SUMMER – 13 EXAMINATION
Model Answer

Subject Code: **12175** Subject Name**: Software Engineering**

Important Instructions to examiners:
1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principl components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

1. A) Attempt any three: [12 Marks]

a) Define the following :

(2 marks for each definition)

Answer:

i) Modeling

Modeling is the translation of requirements into a blueprint of the system to be built. The functional model is created to gain a better understanding of the actual entity to be built. Eg- Functional , behavioural model etc.

ii) Software Engineering

Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

b) What are the characteristics of testing strategies?

(any 4 points, 1 mark for each point)

Answer:

The following are the characteristics of testing strategies:

1. Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
2. Different testing techniques are appropriate at different points in time.
3. Testing is conducted by the developer of the software and (for large projects) an independent test group.

4. Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
5. A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

c) What is project schedule in general?

(1 mark for definition, 3 marks for principle)

Answer:

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is important to note, however, that the schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major software engineering activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here, specific software tasks (required to accomplish an activity) are identified and scheduled.

Like all other areas of software engineering, a number of basic principles guide software project scheduling:

- **Compartmentalization.** The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are decomposed .
- **Interdependency.** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.
- **Time allocation.** Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.
- **Effort validation.** Every project has a defined number of staff members. As time allocation occurs, the project manager must ensure that no more than the allocated number of people has been scheduled at any given time.

- **Defined responsibilities**. Every task that is scheduled should be assigned to a specific team member.
- **Defined outcomes.** Every task that is scheduled should have a defined out come.
- **Defined milestones.** Every task or group of tasks should be associated with a project milestone.

Program evaluation and review technique (PERT) and critical path method (CPM) are two project scheduling methods that can be applied to software development.

d) What is the meaning of Software Estimation?

(2 marks for concept and 2 marks for methods)

Answer:

Software Project Estimation basically includes estimation of cost and effort estimation.

Variables like human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it.

Software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!).
2. Base estimates on similar projects that have already been completed.
3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience (historical data) and takes the form

$d = f(vi)$

where *d* is one of a number of estimated values (e.g., effort, cost, project duration) and *vi* are selected independent parameters (e.g., estimated LOC or FP).

• For example # of people = LOC ÷(Duration*(LOC/PM))

LOC = line of code

PM = person month

Precise estimation is difficult. So, make three estimates:

optimistic

most likely

pessimistic

Then combine as:

EV = (Sopt + 4*Sm + Spess)/6

B] Attempt any one: [6 Marks]

a) Explain the rule that should be followed for creating the analysis model.
(6 marks for all rules)

Answer:

The following are the rules for creating the analysis model :-

1. The information domain of a problem must be represented and understood.

2. The functions that the software is to perform must be defined.

3. The behavior of the software (as a consequence of external events) must be represented.

4. The models that depict information ,function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.

5. The analysis process should move from essential information toward implementation detail.

By applying these principles, the analyst approaches a problem systematically. The information domain is examined so that function may be understood more completely.

b) What are the design quality guidelines?

(any 6 points, 1 mark for each point)

Answer:

The following are the design quality guidelines:

1. A design should exhibit an architectural structure that has been created using recognizable design patterns, is composed of components that exhibit good design characteristics, and can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.

2. A design should be modular; that is, the software should be logically partitioned into elements that perform specific functions and subfunctions.

3. A design should contain distinct representations of data, architecture, interfaces, and components (modules).

4. A design should lead to data structures that are appropriate for the objects to be implemented and are drawn from recognizable data patterns.

5. A design should lead to components that exhibit independent functional characteristics.

6. A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.

7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

2. Attempt any four: [16 marks]

a) What are the characteristics of good design?

