UNIT 1

Introduction to Software Engineering, A Generic View of Process, Process Models

Syllabus

Introduction to Software Engineering: The Evolving Role of Software, Changing Nature of Software, Software Myths.

A Generic View of Process: Software Engineering - A Layered Technology, A Process Framework, The Capability Maturity Model Integration (CMMI), Process Patterns, Process Assessment, Personal and Team Process Models.

Process Models: The Waterfall Model, Incremental Process Models, Evolutionary Process Models, The Unified Process.

PART-A SHORT QUESTIONS WITH SOLUTIONS

Q1. Mention some of the factors to be considered during System modelling.

Answer : May-18(R15), Q1(b)

The following are the restraining factors that has to be taken into consideration by the developer for creating a model.

1. Assumptions
2. Simplification
3. Limitations
4. Constraints
5. Preferences.

Q2. What is Software Development Life Cycle?

Answer : May-18(R15), Q1(a)

Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) refers to the overall process involved in the software development. It involves five phases namely,

1. Requirements Analysis Phase
2. Design Phase
3. Implementation and Unit Testing
4. Testing Phase
5. Deployment Phase
6. Maintenance Phase.

Q3. Define the term software and software engineering. Model Paper-I, Q1(a)

OR

Define Software and its characteristics. Dec.-19(R16), Q1(b)

(Refer Only Topics: Software, Characteristics of Software)

OR

Define software engineering.

(Refer Only Topic: Software Engineering)

Answer : May/June-19(R16), Q1(a)

Software

In general terms, software is referred as an organized set of instructions which when executed by means of a given computing device delivers the desired result by considering various processes and functions. It is an organized set of instructions capable of accepting inputs, processes them and delivers the result in the form of functions and performs as expected by the user. Apart from this, it refers to the records (say software manuals) which helps and guides the users to efficiently deal with it. It is now-a-days delivered in the form of a package accumulating the design documents, source code, installations implementation manuals, operations system manuals.

Characteristics of Software

A software should be analyzed through various levels of perception. To do this, which it needs to be analyzed by its characteristics.

1. Customizable Software
2. "Software Doesn't Wear Out"
3. "Software is Developed or Engineered; it is not Manufactured in the Classical Sense"

Software Engineering

Software engineering is defined as establishment and application of sound engineering principles for obtaining an economically feasible and reliable software that can run efficiently on any real-time machine.

Q4. Write short notes on any two software applications.

Answer :

The two software applications are as follows,

(a) Application Software

Application software usually resides on a single system which is capable of satisfying only business requirements and involves management/technical decision making.

(b) Netsourcing

With an unpredictable growth of the internet, the software engineers are now forced to look forward for the development of simple as well as more sophisticated softwares so that, it is the end user who can take larger benefits from such applications, as internet in today's world, is not only acting like an engine but also as the major source for obtaining large volumes of data.

Q5. Explain software crisis.

Answer : Nov./Dec.-17(R15), Q1(b)

Software crisis refers to critical point which is faced by software developers at the time of software development. The crucial point may be due to low quality, high cost, incompatibility, presence of errors, low productivity, less efficiency in tools and methods adopted.

Q6. Distinguish between software products and software services.

Answer : (Model Paper-II, Q1(b) | Nov./Dec.-17(R15), Q1(a))

	Software Products		Software Services
1.	Software products are considered as the intellectual property of vendors.	1.	Software services are not the intellectual property of vendors.
2.	They offer application specific services which are common to all the users.	2.	They offer client-specific services.
3.	They are developed based on the efforts of vendor.	3.	They are developed based on the requirements of clients, cost and time involved.

Q7. What is legacy software? Explain.

(Model Paper-III, Q1(b) | Dec.-19(R16), Q1(a) | Nov./Dec.-16(R13), Q1(a))

Answer :

Legacy Software can be defined as an old software developed in the past. This software is still being used in the present era as it performs essential business activities. It may include procedure which are no longer relevant in the newer computing environment. As business requirements are dynamic, the legacy software system undergo continuous modifications so as to make the software adaptable to the new business requirements and to make it interoperable with the current computing environments.

Though, legacy software systems are becoming problematic in large organizations because of their high maintenance cost, they are still being used in the organizations due to the difficulties and risks encountered while replacing these systems with modern systems. Legacy systems are supportive to essential business activities and therefore they are considered as business critical system.

Q8. Give any three types of changes made to legacy system.

Answer :

The changes made to legacy system are as follows,

(i) **Making the Software Adaptable**

The software can meet the requirements of new computing environments, by making the software adaptable to the computing environment.

(ii) **Enhancing the Software**

The characteristic features of software needs to be enhanced so that the software can easily implement the new business requirements.

(iii) **Extending the Software**

The software needs to be extended so that it can achieve the interoperability feature.

Q9. List and define the various software myths.

Answer : Model Paper-II, Q1(a)

The three software myths are,

(i) **Management Myths**

These are the myths believed by software managers who are responsible for improving quality and controlling budgets.

(ii) **Customer Myths**

These are the myths believed by customers who can either be internal (technical group, marketing/sales department) or external to an organization.

(iii) **Practitioner's Myths**

Practitioner's myths are misconceptions believed by many software practitioners.

Q10. Discuss the objective of CASE.

Answer :

CASE stands for Computer Aided Software Engineering. It can be defined as a software or a technology, whose objectives are,

(a) To provide support to software process by implementing automation on the activities of software process like, design, programming etc.

(b) To give information regarding the development of software.

Q11. Discuss the major problems with the capability maturity model.

Answer :

CMM provides various advantages to the development of software process, even though some of the disadvantages of CMM model arise in the development of software process. They are as follows,

1. The CMM model does not allow risk analysis and management as the key process areas. Whereas, the risk management is a process that helps to identify the potential risks in the software development process.
2. The CMM model does not concentrate on product development, rather it concentrates on project management which does not allow an organization to utilize the technologies such as structured methods, prototyping and tools for static analysis.
3. The CMM model has complicated rules and procedures for small organizations.

The functionalities of this model are not defined. Only few organizations can use this model and on other hand they do not specify for which kind of organization this model will be suitable. Using this type of model leads to the drawback of the software development. It will be complicated to use this model in small organizations.

Q12. What are the fundamental activities of a software process?

Answer : March-17(R13), Q1(b)

The five generic process framework activities useful in developing several projects are given below,

1. **Communication**

This refers to a framework activity where, usually the end users (say customers) are communicated and their views related to the project are analyzed. Here, reports related to customer requirement specifications are developed.

2. **Planning**

In this framework activity, usually the entire work schedule which is going to be implemented in further stages, is prepared. Hence, various issues to be addressed during this framework are risks, requirement of resources, software schedules, important products to be developed etc.

3. **Modelling**

Once the developer is done with analyzing the customer requirement specifications, the third framework activity will be modelling. Here, usually UML diagrams are used to represent the project in the form of architecture. This helps to developers and the customer to gain an insight of the end product.

4. **Construction**

It is a combination of code generation and testing.

5. **Deployment**

In this framework activity, usually the developed project is delivered to the end users. They deploy the project and provide relative feedbacks to the developers.

Q13. List the sources of software standards.

Answer :

The sources of software standards include different organisations. They are,

1. ISO (International Standards Organization)
2. IEEE (Institute of Electrical and Electronic Engineers)
3. ANSI (American National Standards Institute)
4. DOD (U.S Department of Defense)
5. IEE (Institute of Electrical Engineers in UK) and BS (British Standard Institute)
6. OMG (Object Management Group).

Q14. What is waterfall model?

Answer :

Model Paper-III, Q1(a)

Waterfall model remains one of the oldest strategies ever applied, in the development of a software. It is also known as *classic life cycle model,* which divides the entire software development process into five main phases. Following is the diagrammatic representation of waterfall model,

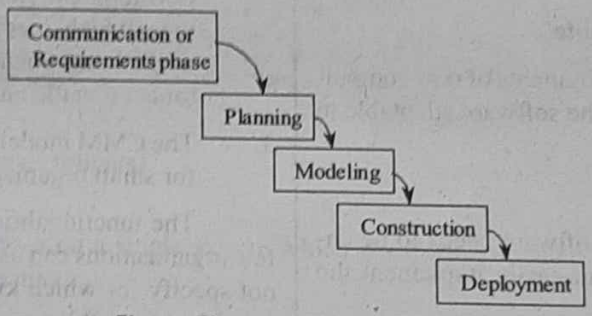


Figure: Phases of Waterfall Model

The waterfall model was proposed with feedback loops. However, most of the organizations avoid these loops. Thus this model is also called Linear Sequential Model.

Q15. What are the advantages of prototyping model over waterfall model?

Answer :

Model Paper-I, Q1(b)

The advantages of prototyping model over waterfall model are as follows,

(i) The Prototype Models give a prototype that can be used to illustrate input data formats, messages, reports and interactive dialogues for the customers. This is a valuable mechanism for explaining various processing options to the customer and for gaining a better understanding of the customer's needs. This is not possible in Waterfall Model.

(ii) The prototype explores technical issues in the proposed product. Often, a major design decision will depend on, say, the response time of a device controller or the efficiency of a sorting algorithm. In these cases, a prototype may be the best, or only, way to resolve the issue. The waterfall model does not explore the technical issues in the proposed product.

Q16. What are the merits of incremental model?

Answer :

(Nov./Dec.-18(R16), Q1(a) | March-17(R13), Q1(a))

The merits of incremental model are as follows,

1. When less number of people are involved in the project, incremental model is the correct choice.
2. With each increment, the technical risks involved in the project are reduced.
3. The customer can expect at least a core product in a short span of time from the project team.

Q17. Differentiate between waterfall and incremental process models.

Answer : May/June-12, Set-4, Q3(b)

	Waterfall Model		Incremental Model
1.	The waterfall model is called a linear sequential life cycle model as it develops the software sequentially.	1.	The incremental model is iterative. It develops the software in multiple iterations.
2.	In waterfall model, the software is developed in sequence and delivered at the end at once.	2.	In incremental model, the software is developed in increments and each phase is delivered separately at successive points of time.
3.	Requirements cannot be changed once fixed.	3.	Requirements can be changed after every increment.
4.	Risks are high in waterfall model and cannot be analyzed before the last phase.	4.	Risks are identified and solved after every iteration.
5.	Changing the requirements becomes costly as all the phases from the beginning have to be repeated.	5.	Changing the requirements in the incremental model is easy, as new features can be added after every iteration.

Q18. What are the advantages of unified process?

Answer : Nov./Dec.-16(R13), Q1(b)

The advantages of unified process are as follows,

1. It covers the complete software development life cycle.
2. Even though the unified process model came into existence after a true hardwork of over 20 years, it is available on internet in the form of an electronic guide (available to everyone located anywhere around the globe).
3. Most of the modern techniques and approaches use the unified process model as a set of guidelines.
4. The best practices for software development are supported by unified process model.
5. Unified process model does not result in a frozen product. Rather the product is ever evolving and is constantly maintained.
6. Its process architecture can be tailored as per requirement.
7. It encourages UML as the best process-oriented languages protocols.

Q19. List evolutionary process models.

Answer : May/June-19(R16), Q1(b)

Evolutionary Process Model

The evolutionary process model tends to develops a high quality software iteratively or incrementally. It is used to highlight the flexibility, extensibility and speed of development. There are two common evolutionary process models discussed below,

1. Prototyping model
2. Spiral model.

1. Prototyping Model

A prototype is a mock-up or model of a software product. In contrast to a simulation model, a prototype incorporates components of the actual product. Typically, a prototype exhibits limited functional capabilities, low reliability and inefficient performance.

2. Spiral Model

The spiral model for software engineering includes the features of classic life cycle and prototyping with an added advantage of element-risk analysis. It provides a framework for the design of software production process with due consideration of risk levels that may incur while designing. The spiral model can be used as a reference for choosing the final development model.

Q20. List the task regions in the spiral model.

Answer : Nov./Dec.-18(R16), Q1(b)

The task regions of the spiral model are as follows,

1. **Customer Communication**

   It is used for establishing communication among the customers.

2. **Planning**

   It carries out the different tasks of planning for defining the time line of resources as well as the other projects that are associated with these tasks.

3. **Risk Analysis**

   It carries out the tasks that are needed for calculating technical and management risks.

4. **Engineering**

   It carries out those tasks that are needed for developing various representations of an application.

5. **Construct and Release**

   It carries out those tasks that are essential for developing, testing, installing an application. Also, it offers certain tasks that support user assistance.

6. **Customer Evaluation**

   It initially collects the customer feedback and then perform the tasks depending upon the customer's priorities. Later on, implement these tasks during installation state.

