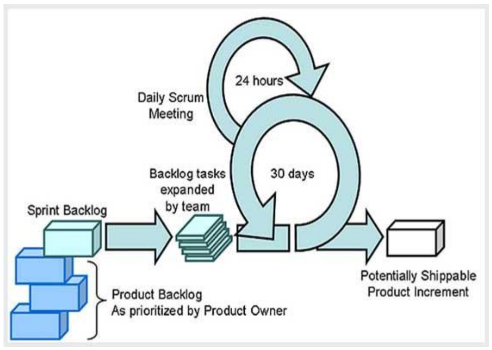
The ***production*** **phase** of the UP coincides with the development activity of the generic process.

b)Explain Scrum process flow with neat diagram. (8M)

Scrum—distinguishing features

Development work is partitioned into **"packets"**

Testing and documentation are on-going as the product is constructed
Work occurs in **“sprints”** and is derived from a **“backlog**” of existing requirements
Meetings are very short and sometimes conducted without chairs
Demo’s are delivered to the customer with the time-box allocated


24 hours
Daily Scrum Meeting
Backlog tasks expanded by team
30 days
Sprint Backlog
Product Backlog As prioritized by Product Owner
Potentially Shippable Product Increment

UNIT-II

4. What is planning? Explain different planning practices. (12M)

PLANNING: The planning activity encompasses a set of management and technical practices that enable the software team to define a road map as it travels towards its strategic goal and technical objectives.

Principle #1: Understand the scope of the project. It’s impossible to use a road map if you don’t know where you’re going. Scope provides the software.

Principle #2: Involve the customer in planning activity. The customer defines priorities and establishes the project constraints.

Principle #3: Recognize that planning is iterative. As work begins, it is very likely that things will change. As a consequence, the plan must be adjusted to accommodate these changes. In addition, iterative and incremental process models dictate re-planning based on feedback received from users.

Principle #4: Estimate based on what you know. The intent of estimation is to provide an indication of effort, cost, and task duration, based on the team’s current understanding of the work to be done.

Principle #5: Consider risk as you define the plan. If the team has defined risks that have high impact and high probability, contingency planning is necessary.

Principle #6: Be realistic. People don’t work 100 percent every day. Noise always enters into any human communication. Omission and ambiguity are facts of life. Change will occur. Even the best software engineers make mistakes. These and other realities should be considered as a project plan is established.

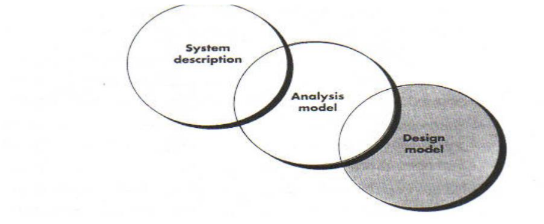
Principle #7: Adjust granularity as you define the plan. Granularity refers to the level of detail that is introduced as a project plan is developed. A “fine granularity” plan provides significant work detail that is planned over relatively short time increments.

Principle #8: Define how you intend to ensure quality. The plan should identify how the software team intends to ensure quality. If formal technical reviews are to be conducted, they should be scheduled.

Principle #9: Describe how you intend to accommodate change. Even the best planning can be obviated by uncontrolled change. The software team should identify how changes are to be accommodated as software engineering work proceeds.

Principle #10: Track the plan frequently and make adjustments are required. Software project falls behind schedule one day at a time. Therefore, it makes sense to track progress on a daily basis, looking for a problem areas and situation in which scheduled work does not confirm to actual work conducted. When slippage is encountered, the plan is adjusted accordingly.

5.a)How analysis model act as a bridge between system description and design model. **(6M)**



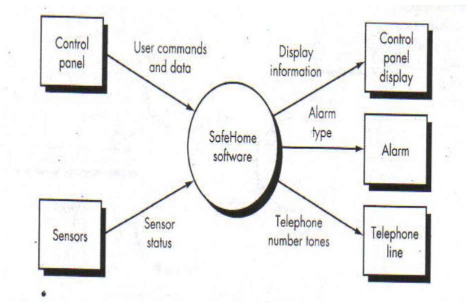
The analysis model must achieve three primary objectives:

- Describe what the customer requires.
- Establish a basis for the creation of a software design.
- Devise a set of requirements that can be validated once the software is built.
- The analysis model bridges the gap between system-level description that describes overall system functionality.
- It is important to note that some elements of the analysis model are present in the system description.
- In addition, all elements of the analysis model are directly traceable to parts of the design model.