***(4 marks for all points*)**

Answer:

Three characteristics that serve as a guide for the evaluation of a good design:

1. The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
2. The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
3. The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

b) Explain briefly:

(2 marks for each method)

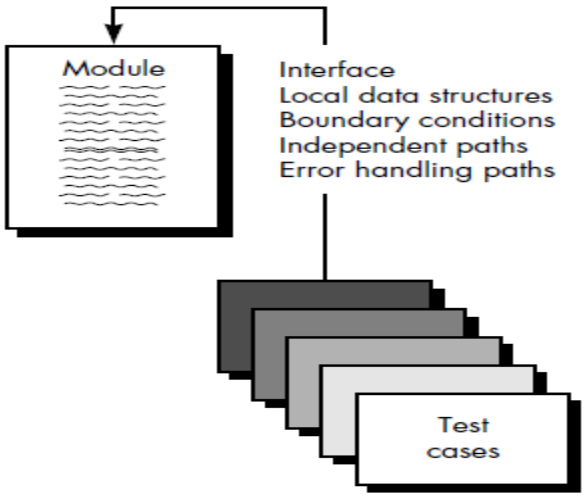
Answer:

i) Unit Testing

(1 mark for definition, 1 mark for explanation, fig is optional)

Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests

and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components



The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.

ii) System Testing

(1 mark for definition, 1 mark for explanation)

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

The software engineer should anticipate potential interfacing problems and (1) design error-handling paths that test all information coming from other elements of the system, (2) conduct a series of tests that simulate bad data or other potential errors at the software interface, (3) record the results of tests to use as "evidence" if finger-pointing does occur, and (4) participate in planning and design of system tests to ensure that software is adequately tested.

Examples of system testing are:

Recovery testing, Security testing, Stress testing, Performance testing

c) Explain about software quality assurance.
(2 marks for explanation, 2 marks activities)

Answer:

Software quality is defined as Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

SQA encompasses (1) a quality management approach, (2) effective software engineering technology (methods and tools), (3) formal technical reviews that are applied throughout the software process, (4) a multitiered testing strategy, (5) control of software documentation and the changes made to it, (6) a procedure to ensure compliance with software development standards (when applicable), and (7) measurement and reporting mechanisms

SQA Activities –

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

1. Prepares an SQA plan for a project.
2. Participates in the development of the project's software process description.
3. Reviews software engineering activities to verify compliance with the defined software process.
4. Audits designated software work products to verify compliance with those defined as part of the software process.
5. Ensures that deviations in software work and work products are documented and handled according to a documented procedure.
6. Records any noncompliance and reports to senior management.

d) Describe the four layers of S/W Engg.

(1 mark for figure, 3 marks for layer explanation)

Answer:

Software engineering encompasses a process, the management of activities, technical methods, and use of tools to develop software products.

The bedrock that supports software engineering is a quality focus. The foundation for software engineering is the *process* layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of key process areas (KPAs) that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Software engineering methods provide the technical how-to's for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

e) Explain the terms :

(2 marks for each question)

Answer:

i) Data objects.

(1 mark for definition, 1 mark for explanation)

A data object is a representation of almost any composite information that must be understood by software. By composite information, we mean something that has a number of different properties or attributes. Therefore, width (a single value) would not be a valid data object, but dimensions (incorporating height, width,and depth) could be defined as an object.

A data object can be an external entity (e.g., anything that produces or consumes information), a thing (e.g., a report or a display), an occurrence (e.g., a telephone call) or event (e.g., an alarm), a role (e.g., salesperson), an organizational unit (e.g., accounting department), a place (e.g., a warehouse), or a structure (e.g., a file). For example, a person or a car (Figure 12.2) can be viewed as a data object in the sense that either can be defined in terms of a set of attributes. The data object description incorporates the data object and all of its attributes.

In this case, a car is defined in terms of make, model, ID number, body type, color and owner. The body of the table represents specific instances of the data object.