PART-B ESSAY QUESTIONS WITH SOLUTIONS

1.1 INTRODUCTION TO SOFTWARE ENGINEERING

1.1.1 The Evolving Role of Software

Q21. Explain the evolving role of software.

Answer :

Model Paper-I, Q2(a)

Definitions of Software

In general terms, software can be defined as an organized set of instructions. These instructions deliver the desired result by considering various processes and functions.

It is an organized set of instructions capable of accepting inputs, processes them and deliver the result in the form of functions. Apart from this, it refers to the records (say software manuals) which help and guide the users to efficiently deal with it. It is now-a-days delivered in the form of a package offering the design documents, source code, installations implementation manuals, operations system manuals.

Software Evolution

Modern software act as a vehicle to transmit a product.

Role of Software as a Vehicle in Product Transmission

Software as a vehicle has certain responsibilities. They are,

(i) Controlling various operating systems.

(ii) Creating and managing programs such as, software tools, software environments etc.

(iii) Delivering information.

Role of Software in Delivery of a Product

As a product, software performs the following tasks,

(i) Transmitting computer data which is a part of computer hardware.

(ii) Delivering the computer data that is a part of computer networks accessible by a local hardware.

An interesting feature of computer software is its location independence as it is basically used to transform information. Transforming information involves,

(a) Creation of information

(b) Management of information

(c) Modification of information

(d) Display of information.

With the above features, it can be said that today's software gives more emphasis on the delivery of information.

Over the period of time, software has made a serious impact in the field of software.

Impact of Software in the Field of Computers

(i) Performance of hardware has improved.

(ii) Memory size of computer has increased.

(iii) Improvement in storage capacity of the computer.

(iv) Availability of latest and exciting input and output devices.

(v) Development of different computer architectures.

The flexibility of software systems is one of the reasons of software being incorporated in large and complex systems. Making changes to the hardware is expensive once it is manufactured. However, changes to the software can be made at any time during or after the system development.

Few software systems are developed as completely new systems and their updates and maintenance are continuous. *Software evolution* is an evolutionary process where the software is periodically changed over its life time in response to the changing requirements.

The evolutionary process is illustrated in below figure.

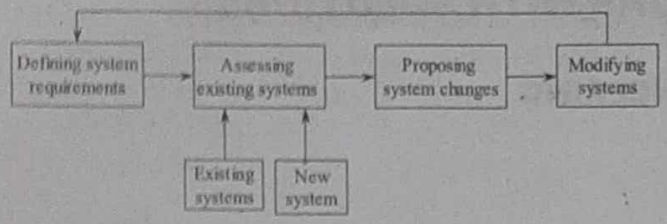


Figure: Software Evolutionary Process

The evolution pattern of a software consists of the following parallel activities,

(i) Where is the evolution pattern done?

The where segment determines the market place from which the client request is obtained.

(ii) Why is it done?

The why segment determines the reasons for errors, failures and the necessary for improving and recovery.

(iii) What are the requirements?

The what segment specifies the design document test plan, interface, user manual for the project/process.

(iv) When it is performed?

The when segment determines at which point of time necessary tests like component test, cost estimate, acceptance test are to be carried out.

(v) How it is performed?

The how segment specifies the method of rewriting, redesigning, redrawing and coding the project.

(vi) By whom it is performed?

The 'by whom' segment specifies the person who is involved in the particular process/project or part of it. The person can be engineer/client/user.

1.1.2 Changing Nature of Software

Q22. Define the term Software. Describe its various characteristics. **March-17(R13), Q2(a)**

OR

Explain various characteristics of software.

Answer : **(May/June-12, Set-2, Q3(b) | Dec.-11, Set-1, Q2(b))**

Software

For answer refer Unit-I, Q21, Topic: Definitions of Software.

Characteristics of Software

A software should be analyzed through various levels of perception. To do this, which it needs to be analyzed by its characteristics.

Following are the attributes of a good software,

1. Customizable Software

In order to analyze the customizable software, the user should have a clear distinction between hardware and software manufacturing procedures. For instance, consider the hardware manufacturing procedure in which certain digital circuitry is to be built. The process starts by drawing neat sketches and analyzing whether the given design is satisfying the intended specifications or not. Once these things are known, the user begin acquiring the required digital circuitry i.e., gates, capacitors etc. Finally testing is performed to complete this process. While closely analyzing the above mentioned scenario it can be concluded that this process contains many reusable facts and very less new facts. But this is not the case during the software development. Here, the given software is first required to be built and later implemented, so as to determine its reusability values. It is the recent trend in software engineering process to rely on certain reusable components along with few new components in framing the entire software.

2. "Software Doesn't Wear Out"

In order to analyze this characteristic, consider development life cycle of hardware and software. The best mode of comparison is to take various aspects of the graphs during their life cycle (hardware and software) to be perfectly analyzed graph describing various consequences related to the hardware is given below,

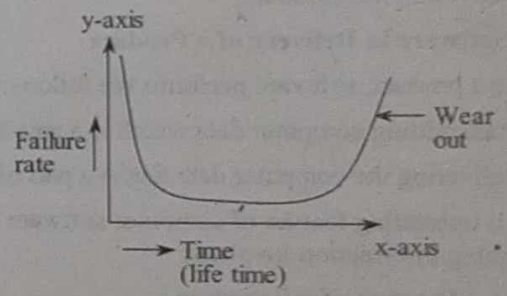


Figure (i): Graph Depicting the Failure Curve for Hardware

Analyzing the curve concludes that during the initial days of manufacturing, the hardware suffers from severe defects. When these failures are eliminated, the curve attains a steady success rate. This success rate does not last longer and many external entities such as, vibration, environmental effects, temperature changes, dust, etc., act as barrier to its prolonged success causing it to wear out steadily. Now, analyze the same aspects of the software in the following curve.

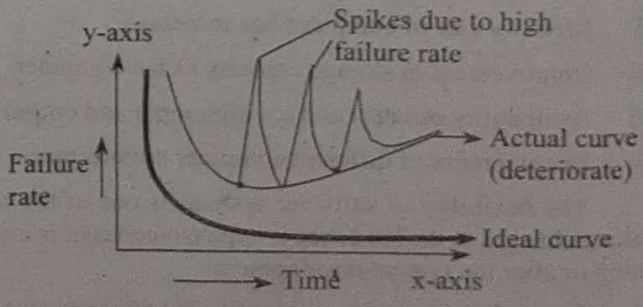


Figure (ii): Software Failure Curve

In the ideal curve, software initially suffers from unexpected defects. But as software does not get affected with climatic or other environmental changes as that of hardware, it comes down to a steady success rate by suitably correcting the errors. Till this stage, the ideal curve remains analogous to the actual curve. Whenever the actual curve is considered, it declines from its initial failure rate and comes down steadily to a particular point, where it demands certain changes to be applied and leads to the introduction of other defects. Hence, a spike is observed in the curve. Now, efforts are applied to nullify these defects and hence, the spike comes down to a point where it demands change and again the same process is repeated which gives rise to other spikes. While change is made to the curve, the main cause of defect remains due to the changes made to the software to a certain extent and due to the side effects of the software to a larger extent.

Hence, with reference to above illustrations it can be concluded that, the "software does not wear out rather it deteriorates".

3. "Software is Developed or Engineered; it is not Manufactured in the Classical Sense"

Development of a software and other hardware involves manpower, but the quantity and the way of approach remain significantly different. In both the manufacturing processes, one will be left out with certain end products, but, the efforts applied in them remain different. Finally, both the activities are initiated with a determination of building high quality of products, but their quality maintaining activities differ. Hence, with these specifications it can be conclude that, "software is developed or engineered, it is not manufactured in the classical sense".

Q23. Explain the categories of software.

OR

Discuss about the changing nature of software.

Nov./Dec.-16(R13), Q2(a)

OR

Elaborate on the changing nature of software in detail.

Answer : **March-17(R13), Q2(b)**

Following are the broad categories of software, often referred as applications of software which describe its changing nature.

1. **Engineering and Scientific Software**

Software is being developed to ease the growth in engineering and scientific areas. Engineering and scientific software development requires large area of information to be covered. For these fields, applications have been developed which cover areas from astronomy to volcanology, molecular biology to automated manufacturing and also from automotive stress analysis to space shuttle orbital dynamics. Now, advanced applications like CAD/CAM, SPSS, MATLABS, etc., have been developed.

2. **Web-based Software**

The software packages have been updated and they can now transfer information on the web. These packages are used to develop highly sophisticated web based-software that simultaneously eases its usage and also ensures a safe transfer of data over network. Web-based software or applications can be developed, by using languages or packages like JAVA, C++, Vb.net, CGI, Javascript, HTML, DHTML etc.

3. **Embedded Software**

Embedded software is developed to control the products under consumer and industrial markets. This software resides in the Read Only Memory (ROM) of the product. This software is developed to perform the limited and the specialized functions of the product.

Example

Keypad control of an air cooler, button control of a washing machine and so on. These software also provide significant functional and control capabilities of the product.

4. **Artificial Intelligence Software**

This type of software makes use of non-numerical algorithms to solve complex problems, they are not adaptable to computation or direct (straightforward) analysis. One of the most active artificial intelligence areas is the, "Expert Systems". Other application areas for AI software include pattern recognition, theorem proving and game playing.

5. **System Software**

System Software refers to a piece of software capable of providing services to other applications. Good examples of such software are categorized in two sets. One of the sets includes file management utilities, compilers, editors etc., and other set includes drivers, operating system, networking software etc. The system software belonging to category-I can easily be applied to quite complex and deterministic applications. On the other hand, category-II can easily be applied to highly non-deterministic applications.

6. **Application Software**

Application software usually resides on a single system capable of satisfying only business requirements and involves management/technical decision making.

7. **Netsourcing**

The growth of internet forced the software engineers to look forward for the development of simple as well as more sophisticated software. This is because, internet in today's world, is not only acting like an engine but also as the major source for obtaining large volumes of data.

8. **Universal Computing**

In today's world, the major source of data communication is by means of wireless circuits. It is expected that, in the future such circuits can be applied in developing distributed systems. Hence, such applications will also remain a challenge to the software engineers.

9. **Open Source**

Open source is also an expected future development in software where a given software is made available to all users irrespective of their profession. They can modify it as per their requirements. In this case, the engineers must strive to make the software easily understandable and compatible so that, it can be easily moulded.

Q24. What is a legacy software? Explain.

Nov./Dec.-17(R15), Q2(a)

OR

What is legacy software? Explain briefly its impact in software engineering.

Nov./Dec.-18(R16), Q2(a)

OR

What type of changes is made to legacy systems if it exhibits poor quality?

Answer :

Legacy Software

Legacy Software can be defined as an old software developed in the past. This software is still being used in the present era as it performs essential business activities. It may include procedures which are no longer relevant in the newer computing environment. As business requirements are dynamic, the legacy software system undergo continuous modifications. These modifications make the software adaptable to the new busines requirements and to make it interoperable with the mputing environments.

ough, legacy software systems are becoming problem- rge organizations because of their high maintenance ey are still being used in the organizations due to the ılties and risks encountered while replacing these systems ıtı modern systems. Legacy systems support essential business activities and therefore they are considered as business critical systems.

Types of Changes Made to Legacy Systems

Re-engineering must be carried out on legacy systems so as to make these systems capable of handling the modern business requirements. This can be done by making the following significant changes to the legacy systems,

(i) **Making the Software Adaptable**

The software can meet the requirements of new computing environments, by making the software adaptable to the computing environment.

(ii) **Enhancing the Software**

The characteristic features of software needs to be enhanced so that the software can easily implement the new business requirements.

(iii) **Extending the Software**

The software needs to be extended so that it can achieve the interoperability feature.

(iv) **Redesigning the Software**

The software needs to be redesigned by making changes to the existing software so as to make the software operable within a network environment.

Q25. Explain, why legacy systems evolve as time passes.

Answer :

Reasons for Legacy Software Evolution

Software evolution is an evolutionary process where software is continuously changed over its lifetime in response to changing requirements. This process is carried by 'change' and is performed when,

(i) The errors identified are corrected.

(ii) The software becomes adaptable to new computing environment.

(iii) The application re-engineering is performed.

The following important laws are defined, so as to get a brief description about the unified theory for software evolution,

1. **Continuing Change Law**

The E-type system software that evolved overtime must be continuously adapted so as to make them implement in real world computing environment. The system is capable of meeting the customers requirements and satisfying them to the maximum extent only if the system undergoes a continuous modification.