5.b)What is DFD?Explain different levels of DFD's with neat diagram. **(6M)**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects.

Different levels of DFD's:



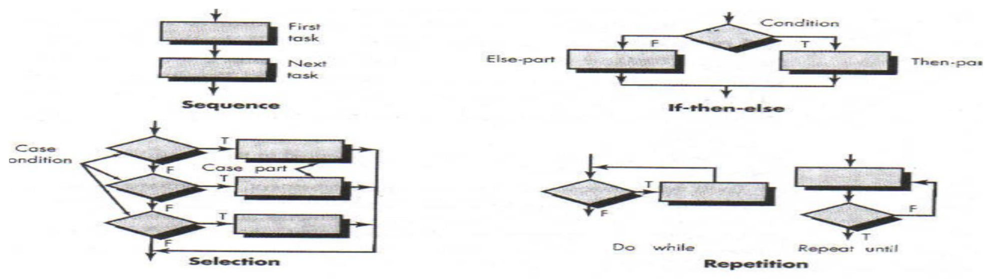
DFD rules

- Each process should have at least one input and an output.Each data store should have at least one data flow in and one data flow out.Data stored in a system must go through a process.All processes in a DFD go to another process or a data store.All icons must be labeled with meaningful names.The dfd evolves through a number of levels of detailAlways begin with a context level diagram (also called level 0).Always show external entities at level 0.Always label data flow arrows.Do not represent procedural logic.Review the data model to isolate data objects and use a grammatical parse to determine "operations".Determine external entities (producers and consumers of data).Create a level 0 dfd.Write a narrative describing the transform.Parse to determine next level transforms.Balance the flow to maintain data flow continuity
- Develop a level 1 dfd.Use a 1:5 (approx.) Expansion ratio.Each bubble is refined until it does just one thingThe expansion ratio decreases as the number of levels increase.Most systems require between 3 and 7 levels for an adequate flow model.A single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)

UNIT-III

6.a)Explain different conventional components for design. **(6M)**

i. **Graphical (e.g. Flowchart, box diagram)**
   - It uses a limited set of logical constructs
     i)Sequence ii)Selection iii)Repetation.
   - It can be used in conjunction with 'proof of correctness'.
   - It leads to more readable, testable code



ii. **Decision table**
Decision tables translates actions and conditions into a tabular form.A decision table is used when a complex set of conditions and actions are encountered within a component.Decision table is divided into four quadrants. Upper left quadrant lists all conditions. Lower left quadrant lists all actions. Right hand quadrants form a matrix indicating condition combinations and corresponding actions

iii. **Pseudo code (e.g., PDL)**
It is easy to combine with source code.It is machine readable.Graphics can be generated from PDL.It is easier to maintain.

iv. **Conduct walkthrough to assess quality.**

b) Discuss about different design issues. (6M)

Design Challenge #1: Requirements Volatility
A major reason for the complexity of software projects is the constant change of requirements. When designed properly, software can be modified or extended easily; however, when designed poorly, modifying software can become overwhelming and lead to all sorts of complex problems.

Design Challenge #2: The Process
Software Engineering is a process oriented field. Software processes allows engineers to organize the steps required to develop software solutions with schedule and cost constraints. Therefore, at the core of every software development company, there should be a sound, well understood, and consistent process for software development. Processes can also be developed and customized for particular phases of the software engineering life-cycle.

Design Challenge #3: The Technology
Software is meant to be everywhere. From healthcare systems, education, defense, and everyday ubiquitous devices, software is required to operate on a massive and always evolving technology landscape. Besides the operating environment, the technology for designing and implementing today's software systems continues to evolve to provide improved capabilities.

Design Challenge #4: The Ethical and Professional Practices
Designers create blueprints that drive the construction of the software. During this creation process, designers are required to determine how design decisions affect the environment and the people that use the software. In many cases, the software development process is traditionally carried out under tight schedule constraints. Inherently, all phases of the development life-cycle suffer from this, including the design phase.

Design Challenge #5: Managing Design Influences
Designs are shaped by many different influences from stakeholders, the development organization, and other factors. These influences can have cyclical effects between the system and its external influences, such that external factors affect the development of the system and the system affects its external factors

7.a)Explain cohesion and coupling methods. (6M)

Cohesion (lowest to highest) (3M)

- **Utility cohesion** – components grouped within the same category but are otherwise unrelated
- **Temporal cohesion** – operations are performed to reflect a specific behavior or state

- **Procedural cohesion** – components grouped to allow one be invoked immediately after the preceding one was invoked with or without passing data
- **Communicational cohesion** –operations required same data are grouped in same class
- **Sequential cohesion** – components grouped to allow input to be passed from first to second and so on
- **Layer cohesion** – exhibited by package components when a higher level layer accesses the services of a lower layer, but lower level layers do not access higher level layer services
- **Functional cohesion** – module performs one and only one function.