Object: Customer

ii) Data attribute

(1 mark for definition, 1 mark for explanation)

Attributes. Attributes define the properties of a data object and take on one of three different characteristics. They can be used to (1) name an instance of the data object, (2) describe the instance, or (3) make reference to another instance in another table. In addition, one or more of the attributes must be defined as an *identifier*—that is, the identifier attribute becomes a "key" when we want to find an instance of the data object. In some cases, values for the identifier(s) are unique, although this is not a requirement. Referring to the data object **car**, a reasonable identifier might be the ID number.

The set of attributes that is appropriate for a given data object is determined through an understanding of the problem context. The attributes for **car** might serve well for an application that would be used by a Department of Motor Vehicles, but these attributes would be useless for an automobile company that needs manufacturing control software.

Attributes:

name

company name

status of contact

f) Enlist the reasons of failure of software project

(1 mark for each reason, any 4 reasons)

Answer :

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

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

3. Attempt any Four: [16 marks]

a) What are reactive and proactive risk strategies?

Answer:

Reactive risk strategies:-

(Explanation -2 marks)

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

Proactive risk strategies:-

(Explanation- 2 marks)

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

b) Enlist the factor on which the success of s/w project depend.

(any 8 points, ½ mark for each point)

Answer:

1. Proper project planning
2. Cultivate a Zero Defect Mentality
3. Proper Understanding of Customer Requirements
4. Timely execution of various project phases
5. Address the Root Cause
6. Efficient Risk management
7. Data Based Approach
8. Use the Correct Tools
9. Enlist Top Management Support
10. Enlist Local Process Owner Support
11. Involve the Rank and File
12. Manage Resistance to Change
13. Leadership

Even if students are explaining 6sigma, ISO, McCall's Quality factor it's correct

c) Explain risk refinement.

(1 mark for concept, 3 marks for explanation)

Answer:

Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.

CTC (condition-transition-consequence) format may be a good representation for the detailed risks (e.g. given that <condition> then there is a concern that (possibly) <consequence>).

Reusable software components must conform to specific design standards and that some do not conform, and then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

Subcondition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.

Subcondition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Subcondition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.

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

d) Give possible reasons of why s/w is delivered late.

(any 8 points, ½ mark for each point)

Answer:

1. Insufficient front-end planning
2. Unrealistic project plan
3. Project scope underestimated
4. Customer/management changes
5. Insufficient contingency planning
6. Inability to track progress

7. Inability to track problem early
8. Insufficient number of checkpoints
9. Staffing problems
10. Technical complexity
11. Sinking team spirit
12. Unqualified resources
13. Product is unknown to customer
14. Requirements late
15. Requirements of insufficient quality
16. (Specifications of) delivered software late.
17. (Specifications of) delivered software of insufficient quality

e) Write steps that required to perform statistical s/w quality assurance

(1 mark for each point)

Answer:

Statistical quality assurance reflects a growing trend throughout industry to become more quantitative about quality. For software, statistical quality assurance implies the following steps:

1. Information about software defects is collected and categorized.
2. An attempt is made to trace each defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, and poor communication with the customer)
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few").
4. Once the vital few causes have been identified, move to correct the problems that have caused the defects.

4 . A) Attempt any three: **[12 Marks]**

a) List different approaches for software sizing.

(1 mark for each approach, explanation is not expected only list)

Answer:

a. "Fuzzy logic" sizing:

b. Function point sizing:

c. Standard component sizing:

d. Change sizing:

b) What is software validation and verification?

(2 marks for definitions, 2 marks for explanation)

Answer:

Verification and Validation

Software testing is one element of a broader topic that is often referred to as verification and validation (V&V).

Verification refers to the set of activities that ensure that software correctly implements a specific function.

Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.

Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

The definition of verification and validation encompasses many activities that we refer to as software quality assurance.

1. Software engineering activities provide a foundation from which quality is built.

2. Analysis, design, coding methods act to enhance the quality by providing uniform techniques and predictable results.

3. Formal technical reviews help to ensure the quality of the products.

4. Throughout the process, measures and controls are applies to every element of software configuration. These help to maintain uniformity.