2. **Increasing Complexity Law**

During the evolution process of an E-type system software, the level of complexity increases when no measures are used for reducing or maintaining it.

3. **Self Regulation Law**

The process of evolving an E-type system is self regulating with respect to the product distribution and process measures. Its value is approximately equal to the normal value.

4. **Conservation of Organizational Stability Law**

The average activity rate is constant over the lifetime of a product when an E-type system is being evolved.

5. **Conservation of Familiarity Law**

When an E-type system is being evolved, it is necessary to ensure that all the members (i.e., developers, sales personnel, users) responsible for performing the system evolution must maintain the entire information about the evolution process. They should also track the behavior using which a satisfactory evolution is achieved.

6. **Continuing Growth Law**

The functionality of an E-type system must increase continuously over the lifetime of the system in response to the customers requirements.

7. **Declining Quality Law**

The quality of an E-type system will be degraded if the software system doesn't undergo continuous modification. Due to this, the system fails to meet the requirements of new computing environments.

8. **Feedback System Law**

The evolution process of an E-type system comprises of feedback systems with multilevel, multiloop, multiagent feature. It is necessary to have such feedback systems for improving the performance of the software.

1.1.3 Software Myths

Q26. Explain the various software myths.

(Model Paper-II, Q2(a) | Nov./Dec.-17(R15), Q3(a))

OR

What are various software myths prevalent in industry? Why do the stakeholders believe them? Contradict the myths with reality.

Dec.-19(R16), Q3(b)

OR

What are various myths about software?

March-17(R13), Q3(b)

OR

Discuss in brief about different software myths and their consequences. **Nov./Dec.-16(R13), Q3(a)**

OR

Discuss managers myths about software development and their effect on the practitioners performance as well as on overall outcome.

(Refer Only Topic: Management Myths)

Answer : **May/June-19(R16), Q2**

Software Myths

Software Myths are the belief's that software managers, customers and software practitioners have about software and the process used for it. These beliefs are continuing over several years of programming culture. Today, software myths are considered as a misconception, which when followed results in disastrous effect.

1. **Management Myths**

These are the myths believed by software managers who are responsible for improving quality and controlling budgets.

Myth (i): A book of standards and procedures that defines the way of developing a software is sufficient to meet the requirements of the people.

Reality

Though there exists a book of standards, it cannot be used because of the following reasons,

(i) Software practitioners do not have the knowledge about the existence of the book.

(ii) It does not take into consideration the modern software engineering practices.

(iii) The book of standards is neither complete nor adaptable.

(iv) It is not possible to reduce the delivering time while concentrating on the quality factor.

Myth (ii): It is possible to add programmers at the later stages of software development life cycle.

Reality

Adding new programmers at later stages will not reduce the amount of time spent on software development. That is because the experienced programmers need to spend their time training the new programmers. This problem can be solved only if the organization prepares a plan, which is executed in a well coordinated fashion.

Myth (iii): Software projects development responsibility is handed over to a third party.

Reality

If software projects cannot be developed within the organization due to lack of understanding of how the software project is managed and controlled, then it will be difficult for the organization to understand the projects developed by a third party.

2. **Customer Myths**

These are the myths believed by customers internal (technical group, marketing/sales department) or external to an organization.

Myth (i): The writing of programs can be started by considering only a general statement of objective. The other details can be filled later.

Reality

General objective statement that conveys incorrect meaning and that misleads the customers can lead to a disaster. If there is a continuous interaction between a customer and a developer then only it is possible to generate requirements that are unambiguous and understandable.

Myth (ii): Software is flexible therefore any changes to the project requirements can be easily accommodated.

Reality

Software project requirements change very frequently whose impact can vary depending on the time at which they are introduced. If changes occur in the early stages of the software development then the cost is less. Whereas, if they occur at the later stages, then the cost is high.

3. Practitioner's Myths

Practitioner's myths are misconceptions believed by many software practitioners.

Myth (i): The job of software practitioners is done when they complete writing a program and executing it in the working environment.

Reality

The actual work or effort of the software practitioner does not stop once the program gets executed but instead it initiates when the software is delivered to the customer.

Myth (ii): The software project quality cannot be assessed until the program is executed.

Reality

Formal Technical Review (FTR) is considered as the best and effective SQA technique that can be applied from the inception of a project. Software reviews act as quality filters and are considered to be effective when compared to testing mechanisms.

Myth (iii): Working program is the only deliverable work product for a successful project.

Reality

Working program is not the complete output but instead it is a part of software configuration which in turn comprises of multiple elements. Documentation is not only considered as a basic step for performing successful engineering but also as a guidance for providing software support.

Myth (iv): The documentation generated from software engineering consists of massive and irrelevant volume of information. Creation of these unnecessary documents slows down the software development process.

Reality

Software engineering refers to a process of creating a better quality product but not creating documents. It decreases the amount of rework to be performed thus speeding up the delivery time.

1.2 A GENERIC VIEW OF PROCESS

1.2.1 Software Engineering – A Layered Technology

Q27. What do you mean by software engineering? Explain the software engineering layers.

OR

What are the advantages of layered technology?

Answer : May-18(R15), Q2(a)

Software Engineering

Software engineering is defined as establishment and application of sound engineering principles for obtaining an economically feasible and reliable software that can run efficiently on any real-time machine.

Software Engineering Layers

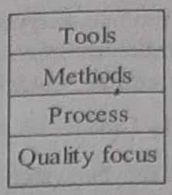


Figure: Layers of Software Engineering

The above figure depicts software engineering layers. The bottom layer claims the *quality* which is extremely essential for any software product. In order to achieve that, the users usually rely on various quality management issues, six sigma etc. With this, software development can be matured at every level and the end product is of very high quality in all aspects.

From the *process layer*, the main software engineering activity begins. It forms the main source to adhere technology as well as the on-time delivery of the product. Process exerts impact on the software development activity in two levels, i.e., in designing the framework and at management level.

Process is effectively used in designing a framework for entire software engineering activity. This has got a crucial role to play in transmitting the software engineering technology.

At management level, usually the software engineering process is applied in regulating and controlling the activities of software projects. It also defines a platform with which, the user can implement technical activities during software project development, achieve the expected goals, ensure a high quality in the developed products etc.

The third layer i.e., *methods* specifies a criterion in construction of high quality software. The methods form the provision for requirement analysis, design models, testing software etc.

The top most layer encompass *tools*. Process and methods often depend on these tools for their implementation. Tools form the major source of development of computer aided software engineering, usually accomplished by integrating these tools, so that the associated information can be acquired by other tools.

1.2.2 A Process Framework

Q28. What do you mean by process framework? Explain the five generic process framework activities. Model Paper-III, Q2(a)

OR

Explain the software process framework. Nov./Dec.-17(R15), Q2(b)

OR

What are the five generic process framework activities? Explain.

Answer : Nov.-15(R13), Q3(a)

Diagrammatic representation of software process framework is given in figure,

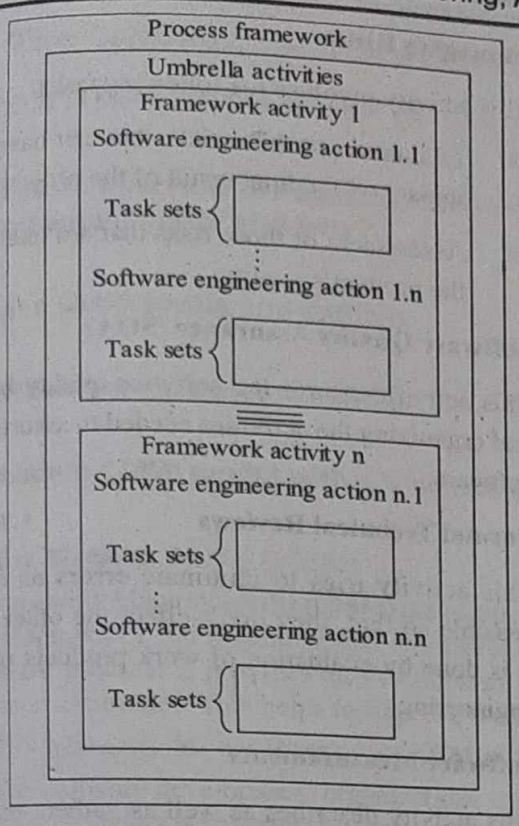


Figure: Process Single Word

Process framework plays a major role in every software development activity as it forms a base in initiating the development process. It is used to estimate certain framework activities which are applied in every developmental activity, irrespective of its size. The process framework includes several *umbrella activities*, which are useful throughout the software development process. Next to umbrella activity is a *framework activity* which includes definite number of software engineering actions applicable in driving a specific software engineering applications. Each of these engineering actions corresponds to the *task set* which encompasses a list of actions to be applied. Each engineering action can perform a constituent activity which forms an essential part of the specific software application being developed.

The five generic process framework activities useful in developing several projects are given below,

1. **Communication**

   This refers to a framework activity where, usually the end users (say customers) are communicated and their views related to the project are analyzed. Here, reports related to customer requirement specifications are developed.

2. **Planning**

   In this framework activity, usually the entire work schedule which is going to be implemented in further stages, is prepared. Hence, various issues to be addressed during this framework are risks, requirement of resources, software schedules, important products to be developed etc.

3. **Modelling**

   Once the developer is done with analyzing the customer requirement specifications, the third framework activity will be modelling. Here, usually UML diagrams are used to represent the project in the form of architecture. This helps developers and the customer to gain an insight of the end product.

4. **Construction**

   It is a combination of code generation and testing.

5. **Deployment**

   In this framework activity, usually the developed project is delivered to the end users. They deploy the project and provide relative feedbacks to the developers.

Q29. Define a task set. List the entities of task set for the following,

(a) Relatively small and simple projects

(b) Large and complex software projects.

Answer :

Task Set

A task set is usually a collection of tasks to be applied in order to obtain the required output from the software engineering action.

For instance, if the communication activity is being considered then, software engineering action would be a collection or documentation of customer requirement specification. All tasks of a task set, prepared for such action, would definitely favour in preparing the customer requirement specification only.

(a) Relatively Small and Simple Projects

A task set may differ from a small-simple project to a large-complex project. Following is a task set for small-simple project, if customer requirement specification is the software engineering action.

- ❖ List all the customers (to whom the project is to be delivered).
- ❖ Gather these customers along with the developers.
- ❖ Invite each of these customers to reveal their vision of project (the way the product should be developed)
- ❖ Develop a list of requirements.
- ❖ Sort these requirements according to their priorities.
- ❖ Mark the uncertain areas.

(b) Large and Complex Software Projects

If the project is large-complex, and the customer requirements specifications is considered to be the software engineering action then the task set would include the following tasks,

- ❖ List all the customers.
- ❖ Instead of gathering them together, invite them separately to grasp their vision of project.
- ❖ Build the requirement list for each user.
- ❖ Refine these requirements.
- ❖ After refining, suitably develop a final list of requirements.
- ❖ Provide this list with suitable priorities depending on the quality of deployment.
- ❖ Sort and gather them sequentially, so that they can be applied accordingly.
- ❖ Prepare a list of constraints which are probable when the product is being deployed.
- ❖ Finally end up the session by finding the methods which remain useful in validating systems.

Q30. "Any software process framework incorporates a set of umbrella activities". Discuss them briefly.

Answer :

There are eight umbrella activities of a software process framework. They are,

(i) Tracking and controlling software project

(ii) Managing risks

(iii) Software Quality Assurance (SQA)

(iv) Formal technical reviews

(v) Software measurements

(vi) Software Configuration Management (SCM)

(vii) Managing reusability factor

(viii) Preparing and producing software work products.

(i) Tracking and Controlling Software Project

The first umbrella activity is tracking controlling software project. This activity allows the software team to perform the following tasks.

- ❖ Assessing of progress of the project against the plan of the project.
- ❖ Maintaining the schedule of the project by taking appropriate action based on the above assessment.

(ii) Managing Risks

This activity involves the following tasks,

- ❖ Evaluation of those risks that can have a serious impact on the final result of the project.
- ❖ Assessment of those risks that are likely to effect the product's quality.

(iii) Software Quality Assurance (SQA)

This activity ensures the software quality by defining and organizing the activities needed to assure quality of software.

(iv) Formal Technical Reviews

This activity tries to eliminate errors as quickly as possible so that, they don't effect the other activities. It is done by evaluation of work products of software engineering.

(v) Software Measurements

This activity describes as well as gathers measures of process, product and project through which software team can deliver a software fulfills the needs of the customer.