Coupling (3M)

- **Content coupling** – occurs when one component surreptitiously modifies internal data in another component
- **Common coupling** – occurs when several components make use of a global variable
- **Control coupling** – occurs when one component passes control flags as arguments to another
- **Stamp coupling** – occurs when parts of larger data structures are passed between components
- **Data coupling** – occurs when long strings of arguments are passed between components
- **Routine call coupling** – occurs when one operator invokes another
- **Type use coupling** – occurs when one component uses a data type defined in another
- **Inclusion or import coupling** – occurs when one component imports a package or uses the content of another
- **External coupling** – occurs when a components communications or collaborates with infrastructure components (e.g. database)

7.b)What are the types of golden rules? (6M)

- **Place the user in control (2M)**
- **Reduce the user's memory load (2M)**
- **Make the interface consistent (2M)**

Place the User in Control

i. Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
ii. Provide for flexible interaction.
iii. Allow user interaction to be interruptible and undoable.
iv. Streamline interaction as skill levels advance and allow the interaction to be customized.
v. Hide technical internals from the casual user.
vi. Design for direct interaction with objects that appear on the screen.

Reduce the User's Memory Load

i. Reduce demand on short-term memory.
ii. Establish meaningful defaults.
iii. Define shortcuts that are intuitive.
iv. The visual layout of the interface should be based on a real world metaphor.
v. Disclose information in a progressive fashion.

Make the Interface Consistent

i. Allow the user to put the current task into a meaningful context.
ii. Maintain consistency across a family of applications
iii. If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

UNIT-IV

8. a)What are different usecase oriented metrics. (6M)

Use Case-Oriented Metrics

- Describe (indirectly) user-visible functions and features in language independent manner.

- Number of use case is directly proportional to LOC size of application and number of test cases needed.
- However use cases do not come in standard sizes and use as a normalization measure is suspect.
- Use case points have been suggested as a mechanism for estimating effort.

b) Explain functional based metrics. **(6M)**

- Function points are computed from direct measures of the information domain of a business software application and assessment of its complexity.
- Once computed function points are used like LOC to normalize measures for software productivity, quality, and other attributes.
- The relationship of LOC and function points depends on the language used to implement the software.
- The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design
- Function points and LOC-based metrics have been found to be relatively accurate predictors of software development effort and cost
- Using LOC and FP for estimation a historical baseline of information must be established.

9. Discuss different metrics used for the following : **(4+4+4M)**

a)Source code **b)Testing** **c)Maintenance**

a)Source code:

Lines of Code: Functions, classes, files and interfaces can indicate do-everything code that is hard to maintain and costly in the long term. It's infinitely hard to compare the total lines of code versus what a system does. It could be something that does many things and in that case there will be many lines of code.

Complexity: This measurement is good to do on code bases you haven't worked on, and can give you a good indication of where problem areas lie. If management had these measurements at hand they could plan refactoring iterations or redesigns of specific areas of a system.

Code duplication: This is a very important measurement. Code duplication is a very bad sign and could point to either deep problems in low levels of a system's design or developers that are copy pasting, causing massive problems in the long term and systems that are unmaintainable.

Dependency Graphs: Finding bad dependencies and circular dependencies are an important measurement in code. This almost always points to an incorrect high level design that needs revising.

b)Testing Metrics:

Testing Metrics are mainly divided into 2 categories.

1. Base Metrics
2. Calculated Metrics

BaseMetrics: Base Metrics are the Metrics which are derived from the data gathered by the Test Analyst during the test case development and execution.This data will be tracked throughout the Test Lifecycle. i.e., collecting the data like Total no. of test cases developed for a project (or) no. of test cases need to be executed (or) no. of test cases passed/failed/blocked etc.

CalculatedMetrics: Calculated Metrics are derived from the data gathered in Base Metrics. These Metrics are generally tracked by the test lead/manager for Test Reporting purpose.

c)Maintenance Metrics:

- Maintenance Backlog.
- Mean Time To Repair (MTTR)
- Mean Time Between Failures (MTBF)
- Overall Equipment Effectiveness (OEE)
- Preventive Maintenance (PM) Compliance.
- Planned Maintenance Percentage.