5. Testing is the last phase in which quality is accessed and errors are uncovered.

c) What is meant by well-formed design class?

(any 4 points each carry 1 mark)

Answer:

1) Classes must contain all the attributes, which are related to that domain
2) Classes must contain required function with respect to domain
3) All classes must be connected by means of association, generalization/specialization, inheritance, or link.
4) Cohesion and coupling must be understood and should be consider in designing
5) A class is well formed if its field and methods are well formed and overriding methods preserve types.
6) A method is well formed if its body is well formed under the type environment, method's formal parameters, local variable declarations, or fields and special variable return, used for returning a value from the method. Well-formed statement can be sequenced in standard fashion into a well-formed compound statement.

d) What is white box and black box testing?

Answer:

White-box testing

(Explanation -2 marks)

White-box testing**,** sometimes called glass-box testing or code testing, is a test case design method that uses the control structure of the procedural design to derive test cases.

Using white-box testing methods, the software engineer can derive test cases that

1. Guarantee that all independent paths within a module have been exercised at least once.
2. Exercise all logical decisions on their true and false sides,
3. Execute all loops at their boundaries and within their operational bounds, and
4. Exercise internal data structures to ensure their validity.

Even though white box testing seems to be an ideal method for testing, it does not guarantee failures from faulty data. Secondly it is difficult to conduct such extensive and exhaustive testing in large organizations.

Black -box testing

(Explanation-2 marks)

Black Box testing: In this method the analyst examines the specifications to see what the program must do under various conditions. Test cases are then developed for a condition or a combination of conditions and submitted for processing. By examining the results the analyst can determine if the specified requirements are satisfied.

Black-box testing attempts to find errors in the following categories:

1. Incorrect or missing functions,
2. Interface errors,
3. Errors in data structures or external data base access,
4. Behavior or performance errors, and
5. Initialization and termination errors.

B) Attempt Any One: [6 Marks]

i) How do the process models differ from each other?

(1 mark for each comparison)

(Remark: comparison between two or more models should be considered)

Answer:

A software process model is an abstract representation of a process methodology. Waterfall is a process model. Agile is a process model. They don't specify how to do things, but outline the types of things that are done. For example, Waterfall identifies the phases that a project goes through - requirements, design, implementation/unit testing, integration testing, system testing, deployment - without saying what artifacts to produce or what tools to use (although the output of code is implied). Agile defines core values in the form of the Agile manifesto, time-boxed iterations, and continuous response to change, but it doesn't say how long your iterations should be or how you go about responding to change. The Spiral model is a third software process model.

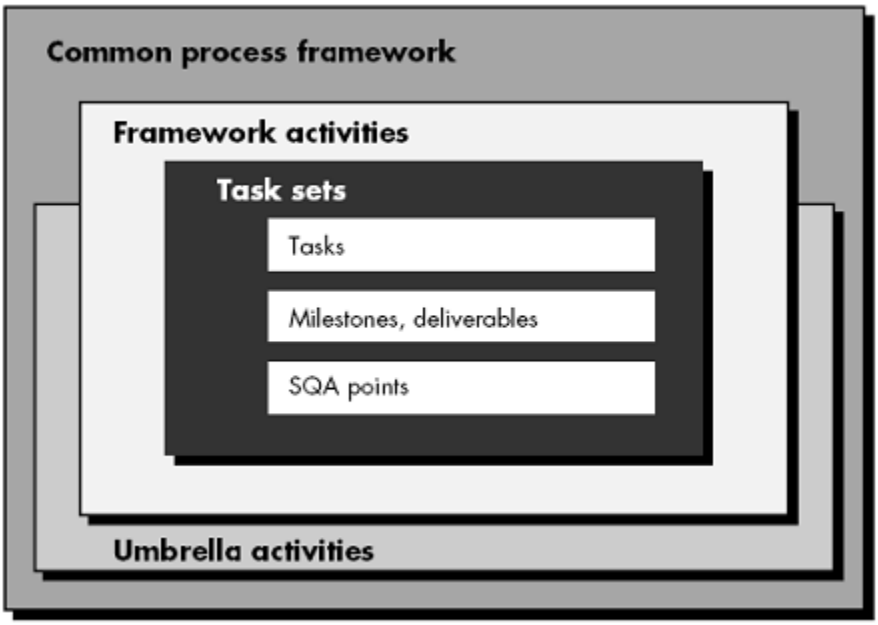
A software process methodology is a specific way of conducting a software project. These are things like the Rational Unified Process and Scrum. They define exactly what, when, and/or how various artifacts are produced. They might not be entirely explicit with all regards - for example, Scrum doesn't identify what documents to produce or not to produce, since it's focus is on delivering value to the customer - but they define, in some way, the actions that members of the project team must deliver.