(vi) Software Configuration Management (SCM)

SCM is also referred as Change Management (CM). During the software engineering process, SCM defines a set of activities for managing the changes made to the software components.

(vii) Managing Reusability Factor

This activity is incorporated by the software process framework because of the following reasons.

- ❖ It defines the basic criteria for reusing a work product. Other than this, it also defines a criteria for software components reuse.
- ❖ It also obtains reusable components by developing the desired methods.

(viii) Preparing and Producing Software Work Products

The last and final activity is the preparation and production of work product. It includes certain activities that are needed for the creation of the following work products,

(a) Models

(b) Logs

(c) Documents

(d) Lists and forms.

1.2.3 The Capability Maturity Model Integration (CMMI)

Q31. What is CMM? Discuss how various maturity levels of CMM can be measured. Model Paper-I, Q2(b)

OR

Give an overview of capability Maturity Model Integration. Which level of organizations as a customer you would prefer and why? Dec.-19(R16), Q3(a)

OR

Give CMMI levels and explain. May-18(R15), Q2(b)

OR

Write detailed notes on CMMI. Nov./Dec.-18(R16), Q3(b)

OR

Explain CMMI model with a neat sketch. Nov./Dec.-16(R13), Q3(b)

Answer :

Capability Maturity Model (CMM)

Capability Maturity Model (CMM) is a maturity framework strategy that focuses on continuously improving the management and development of the organizational workforce. CMM provides the organization with the basic requirements for building the software process. It provides an evolutionary path from an inconsistent organizational practices, to a highly disciplined developmental practice. This helps to improve the knowledge, skills and development of the software process.

Capability Maturity Model Integration (CMMI)

The software development organizations can be ranked depending on the quality of products they develop using a meta-model. This meta data is governed by certain set of systems and software engineering capabilities. These are needed to be satisfied by the organizations, as they attain various levels of capability as well as maturity. Hence, to remain on this track, each organization is required to develop a process model, based the guidelines of capability maturity model integration, The model is given below,

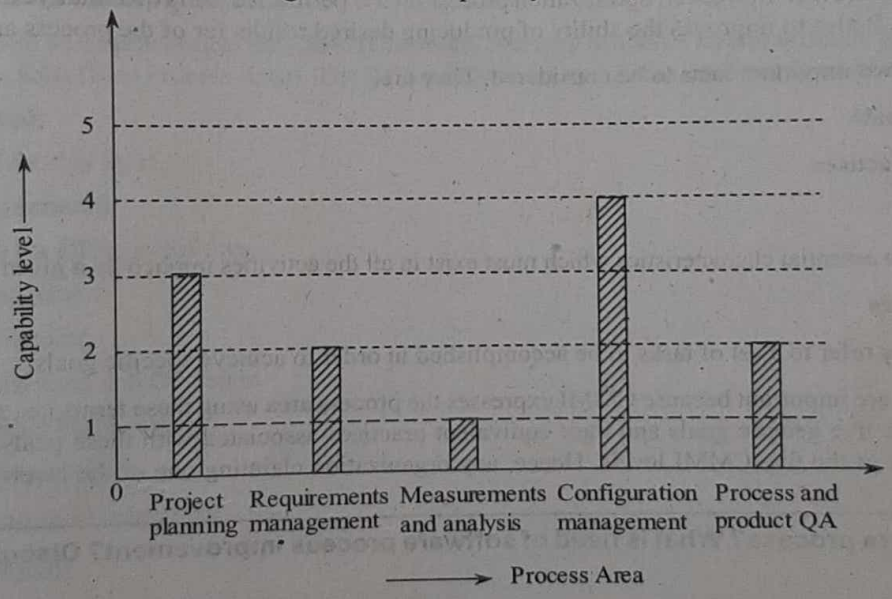


Figure: Graph Depicting the Process Area Capability Scenario

The above figure is also referred as a *continuous model*. Here, the process area is plotted against the standard levels, ranging from 1 to 5. However level '0' is also considered to represent the lowest of all. Each level and their equivalent values is expressed below. As a customer, it is preferable that an organization belongs to CMMI level 5 which indicates that it is successful in terms of providing services and satisfying customers.

Level 0 : Incomplete

At this level, there are two possibilities.

(i) The process area (along x-axis) is not performed.

(ii) The process area has not achieved the targets set by CMMI for level 1's capability.

Level 1 : Performed

(i) The tasks specified by the CMMI at process level have been achieved.

(ii) The work tasks required in developing a given work product are set.

Level 2 : Managed

(i) All the criteria defined at CMMI level 1 are met.

(ii) The work related to the process area is up-to-date with the expectations of organization.

(iii) The developers involved in the production have access to all the available resources for completing their tasks.

(iv) All the specimens are subjected to the process of project development whenever necessary.

(v) The tasks as well as products (under development) are regularly being monitored, controlled and reviewed.

(vi) The product is executed to ascertain that, it is functioning as per the expectation.

Level 3 : Defined

Entire conditions of level 2 are met. Apart from this, the process is customized according to the organization's set of standard processes based on the guidelines of the organization. This helps the process assets in terms of work products, measures and other process improvement information.

Level 4 : Quantitatively Managed

Entire consequences of level 3 are met. Also, by means of a quantitative assessment the process area is improved and controlled. Moreover, the establishment of the quantitative objectives for quality and process performance is also done and used as a criteria in process management.

Level 5 : Optimized

All level 4, targets are met. Moreover, optimization process area is performed using quantitative means to satisfy the varying customer requirements. It also to improves the ability of producing desired results for of the process area under consideration.

Here, there are two important facts to be considered. They are,

(a) Specific goals

(b) Specific practices.

(a) Specific Goals

These refer to the essential characteristics which must exist in all the activities implied by a given process area.

(b) Specific Practices

Specific practices refer to a set of tasks to be accomplished in order to achieve specific goals.

These two terms are important because CMMI expresses the process area using these terms i.e., specific goals and specific practices. Also there are five generic goals and their equivalent practices associated with these goals. Basically, these generic goals correspond to one of the five CMMI levels. Hence, any organization claiming one of the levels of CMMI should satisfy these generic goals.

Q32. What is software process? What is need of software process improvement? Discuss capability maturity models.

Answer :

May/June-19(R16), Q3

Software Process

A software process consists of a set of activities along with the ordering constraints, which specifies the proper functioning of these activities for generating the desired output (or) a software process refers to a process that involves various issues related to technical and management aspects of software development.

Software Process Improvement

For answer refer Unit-I, Q36.

CMM

For answer refer Unit-I, Q31.

Q33. Discuss various process maturity levels. Also discuss various KPAs that must be achieved in each level. Model Paper-III, Q3(b)

OR

Explain the process areas required to achieve various maturity levels in CMMI. Dec.-11, Set-3, Q3

(Refer Only Topic: Various KPA Defined in Each Level of Maturity)

OR

Explain different levels of capability maturity model and list the KPA's of each level.

(Refer Only Topic: Various KPA Defined in Each Level of Maturity)

Answer : Nov.-15(R13), Q3(b)

Process Maturity Levels

For answer refer Unit-I, Q31.

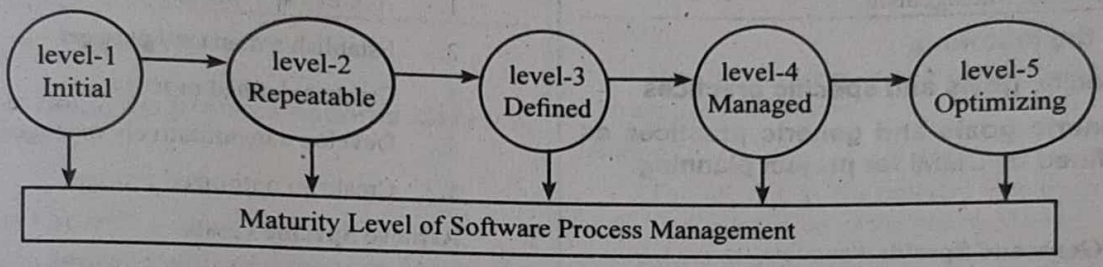


Figure: Levels of Maturity

Various KPA Defined in Each Level of Maturity

Software management process measures the CMM (Capability Maturity Model) with five different levels of maturity, which can be obtained by using a KPA (Key Process Area). The KPA performs its operations on every level of maturity as follows,

Level 1 of Maturity (Initial)

KPA is not defined for this level.

Level 2 of Maturity (Repeatable)

This level specifies six KPAs as follows,

(i) Requirements Management
(ii) Software Project Planning
(iii) Software Project Tracking and Oversight
(iv) Software Sub-contract Management
(v) Software Quality Assurance
(vi) Software Configuration Management.

Level-3 of Maturity (Defined)

This level defines seven KPA's as follows,

(i) Organization Process Focus
(ii) Organizational Process Definition
(iii) Training Program
(iv) Integrated Software Management
(v) Software Product Engineering
(vi) Intergroup Coordination
(vii) Peer Reviews.

Level-4 of Maturity (Managed)

This level follows the basic two principles,

(a) Analysing and grouping of data

(b) Software quality management.

The two KPA's defined by this level are,

(i) Quantitative Process Management

(ii) Software Quality Management.

Level-5 of Maturity (CMM Optimizing Process)

This level defines three KPA's as follows,

(i) Prevention of Defects

(ii) Technology Change Management

(iii) Process Change Management.

Q34. Explain the following,

(a) Specific goals and specific practices

(b) Generic goals and generic practices as defined by CMMI for project planning.

Answer :

(a) Specific Goals and Specific Practices

The process areas are defined by Capability Maturity Model Integration (CMMI) on the basis of certain Specific Goals (SG) and their related Specific Practices (SP).

Consider project planning which is one of the eight process areas. For this process area, there are three specific goals with different specific practices under each goal.

The specific goals are,

1. Project estimation
2. Project plan development
3. Commitment to the plan.

Specific practices for each goal are as follows,

1. Project Estimation

(a) Estimation of the project scope.

(b) Estimation of task attributes and work product.

(c) Definition of life cycle of the project.

(d) Compute the estimation of effort and cost of the project.

2. Project Plan Development

(a) Creation of budget and schedule of the project.

(b) Identification of risks of the project.

(c) Generation of a data management plan.

(d) Creation of project resources plan.

(e) Creation of a plan that involves skills and knowledge needed for a project.

(f) Development of a stakeholder involvement plan.

(g) Generation of plan for a project.

3. Commitment to the Plan

(a) Review of those plans that may affect the project.

(b) Reconciliation of resource levels and work of a project.

(c) Acquire commitment towards the plan of the project.

(b) Generic Goals and Generic Practices

CMMI also defines process areas in terms of Generic Goals (GG) and Generic Practices (GP).

The process area called Project Planning consists of five generic goals with various generic practices under each goal.

The five generic goals of project planning are,

1. Achieve specific goals
2. Establish a managed process
3. Create a defined process.
4. Develop a quantitatively managed process
5. Create an optimized process.

1. Achieve Specific Goals

This generic goal of project planning includes only one generic practice. That is to implement base practices on project planning process area.

2. Establish a Managed Process

In this generic goal, there are many generic practices. They are,

(a) Creation of organisationally defined policy for the work involved in planning of a project.

(b) Ensurance of providing resources that are required for project planning.

(c) Assign responsibilities to each and every person working on a project.

(d) Provide training to new people involved in the project.

(e) Configuration management.

(f) Identify stakeholders that are needed for project planning process area and include them in it.

(g) Monitoring and controlling of work products.

(h) Evaluation of work products in an objective way so that they adhere to the description of the process.

(i) Allow high level management to review the status of the project.

3. Create a Defined Process

This generic goal includes two generic practices.

(a) The generic goal itself.

(b) Gather information regarding improvement in the software process.

4. **Develop a Quantitatively Managed Process**

Generic practices of this goal are as follows,

(a) Generate quantitative objectives for the quality and performance of the process.

(b) Stabilize the performance of the subprocess.

5. **Create an Optimized Process**

This generic goal involves the following practices,

(a) To make sure that there is a continuous improvement in the process by satisfying the needs of the customer through quantitative means.

(b) Improve the effectiveness of project planning process area by identifying root causes of problems.

1.2.4 Process Patterns

Q35. Explain in detail the process patterns. Give few examples of it.

Answer :

Software Process Patterns

A software process pattern usually refers to the framework comprising of several tasks, actions, activities to be performed and other essential activities involved in software development. A software team considers many of these patterns throughout the software development process. A pattern refers to the entire software development process, or it can also refer to any one of the fundamental activities in the whole development process. Following are certain essential aspects of a process pattern,

Pattern Name

Initially, almost every pattern is assigned with a meaningful name. The name usually reflects the kind of action associated with that pattern.

Intent of Pattern

Intent of pattern usually defines its purpose. Hence, a text describing the purpose of the pattern is assigned to it. Here, it all depends on the way the developer elucidates the text. Also, he can account for several diagrams along with the text to retain its granularity.

Pattern Type

A pattern can be one amongst the following three types.

(a) **Phase Patterns**

Here, usually large sets of framework activities essential to the overall success of the process are specified. It has to be remembered that, these activities are associated with software development process.

(b) **Task Patterns**

Here, usually the software engineering action and/or work tasks are defined. These actions form the constituent activities associated with the given software process and also they a frame crucial aspects to the success of entire software development activity.

(c) **Stage Patterns**

Stage patterns usually signify a single framework of activity. But, the software process pattern in this regard require various work patterns in order to implement the framework activity.

Initial Context

In this case, the conditions or states during which the given patterns are applicable are specified. Hence, following are the questions for which suitable answers are determined initially.

- What tasks are already performed by the whole organization or various team members?
- What is the available software engineering information?
- What is the entry state for the available processes?

Problem

Here, the problem specifications to which pattern needs to be applied is discussed.

Solution

The solution to this problem is to describe the method for the implementation of the pattern. Hence, in this case the modification of initial process states as result of the initiaton of the pattern is depicted. Also the information of software engineering which eventually gets transformed in effect to the application of the pattern is also discussed.

Resulting Context

The situation which is prevailing on successful application of the pattern is addressed here. Hence, in this case we need to answer the following questions,

(a) What activities has taken place by the organization and the team members?

(b) What is the terminating state of the process?

(c) What is the current software engineering information available?

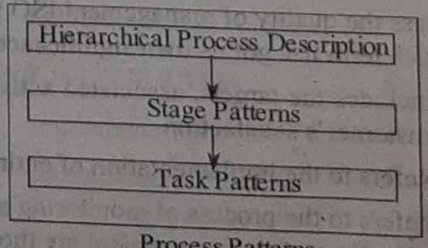
Related Patterns

All the probable patterns which can be associated with the current pattern list are extracted and are framed in the form of a hierarchical tree or expressed diagrammatically.

Known Uses Examples

The conditions during which the patterns can be applied are specified.

Finally, a small pictorial representation of a process pattern is generated as given below,



Process Patterns

Figure

1.2.5 Process Assessment

Q36. What is software process assessment? Discuss different approaches to it.

Model Paper-II, Q2(b)

Answer :

Application of process patterns to the software project under development doesnot ensure that the software is going to satisfy all the essentials (i.e., on-time delivery, customer satisfaction, high quality values etc.) Hence, in order to achieve this, the software patterns should be collaborated with highly valued software engineering practices. Moreover, the entire process should be sufficiently assessed as required. Following figure depicts the software process and methods that are useful during process assessment and improvement.

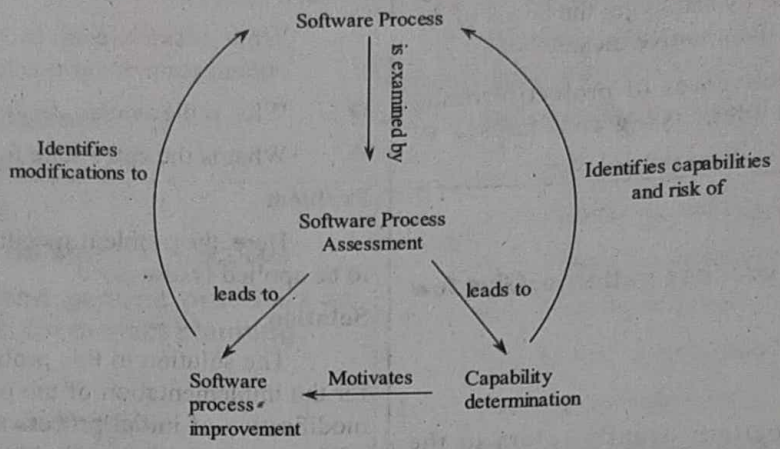


Figure: Software Process and Methods Applied for Assessment and Improvement

Following are the techniques for software process assessment.

Standard CMMI Assessment Method for Process Improvement (SCAMPI)

Five important steps which form the basis for software process assessment are as follows,

1. Initiating
2. Diagnosing
3. Establishing
4. Acting and
5. Learning.

In this case, SCAMPI usually relies on SEI CMMI for the requirement of software process assessment.

CMM-based Appraisal for Internal Process Improvement (CBA IPI)

In this technique, usually the granularity or maturity of the product organization is assessed using the SEI CMMI.

SPICE (ISO/1EC15504)

This standard remains effective in deriving software process assessment techniques.

ISO 9001 : 2000

Any software industry, can adopt the ISO 9001 : 2000 standard to reach quality peaks in terms of its products (being manufactured), systems as well as services delivered by it.

ISO 9001 : 2000 specifies the quality management requirements for a given organization seeking an overall development along with the customer's satisfaction.

To assess the quality of management ISO 9001 : 2000 has derived a special cycle referred to as, "*plan-do-check-act*". In this phrase, each term has got its own significance i.e.,

Plan – Includes the targets, associated activities, objectives with an aim to produce high quality software with complete customer's satisfaction.

Do – Refers to the implementation of entire software development process.

Check – Refers to the process of monitoring and assessing the software process. This assures that all mechanisms related to software quality management are thoroughly implemented to improve the software development process.

Act – Refers to the implementation of ideas of improving the current software development process.

1.2.6 Personal and Team Process Models

Q37. Compare the personal and team process models. Nov./Dec.-12(R09), Q1(b)

OR

Distinguish between personal and team process models.

Answer : Dec.-14(R09), Q1(b)

Personal Software Process (PSP)

Every team member possesses his own personal software process. If this personal process remains ineffective, then he/she is advised to go through the four phases of software development, each time being trained thoroughly. This ensures that he remain in a position to analyze the project development, at the same time focusing the quality of it. But, this session does not end here. Rather, PSP makes an individual involved in the project planning and also in maintaining the quality of all the software products. In order to achieve this, PSP adheres to the following five major framework activities,

(a) **Planning**

In this activity, the user initially focuses on the requirements of the project. Later, the vision is extended on the defect estimates. Finally, by estimating the development task and documenting them, this session is concluded.

(b) **High Level Design**

In this case, user usually addresses the scenario of entire project by developing certain high level designs. If uncertainty prevails in such diagrams, prototypes are considered. By documenting the whole scenario, current session is concluded.

(c) **High Level Design Review**

When the designing process completes, the design is to be reviewed, in order to isolate the errors. In this case, several designing reviewing methods are considered.

(d) **Development**

This is same as coding phase. The code for each component of the software is developed. Later these components are carefully integrated. Finally suitable testing methods are implemented to thoroughly test the resultant product.

(e) **Postmortem**

This is the final step in the entire PSP. In this case, by using several methods, the effectiveness of current process is claimed. If it is not up-to-date, then several modifications to the current process is made.

Finally, the PSP looks forward for the software engineers to trap the errors in the software process early. They continuously access the process and implement many process refining steps.

Team Software Process (TSP)

Team Software Process aims in developing self directed project team, which is motivated in organizing effective software process. At the same time it produces software which are of high quality. Following are certain preliminary guidelines, offered by TSP for any software engineering team.

- Form a team with self motivated members capable of analyzing their work, setting up their targets and scheduling the processes. Hence, such teams can be called as integrated product teams, accounting for maximum of 3–20 engineers as team members.
- Developing strong leadership skills in the project leaders or managers. This makes them capable of bursting and enriching enthusiasm among team members to attain high degree of performance.
- Take the team on the track of achieving CMM-5 targets.
- Address guidance of improving current process standards of high maturity organizations.
- Bestow industrial grade skills among the technological students, earning their university degrees.

Hence, any team claiming to be self directed will possess the following virtues.

- Clear idea of targets to be achieved.
- Each team member remains self acquainted with his responsibilities in delivering his role.
- Always maintain the entire project information up-to-date.
- Acquiring new team processes which remain adequate for team as a whole as well as gaining mechanisms to implement them.
- Maintaining a record of probable risks and defining remedies for curing them.
- Maintaining a record, which provides information on the entire project status.

Important activities required to be performed by a given team (addressed by ISP) are as follows,

- Launch
- High level design
- Implementation
- Integration
- Test
- Postmortem respectively.

Hence, while traversing through the above mentioned activities, a given team can successfully resort to systematic development of the project. At the same time, it remains curious about maintaining high quality standards. ISP also avails several scripts, standards etc., in continuously guiding the team members in achieving the goals set.

Finally, one can say that, TSP is a standard approach which aims in making current software development process competent in both productivity as well as quality aspects. Hence, it is always fruitful in perfectly implementing the TSP issues.

1.3 PROCESS MODELS

Q38. Explain software development life cycle. Discuss various activities during SDLC.

Answer : (Model Paper-III, Q2(b) | March-17(R13), Q3(a))

Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) refers to the overall process involved in the software development.

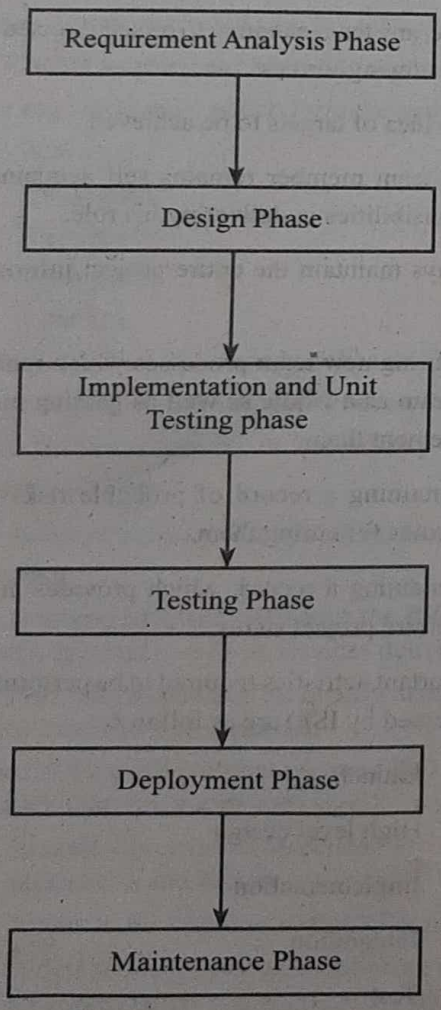


Figure: Different Phases in Software Development Life Cycle

It involves six phases namely,

1. Requirements Analysis Phase

Requirements analysis and specification phase is very important in the successful development of a software. If the customer requirements are not analyzed properly, the chances of project failure or amount of rework on project becomes very high. Therefore, the project startup without proper requirements analysis and documentation is considered as a major mistake. It also increases the burden on developer to reflect/implement the changes multiple times. Such iterations increase the cost incurred in the overall project and also leads to customer dissatisfaction.

The importance of requirements understanding and documentation is independent of the type of project i.e., internal or contract based. The documentation generated after successful analysis and specification is called Software Requirement Specification (SRS) document. This document not only helps in getting a clear understanding about product requirements, but also helps in carrying out different activities of later phases. Due to this fact, expert developers spend a lot of time to gather requirements.

2. Design Phase

Design is a meaningful engineering representation of something that is to be built. It can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria for "good" design. In the software engineering context, design focuses on four major areas of concern - data, architecture, interfaces and components.

Software design process is an iterative process through which requirements document (SRS) is transformed to design document.

3. Implementation and Unit Testing Phase

In this phase, actual code is developed using a programming language. As soon as the code is developed, each unit or component in it is tested individually. Such testing is called unit testing. It ensures that maximum errors are detected and entire code is tested by following certain paths in the control structure.

4. Testing Phase

Testing is a major activity for software development and its main function is to detect bugs or errors that occur while executing a software. During the phases like requirement analysis and system design, the output results in the form of textual documents whereas, for the coding phase, the output is a computer program. Hence, testing not only detect errors during coding but also detect errors that occur during the previous development phases. Subsequently, there are four different types of testing that can be performed on software development.

5. Deployment Phase

This forms the final phase of the software development process where the software is usually delivered to the end users for its effective implementation. Later, feedback is collected from the users and if, the software fails to meet the users requirements, it is modified. Hence, the important activities involved in this phase are:-

(a) Perfect delivery

(b) Support and

(c) Feedbacks.