ii) Explain the software development process-Umbrella activities.

(2 marks for figure, 2 marks for explanation)

Answer:

A software process can be characterized as shown in the diagram



1) A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
2) A number of task sets—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
3) Umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model.
4) Umbrella activities are independent of any one framework activity and occur throughout the process.
5) Software Engineering Umbrella Activities (Software project tracking and control, Formal technical reviews, Software quality assurance, Software configuration management, Document preparation and production, Reusability management, Measurement, Risk management)

5. Attempt any two: [16 marks]

a) With neat diagram explain the translation of analysis model into design model

(3 marks for figure, 5 marks for explanation)

(Remark: The above topic is not mentioned in the curriculum)

Ans:

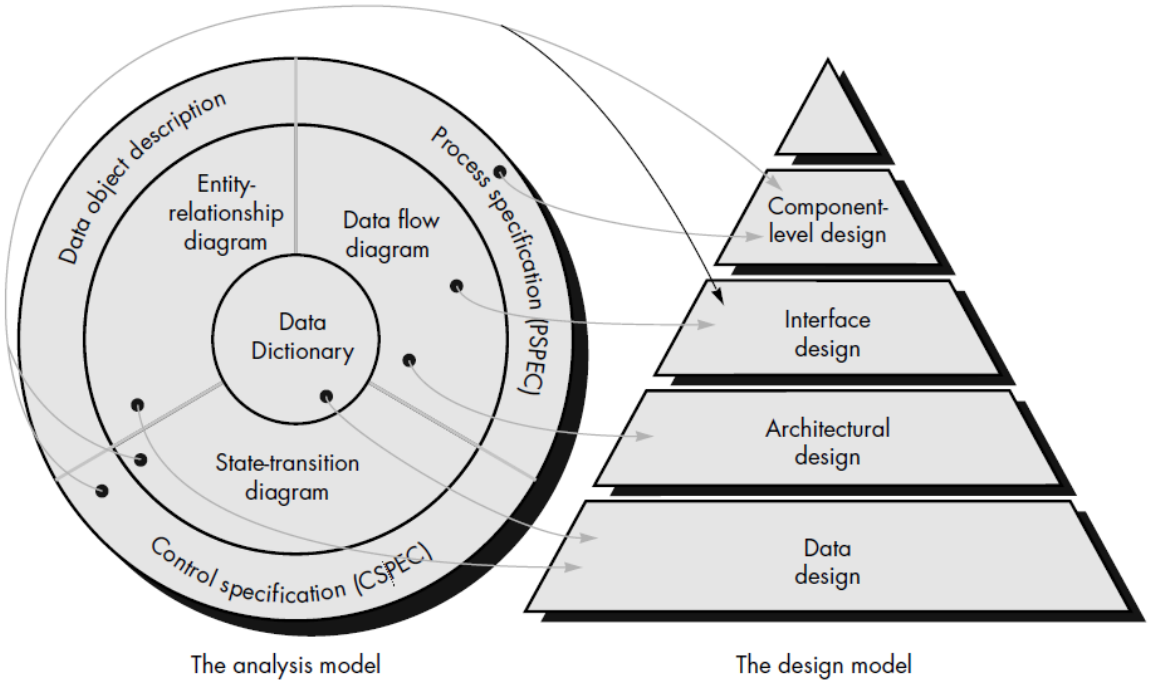


FIGURE 13.1 Translating the analysis model into a software design

Software design sits at the technical kernel of software engineering and is applied regardless of the software process model that is used. Beginning once software requirements have been analyzed and specified, software design is the first of three technical activities—design, code generation, and test—that are required to build and verify the software. Each activity transforms information in a manner that ultimately results in validated computer software.

Each of the elements of the analysis model provides information that is necessary to create the four design models required for a complete specification of design. The flow of information during software design is illustrated in above figure.

Software requirements, manifested by the data, functional, and behavioral models, feed the design task. Using one of a number of design methods (discussed in later chapters), the design task produces a data design, an architectural design, an interface design, and a component design.

The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software. The data objects and relationships defined in the entity relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design activity.