6. Maintenance Phase

Software maintenance refers to the changes which are performed once the software is delivered to users. Maintenance activity is compulsory for each type of product and cannot be avoided. Generally, products requires maintenance activity due to its heavy usage. But, software need maintenance in order to improve features, correct errors. It is an essential activity adopted by large number of organization because of the misuse of hardware.

1.3.1 The Waterfall Model

Q39. What is waterfall model? How is it different from other engineering process models?

Answer : Model Paper-I, Q3(a)

Waterfall Model

Waterfall model remains one of the oldest strategies ever applied, in the development of a software. It is also known as *classic life cycle model,* which divides the entire software development process into five main phases. Following is the diagrammatical representation of waterfall model,

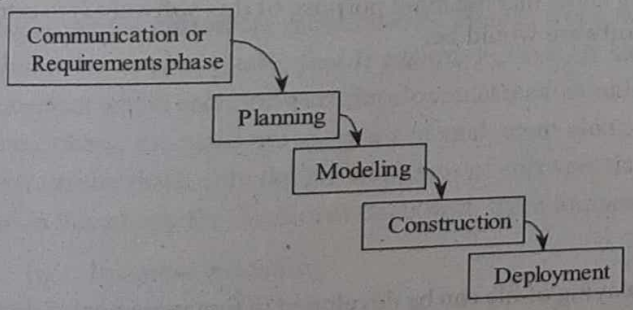


Figure: Phases of Waterfall Model

The waterfall model was proposed with feedback loops. However, most of the organizations avoid these loops. Thus this model is also called Linear Sequential Model.

1. Communication or Requirements Phase

In the first phase, the customer requirements are retrieved to generate customer requirements specification. Hence, all the stakeholders (or the end users) are called and their vision about the end product is enquired/collected and documented.

There are two important facts about this phase i.e.,

(a) Project initiation and

(b) Requirements gathering.

2. Analysis or Planning Phase

In this phase, the entire project strategy is devised. The main addressable issues in this phase are as follows:

(a) The entire project schedule is estimated.

(b) A thorough analysis of resource requirements is made.

(c) Number of people involved in the project is determined.

(d) Duration and expected cost of the project are estimated.

Therefore, the main focus of this phase remains on,

(i) Estimation

(ii) Scheduling and

(iii) Tracking.

3. Modeling Phase

In this phase, the entire project scenario which was analyzed in the second phase is modelled diagrammatically. To do so, the UML diagrams are used. Modeling remains important for successful completion of the project. The two important aspects of this phase are,

(a) Analysis and

(b) Design.

4. Construction Phase

This is usually the coding phase of the software. Here, each component of the software is coded and is suitably integrated. Once the coding part is completed, the entire software is thoroughly tested. Hence, the two most important activities performed during this phase are,

(a) Coding and

(b) Testing.

5. Deployment Phase

This is the final phase in which the software is usually delivered to the end users for its effective implementation. Later, feedback is collected from the users. If the software fails to meet the users requirements, it is modified. Hence, the important activities involved in this phase are,

(a) Perfect delivery

(b) Support

(c) Feedbacks.

Though the above software development process looked quite straight forward, it is now rarely used. Even the organizations, which remained loyal to this strategy, started raising serious objections, which added to its failure.

Problems Encountered in Waterfall Model

Following are the few factors or problems, encountered by the organizations implementing this model.

1. Customers are involved only during the customer requirement phase. If the customer addresses dissatisfaction after delivering the project, it imposes higher cost to the organization. The customers however do not describe their requirements in just one stroke.
2. It requires patience from the customer because, a working version of the project is made available to the customers late during the project life span. Hence, identification of a major problem would lead to disastrous results.
3. This is a time consuming process and leads to confusions among project members. Such projects cannot be truly tested or debugged.

1.3.2 Incremental Process Models

Q40. Describe the incremental software development process model.

Answer :

Incremental Model

Following is a diagram depicting an overview of incremental model.

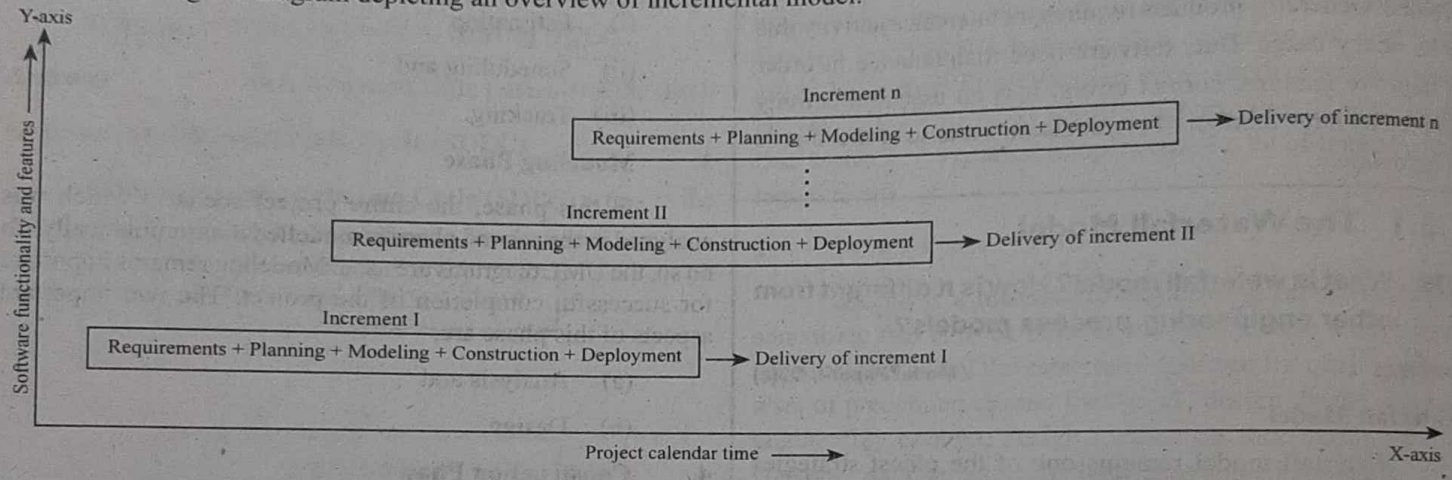


Figure: Representation of Incremental Model

It can be observed from the above figure that the incremental model divides its software development process into certain number of increments. Each increment comprises of five phases of waterfall model. At the end of each increment, an effective module of a given software is developed. In order to understand the above process, consider an example where a Video cutter software is developed. Students familiar with this software will certainly know that the main purpose of this software is to extract any number of files from a large video. The main components of this software would be,

(a) File extraction
(b) Playing of file
(c) Selection of initial point
(d) Selection of final point
(e) Storing of abstracted file.

In this process, the software components i.e., file extraction and playing of file can be developed in first increment. Selection of initial and final points can be done in the next increment and storing of abstracted file in the final increment.

Hence, a core part of the entire software has resulted from the first increment. Later, this core part is delivered to the users and feedbacks are claimed. Based on these feedbacks and with an intention to provide certain new functionalities to the software, next increment is carried out. These increments are repeated until the whole project is completed.

Advantages

1. When less number of people are involved in the project, incremental model is the correct choice.
2. With each increment, the technical risks involved in the project are reduced.
3. The customer can expect at least a core product in a short span of time from the project team.

Q41. Discuss the statement, "RAD model is a high-speed adaptation of the waterfall model".

Dec.-11, Set-1, Q7(a)

OR

Illustrate on RAD process model.

Answer :

Model Paper-II, Q3(a)

RAD (Rapid Application Development) Model

Diagrammatic representation of RAD model is given below.

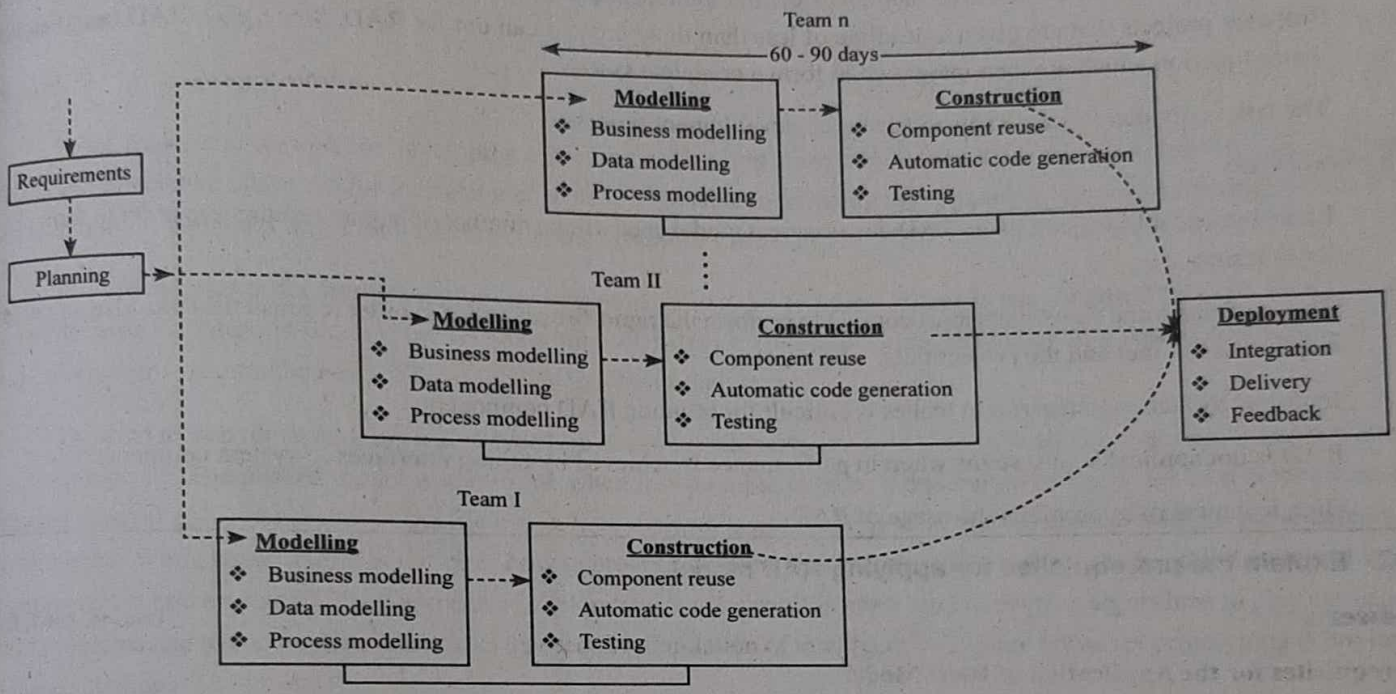


Figure: Rapid Application Development Model

RAD or Rapid Application Development model is one of the software development process models that focuses on short term delivery of the product whenever the developer gets knowledge about the user requirements. Moreover, if the development time is short, then RAD will be the best option. It is a version of waterfall model with rapid development.

It can be observed from the above figure, that the RAD model initially begins with requirements phase which is nothing but acquiring the customers requirements. Hence, when the requirements for what is to be developed are known, the developer can switch on to next phase, that is *planning phase*. It has to be noted that the RAD model emphasizes on component-based development where each software development team is authorized to handle single component of the given product. Hence, during planning phase, the tasks are assigned to each team along with the project schedule. The first two phases i.e., *requirement* and *planning phase* mark only the initial phases of software development. The main process of development begins at the *modeling phase*. In this phase, the developer deals with three important aspects i.e.,

(i) Business modelling

(ii) Data modelling

(iii) Process modelling.

These three aspects are separately represented by an individual team. The next phase is nothing but the *construction phase* where the actual coding is done. The main aspects of this phase are,

(i) Component reuse

(ii) Automatic code generation and

(iii) Testing.

The final phase is the *deployment phase* where different components are integrated to form single software. The software is generally delivered to the end users and their feedbacks are collected. Accordingly, rectifications (if required) are made to the software. Hence, the fundamental aspects of this phase are,

(a) Integration

(b) Delivery and

(c) Feedback.

Advantages

1. Software projects that are given a deadline of less than three months can opt for RAD. Since, each RAD team handles a major function which are then integrated to form a complete system.
2. The task is divided among teams to fasten the development process.

Disadvantages

1. Large but scalable projects using RAD development model need a huge number of human resources in order to form proper RAD teams.
2. Both customers and developers must commit to perform the rapid-fire activities in order to finish the task. Else, the defined deadline is not met and the project fails.
3. Improper system modularization makes it difficult for building RAD components.
4. RAD is not applicable to systems wherein performance is achieved by tuning interfaces to system components.
5. High technical risks prohibits the usage of RAD.