Part of data design may occur in conjunction with the design of software architecture. More detailed data design occurs as each software component is designed. The architectural design defines the relationship between major structural elements of the software, the "design patterns" that can be used to achieve the requirements that have been defined for the system, and the constraints that affect the way in which architectural design patterns can be applied

The architectural design representation the framework of a computer-based system—can be derived from the system specification, the analysis model, and the interaction of subsystems defined within the analysis model. The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it. An interface implies a flow of information (e.g., data and/or control) and a specific type of behavior. Therefore, data and control flow diagrams provide much of the information required for interface design.

The component-level design transforms structural elements of the software architecture into a procedural description of software components. Information obtained from the PSPEC, CSPEC, and STD serve as the basis for component design.

b) With neat diagram explain clean- room S/W engineering approach

(3 marks for Diagram 5 marks for explanation of various phases)

Ans :

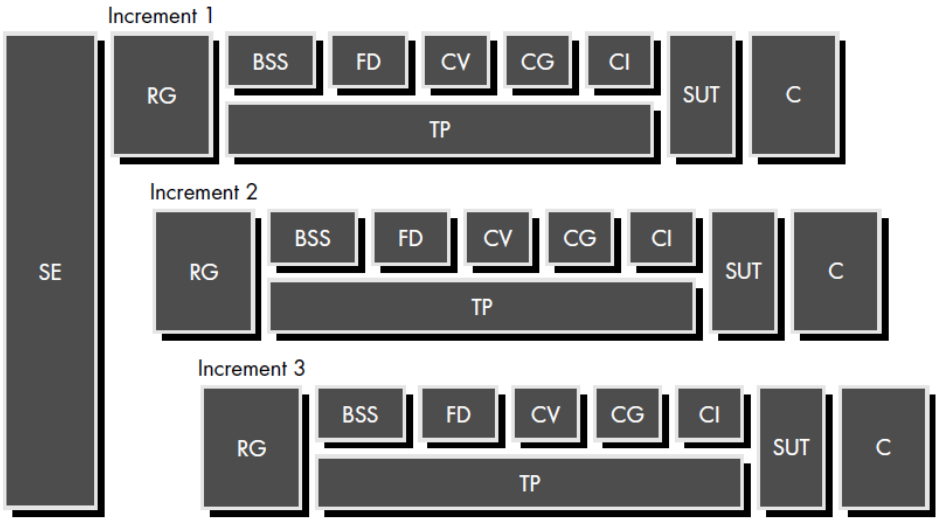


FIGURE 26.1
The cleanroom process model

The clean room approach makes use of a specialized version of the incremental software model. A pipeline of software increments is developed by small independent software engineering teams. As each increment is certified, it is integrated in the whole. Hence, functionality of the system grows with time. Overall system or product requirements are developed using the system engineering methods. Once functionality has been assigned to the software element of the system, the pipeline of clean room increments is initiated. The following tasks occur: Increment planning. A project plan that adopts the incremental strategy is developed. The functionality of each increment, its projected size, and a clean room development schedule are created. Special care must be taken to ensure that certified increments will be integrated in a timely manner.

Requirements Gathering: Using techniques a more-detailed description of customer-level requirements (for each increment) is developed.

Box Structure Specification: A specification method that makes use of box structures is used to describe the functional specification. Conforming to the operational analysis principles, box structures isolate and separate the creative definition of behavior, data, and procedures at each level of refinement.

Formal Design: Using the box structure approach, clean room design is a natural and seamless extension of specification. Although it is possible to make a clear distinction between the two activities, specifications (called black boxes) are iteratively refined (within an increment) to become analogous to architectural and component-level designs (called state boxes and clear boxes, respectively).

Correctness Verification: The clean room team conducts a series of rigorous correctness verification activities on the design and then the code. Verification begins with the highest-level box structure (specification) and moves toward design detail and code. The first level of correctness verification occurs by applying a set of correctness questions. If these do not demonstrate that the specification is correct, more formal (mathematical) methods for verification are used.

Code Generation, Inspection, And Verification: The box structure specifications, represented in a specialized language, are translated into the appropriate programming language. Standard walkthrough or inspection techniques are then used to ensure semantic conformance of the code and box structures and syntactic correctness of the code. Then correctness verification is conducted for the source code.

Statistical Test Planning: The projected usage of the software is analyzed and a suite of test cases that exercise a probability distribution of usage are planned and designed. Clean room activity is conducted in parallel with specification, verification, and code generation.

Statistical Use Testing. Recalling that exhaustive testing of computer software is impossible, it is always necessary to design a finite number of test cases. Statistical use techniques execute a series of tests derived from a statistical sample (the probability distribution noted earlier) of all possible program executions by all users from a targeted population.

Certification: Once verification, inspection, and usage testing have been completed (and all errors are corrected), the increment is certified as ready for integration.

c) Explain in detail RAD model with neat diagram

(2 marks for Diagram 6 marks for explanation of various phases)

Answer:

RAD Model

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods (e.g., 60 to 90 days). Used primarily for information systems applications, the RAD approach encompasses the following phases