Q42. Explain the prerequisites for applying RAD model.

Answer : Dec.-11, Set-1, Q7(b)

Prerequisites for the Application of RAD Model

A project manager applies RAD model when the following prerequisites occur,

1. Complete involvement of the users in the use of automated tools.
2. Involvement of the end user throughout the life cycle.
3. Availment of reusable parts through automated software repositories.
4. Reduced technical risks.
5. Satisfaction of the project team in,
   (a) Knowledge of problem domain.
   (b) Expertise in development tools.
   (c) Dynamic motivation.

RAD model is also applied on the following systems,

(i) Low performance systems

(ii) Non-critical or small system

(iii) Incremental developmental system

(iv) Short term projects

(v) Well defined requirements system

(vi) Scalable and modularized system.

1.3.3 Evolutionary Process Models

Q43. What is meant by prototyping? Discuss in detail the prototyping model.

Answer :

Prototyping Model

A prototype is a mock-up or model of a software product. In contrast to a simulation model, a prototype incorporates components of the actual product. Typically, a prototype exhibits limited functional capabilities, low reliability and inefficient performance.

Reasons for Developing Prototype

There are several reasons for developing a prototype. One important reason is to illustrate input data formats, messages, reports and interactive dialogues for the customer. This is a valuable mechanism for explaining various processing options to the customer and for gaining a better understanding of the customer's needs.

The second reason for implementing a prototype is to explore technical issues in the proposed product. Often a major design decision will depend on, say, the response time of a device controller or the efficiency of a sorting algorithm. In these cases, a prototype may be the best, or the only way to resolve the issue.

The third reason for developing a prototype is in situations where the phased model of analysis → design → implementation is not appropriate. The phased model is applicable when it is possible to write a reasonably complete set of specifications for a software product at the beginning of the life cycle. Sometimes it is not possible to define the product without some exploratory development. While sometimes it is not clear how to proceed with the next enhancement to the system, until the current version is implemented and evaluated. The approach of exploratory development is often used to develop algorithms to play chess, solve maze problems and to accomplish other tasks that require simulation of intelligent behaviour. However prototyping is not limited to these situations.

The nature and extent of prototyping to be performed on a particular software project is dependent on the nature of the project. New versions of the existing product can most likely be developed using the phased life-cycle model with little or no prototyping.

Method of Prototyping

Prototyping enables the developer to create a model of the software that must be built. The model can take one of the three forms,

(i) A paper prototype or PC-based model that depicts human-machine interaction in a form that enables the user to understand how such interaction will occur.

(ii) A working prototype that implements some subset of the function required for the desired software or

(iii) An existing program that performs part or all of the functions desired. The features will be improved in the new development effort.

Prototyping begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known and outline areas where further definition is mandatory. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the user (e.g., input approaches and output formats). The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and is used to refine requirements for the software to be developed. A process of iteration occurs as the prototype is "turned" to satisfy the needs of the customer along with enabling the developer to better understand about what needs to be done.

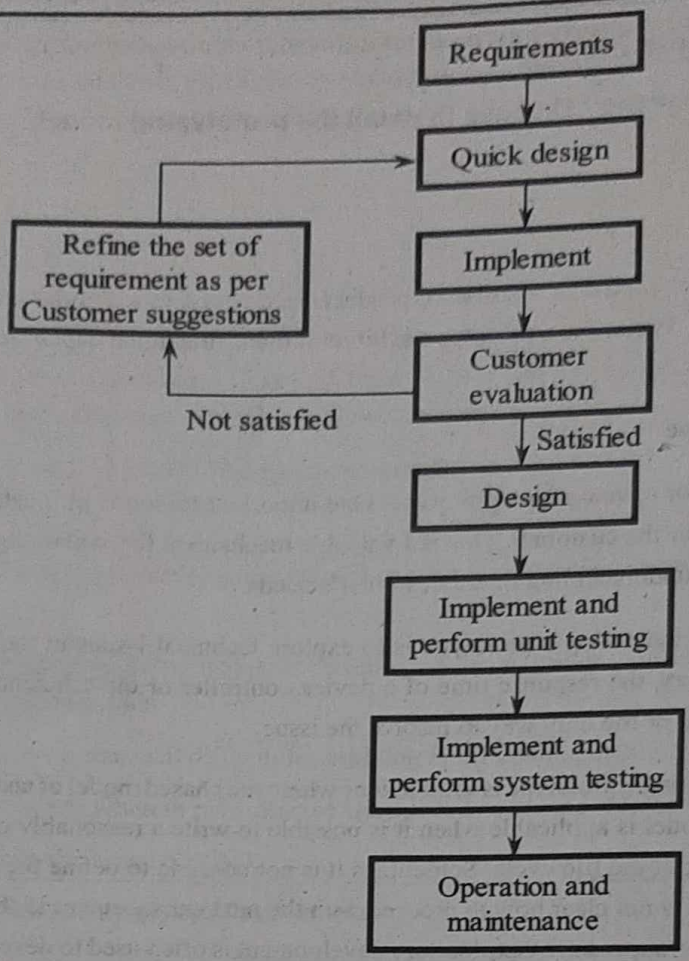


Figure: Prototyping Software Life-Cycle

Advantages of Prototyping Model over Waterfall Model

(i) The Prototype Models give a prototype that can be used to illustrate input data formats, messages, reports and interactive dialogues for the customers. This is a valuable mechanism for explaining various processing options to the customer and for gaining a better understanding of the customer's needs. This is not possible in Waterfall Model.

(ii) The prototype explores technical issues in the proposed product. Often, a major design decision will depend on, say, the response time of a device controller or the efficiency of a sorting algorithm. In these cases, a prototype may be the best, or only way to resolve the issue. The waterfall model does not explore the technical issues in the proposed product.

Example

The process of system development begins with the initial requirements laid by the users and customers. According to these requirements, several alternatives are derived which can be accessed directly by customers and users. Customers and user finally decide from the various alternatives and agree upon a particular alternative which is then passed for design. Similar to the requirements and output given by developers, designs are also explored and the final design is chosen with consulation for customer.

Drawbacks

1. The customer seeks for a quick working version with few fixes. However, prototyping model needs that the system is rebuilt and high quality is maintained.
2. While developing the prototype, the developers compromise the implementation for its quick working. However, many times these compromises cannot be retaind when prototype start working thereby leading to inaccurate design.

Q44. Explain spiral model with a neat sketch. What can you say about the software that is being developed or maintained as you move outward along the spiral process flow?

Answer :

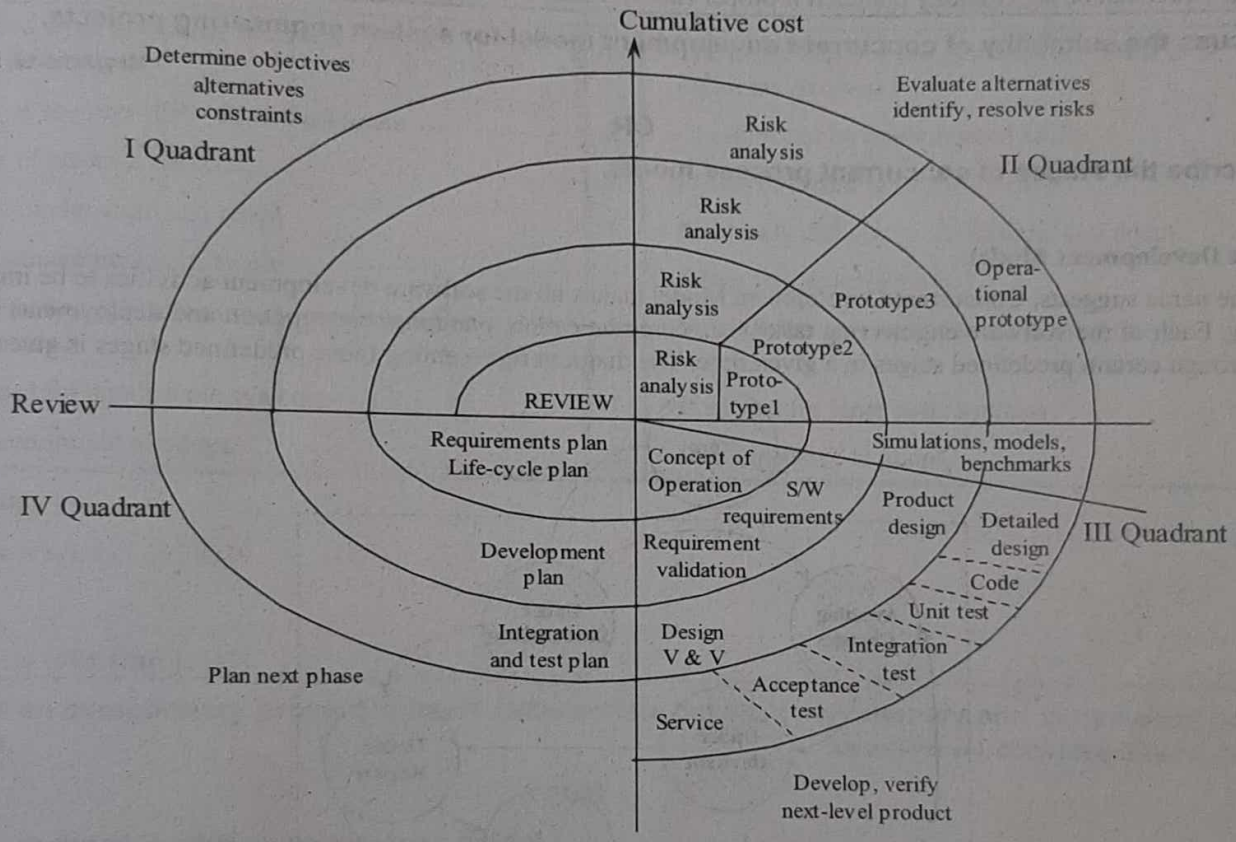
Spiral Model

(Model Paper-III, Q3(a) | Nov.-15(R13), Q2(b))

The spiral model for software engineering includes the features of classic life cycle and prototyping with an added advantage of element-risk analysis. It provides a framework for the design of software production process with due consideration of risk levels that may incur while designing. The spiral model can be used as a reference for choosing the final development model.

Risks in software production may create problems in its development and can effect the product quality. These risks may result in the failure of the software operation or expensive software rework. Spiral model identifies and eliminates these high risk problems.

Spiral model is cyclic with each spiral cycle consisting of four stages. Each stage is represented by one quadrant of Cartesian diagram. The radius of the spiral shows the cost of the process while the angular dimension denotes the progress in the development process.



The four sectors (stages) are as follows,

(i) Object Setting

Specific objectives for the phase of the project are defined. Constraints on the process and on the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on the risk, may be planned.

(ii) Risk Assessment and Reduction

For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

(iii) Development and Validation

After the risk evaluation, a development model for the system is then chosen. For example, if user interface risks are dominant, an appropriate development model might be evolutionary prototyping. If safety risks are the main consideration, development based on formal transformations may be the most appropriate and so on. The waterfall model may be the most appropriate development model, if the main identified risk is subsystem integration.

(iv) Planning

The project is reviewed and a decision is made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

While developing the software, the software team progresses around the spiral. The first track in the spiral gives the development of product specification. The consecutive tracks around the spiral is used to create prototype while proceeding through each track. In planning region, the programmers makes some adjustments in the project plan. And lastly, depending upon the customer requirement, cost and schedule adjustments are made.

Advantages

1. At each stage, the set of requirements can be changed depending on the requirement.
2. Identification and rectification of risks can be done at early stages.

Disadvantages

1. The product is developed based on the communication done with customer. Thus, improper communication may result in an inefficient product.
2. The product can be successfully obtained if proper risk assessment is done.

Q45. Discuss the suitability of concurrent development model for system engineering projects.

May/June-12, Set-3, Q4(a)

OR

Describe the stages of concurrent process model.