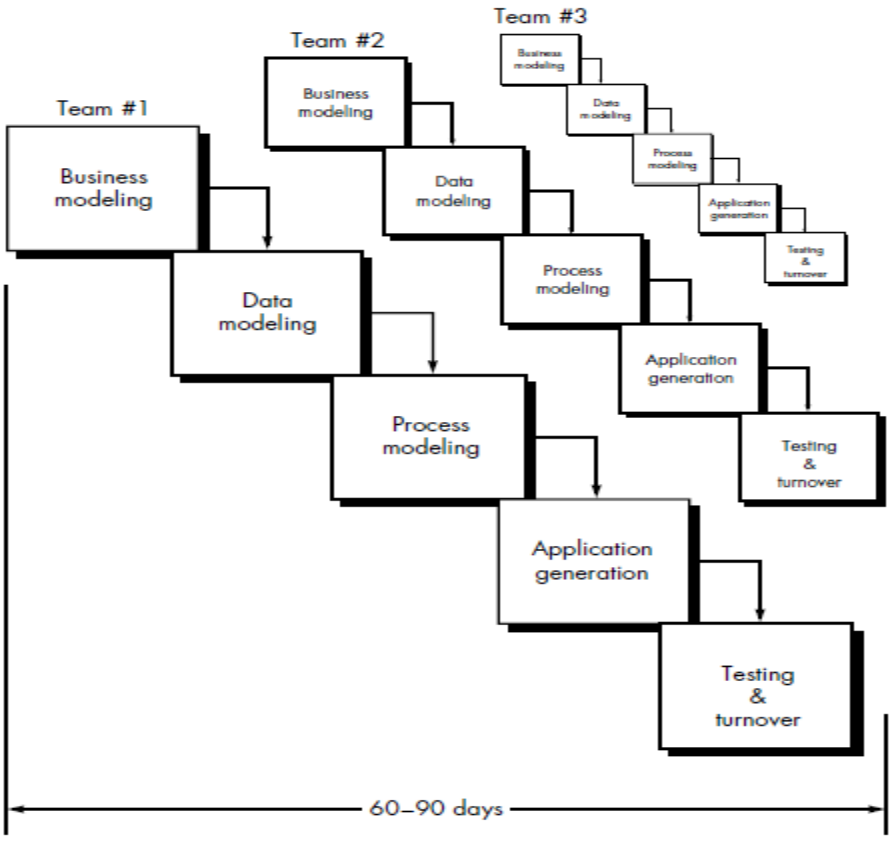


FIGURE 2.6
The RAD model

Business modeling: - The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

Data modeling: - The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined. Data modeling is considered in.

Process modeling: - The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation: - RAD assumes the use of fourth generation techniques (Section 2.10). Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and turnover: - Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

6. Attempt Any Four **[16 Marks]**

a) Explain the importance of decomposing the task

(any 4 points, each 1 mark for each point)

(Remark:-Any generalized importance shall be considered and assessed accordingly)

Answer:

Following are the importance of decomposing the task

1. Ease of allocation

   The task can be easily allocated to the team members
2. Ease of monitoring task

   The task can be easily monitored so that its status can be easily tracked
3. Ease of completion

   Smaller task can be easily completed.
4. Proper management

   The manager can have proper co-ordination with the team hence can manage them properly.
5. Higher success rate

   Completion of small task is easy and hence project completion is assured.

b) List the characteristics of software

(List of all three characteristic 4 marks, explanation is not expected)

Answer:

Following are the characteristics of Software

1. Software is developed or engineered; it is not manufactured in the classical sense.

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

2. Software doesn't "wear out".

The aspect of wear illustrates the difference between hardware and software. When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity than hardware maintenance.

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

A software component should be designed and implemented so that it can be reused in many different programs. In the 1960s, we built scientific subroutine libraries that were reusable in a broad array of engineering and scientific applications. These subroutine libraries reused well-defined algorithms in an effective manner but had a limited domain of application. Today, we have extended our view of reuse to encompass not only algorithms but also data structure. Modern reusable components encapsulate both data and the processing applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today's graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction

mechanisms. The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.

c) Give various risk component
(1 mark for each component)

Answer

The risk components are defined in the following manner:

1. *Performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
2. *Cost risk*—the degree of uncertainty that the project budget will be maintained.
3. *Support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
4. *Schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

d) What is alpha and beta testing?
(2 marks for alpha testing; 2 marks for beta testing)

Answer:

Alpha Testing: -The *alpha test* is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.

Beta Testing: - The *beta test* is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.

e) What are debugging strategies?
(List of various strategies-1 mark; 1 mark for explanation of each strategy)

Answer

There are three Different Debugging Strategies are available as follows:-

(1) Brute Force,

(2) Backtracking, And

(3) Cause Elimination.

The brute force category of debugging is probably the most common and least efficient method for isolating the cause of a software error. Brute force debugging methods are applied when all else fails. Using a "let the computer find the error" philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. In the morass of information that is produced a clue is found that can lead us to the cause of an error. Although the mass of information produced may ultimately lead to success, it more frequently leads to wasted effort and time. Thought must be expended first.

Backtracking is a fairly common debugging strategies that can be used successfully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backward (manually) until the site of the cause is found. Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.

The third strategies to debugging—cause elimination—is manifested by induction or deduction and introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes. A "cause hypothesis" is devised and the aforementioned data are used to prove or disprove the hypothesis. Alternatively, a list of all possible causes is developed and tests are conducted to eliminate each. If initial tests indicate that a particular cause hypothesis shows promise, data are refined in an attempt to isolate the bug.