Answer :

Concurrent Development Model

As the name suggests, Concurrent Development Model makes all the software development activities to be implemented concurrently. Each of the software engineering tasks (i.e., communication, planning, construction and deployment) need to be traversed through certain predefined stages in a given time. The diagram representing those predefined stages is given below,

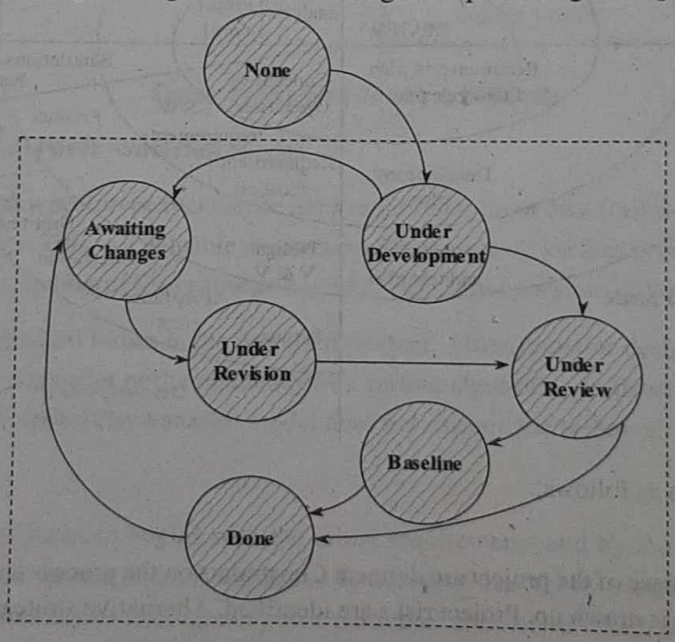


Figure: Stages of Software Engineering Tasks

It has to be noted that any software engineering task will reside at any of the above represented stages at a given time.

The arrows represented in above diagram depict flow of tasks from one state to the other. Usually, the concurrent process model can be applied in any type of software development activities. The speciality of this model is that, it does not support sequential flow of software engineering tasks (which is observed in other model). Rather, it motivates simultaneous development.

Advantages

1. CDM can be applied to almost all software development projects.
2. The engineering activities, tasks and actions are defined as a network of activities, and are not confined to a sequence of event.

Q46. Compare and contrast between waterfall model and spiral model with neat diagrams.

Dec.-19(R16), Q2(a)

OR

Explain the following,

(i) Waterfall model

(ii) Spiral model.

Nov./Dec.-18(R16), Q2(b)

(Refer Only Topics: Waterfall Model, Spiral Model)

OR

Compare the linear sequential model with the spiral model.

(Refer Only Topics: Comparison between Waterfall and Spiral Models)

Answer :

Nov./Dec.-12(R09), Q2(a)

Comparison between Waterfall and Spiral Models

	Linear Sequential (Waterfall Model)		Spiral Model
1.	The flow from one phase to another phase is in linear fashion.	1.	The flow from one phase to another phase is in iterations (spirals).
2.	Changes are very difficult to implement.	2.	Changes can be implemented easily.
3.	Fixing of errors is difficult.	3.	Fixing of errors is relatively easy.
4.	Easy to understand and adapt.	4.	Relatively difficult to understand and adapt.
5.	Risk management is difficult.	5.	Better risk management.
6.	Suitable for applications that need development from scratch.	6.	Suitable for component based development.
7.	It is used for small scale systems.	7.	It is used for large scale systems.
8.	It is economical to adopt.	8.	It is expensive to adopt.

Waterfall Model

For answer refer Unit-I, Q39.

Spiral Model

For answer refer Unit-I, Q44.

Q47. What is an evolutionary process model? Differentiate between evolutionary and incremental process models.

(Model Paper-I, Q3(b) | May/June-12, Set-1, Q5)

OR

Explain in detail Evolutionary process model.

(Refer Only Topic: Evolutionary Process Model)

Answer :

May-18(R15), Q3(b)

Evolutionary Process Model

The evolutionary process model tends to develops a high quality software iteratively or incrementally. It is used to highlight the flexibility, extensibility and speed of development. There are two common evolutionary process models discussed below,

1. Prototyping model
2. Spiral model.

1. Prototyping Model

For answer refer Unit-I, Q43.

2. Spiral Model

For answer refer Unit-I, Q44.

Differences between Evolutionary Process Model and Incremental Process Models

	Evolutionary Process Model		Incremental Process Model
1.	The evolutionary process models are designed to adopt the changes. The iterative and incremental nature of software projects are identified by these models.	1.	The nature from incremental models is iterative. The software working versions are produced quickly from these models.
2.	The whole cycle of activities is repeated for each version.	2.	Each increment is designed, tested and delivered separately at successive points in time.
3.	Although, compatibility is desirable between the successive versions, it has no assurance.	3.	Compatibility is necessary between the successive increments, for this model to be accepted.
4.	It supports changing of requirements.	4.	Changing of requirement is cost-effective but it is not suitable to change.
5.	Analysis of risk is better.	5.	Management of risk is easy. It is identified and resolved for each iteration.
6.	Progress depends upon the risk analysis phase.	6.	Possibility for measuring the progress.

Q48. Explain the working of specialized process models.

Nov./Dec.-17(R15), Q3(b)

Answer :

The working of specialized process models depends on the one or more characteristics of conventional models. Following are various specialized process models,

1. **Component Based Development**

   For answer refer Unit-I, Q49.

2. **Formal Methods Model**

   For answer refer Unit-I, Q50.

3. **Aspect-oriented Software Development**

   For answer refer Unit-I, Q51.

Q49. Explain briefly about the component-based development.

Answer :

Component-based Development

Component based Development is a specialized software development scheme which offers new approach to the design, construction, implementation and evolution of software applications. This model is integrated with several aspects of the Spiral Model.

In a component-based development, software applications are developed by assembling Commercial Off-the-Shelf (COTS) software components (prepackaged components) which are developed by manufactures. These components can be software packages, web services or modules that provide specific (required) functionality with well defined interfaces that enable the components to be easily embedded into the software.

The first step in constructing and modelling a component model is the identification of candidate components. Once the components are identified, the design process is done using any of the following technologies,

(i) Conventional software module

(ii) Object-oriented class

(iii) Package.

Irrespective of the technology used in creating the component, the development of component-based model is carried out in the following stepwise manner:

Step1: In this step, the component-based products are researched and evaluated to fit into the need of application domain to be constructed.

Step2: After searching and evaluating, the selected components are verified for any integration problems.

Step3: In this step, a software architecture is designed, such that the selected components can be embedded together.

Step4: Once the architecture is ready, the components are integrated into the architecture.

Step5: Finally, the application is extensively tested to make sure that it functions according to the expectations (requirements).

Advantage of Component based Development

One of the major advantage of the Component Based Development is the reusability factor. Reusability is an important characteristic of a high quality software which provides many benefits and case-of-use for software engineers.

Q50. Discuss about the formal methods model.

Answer :

Formal Methods Model

Formal Methods Model is a specialized software development approach that uses mathematical based techniques for specifying, developing and verifying the computer software. This approach helps the software developers to apply correct mathematical notations to create a specification that is more complete, consistent and unambiguous.

In contrast to conventional software development schemes, formal methods provide many ways of solving the problems that were difficult in case of conventional techniques. Some of these issues are related to insufficiency, inconsistency and uncertainty of the software. Formal methods easily detect and correct these issues by applying mathematical analysis without any adhoc review. During the design phase, formal methods model acts as a program verifier. They help the software developers to detect and correct those errors that were otherwise, very difficult to be detected.

Formal Methods Model assures a defect-free software, but it has its own drawbacks in its applicability into a business environment.

Drawbacks of Formal Methods Model

1. Software application development using the Formal Methods Model is very time consuming and expensive.
2. Many of the software developers requires extensive training about the procedure of applying this model.
3. If the clients are not technically sound then it is difficult to use this model as a communication tool.

Because of these reasons, formal methods are usually used only in development of high integrity software applications where safety and security is of utmost importance.

Q51. Explain in detail the aspect-oriented software development.

Answer :

Aspect-Oriented Software Development is a specialized approach in software development that overcomes the limitations of other software developing approaches such as Object Oriented Programming.

The developers of complex software executes certain localized characteristics of software (such as localized function, feature, information). These are initially modelled as component and then constructed in accordance to the system architectural requirement. However, due to the rapid development, the complexity of such software systems is increasing, which inturn giving rise to various concerns. Some of these concerns are related to high-level properties such as security and fault-tolerance. In addition to these concerns, there are other concerns that have impact on the functionality. A concern is referred to as a "crosscutting" concern, if they span across multiple system function, feature, information. The crosscutting concern that affects the entire software architecture is referred as "aspectual requirements".

The Aspect-Oriented Software Development focuses on identification, specification and representation of these crosscutting concerns and provides various mechanisms to systematically modularize these concerns. The modularization is done by encapsulating every crosscutting concern in a separate module which is known as 'aspect'. The encapsulation promotes localization which provides better support for modularization. Due to this, the costs related to the development, maintenance and evaluation of the software are reduced.

Presently, there is no distinct aspect-oriented approach. However, if such an approach is developed, then it must integrate the characteristics of both the spiral and concurrent models. The spiral process is used because of it's evolutionary nature as aspects are initially identified and then constructed. On the other hand, the concurrent process is used due to it's parallel nature, as aspects are constructed without depending on the localized software components.

1.3.4 The Unified Process

Q52. Analyze the importance of the Unified process in software development. Dec.-19(R16), Q2(b)

OR

Give an overview of unified process model. (Model Paper-II, Q3(b) | Nov./Dec.-18(R16), Q3(a))

OR

Elaborate on unified process in detail. Dec.-11, Set-2, Q3

Answer :

Unified Process or Rational Unified Process covers five main phases,

1. Inception
2. Elaboration
3. Construction
4. Transition and
5. Production.

These five phases are bound to give framework activities which remain common to almost all older software development processes. Following is a diagrammatic overview of unified process along with the five generic software development activities.

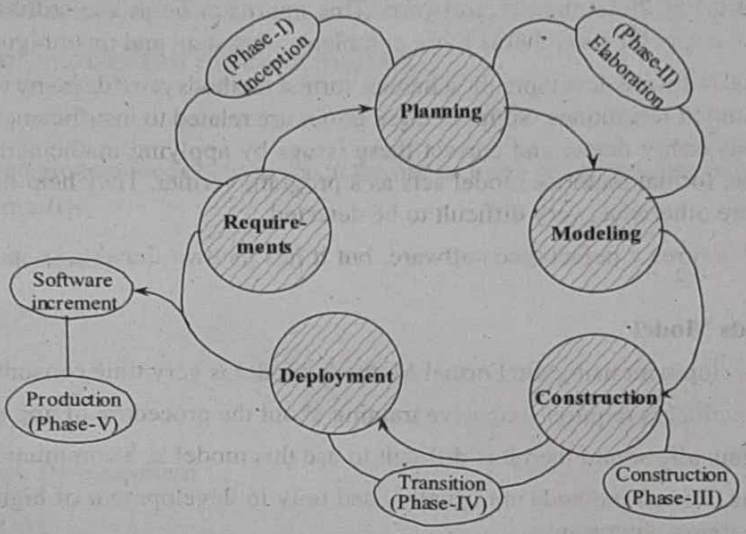


Figure: Different Phases of Unified Process along with Five Generic Software Activities

1. Inception Phase

Inception phase combines the features of both communication and planning tasks of software development. Hence in this phase, end user is communicated and business requirements are collected. Later, a simple outline depicting the architecture of the projects is drawn. Here, the major requirements of the project and also the activities performed by each stakeholder in the project is represented in the form of highly abstract use cases. Hence, at this stage only fundamental details of the project are revealed. Since, the planning phase is also being considered, the developer must even work on the projects like identification of resources, major risks, the schedule of the project etc. As all the details in this form are represented in the most abstract form, it can be refined in forthcoming phases. Usage of use cases provides project scope which is essential during project planning.

2. Elaboration Phase

As shown in the figure, the elaboration phase combines the features of customer communication and modelling tasks of the software development process. At this stage, the abstract information provided during the inception phase is refined and enlarged. The only view which existed previously is now divided into five different aspects i.e., five different models providing five different views. These models are,

(a) Use case model

(b) Analysis model

(c) Design model

(d) Implementation model and

(e) Deployment model.

In this phase, an executable architectural base line is proposed. This base line though provides viability of the project but at the same time cannot provide a complete functionality of it. Further, any modifications (if required) are also made to the project depending on the requirement.

3. **Construction Phase**

Construction phase is same as the Software Development Construction. The primary aspect of this phase is to develop suitable code for each component of the software. For this purpose, various models are considered which are designed during inception and elaboration phases of a unified process. Especially, a final approval is made to the use case developed and they are transformed into code. Then unit testing is performed for each component. When this testing is done, integration of each of these components is performed. Finally, this session is concluded by conducting acceptance tests.

4. **Transition Phase**

Transition phase marks the end of construction and the beginning of deployment phases respectively. Hence all the activities necessary to leave behind in construction phase completely are performed. So the attention will now be only towards the implementation of the deployment phase. During this phase, usually the software developed is adhered to the end users for a better testing. After collecting the information on the number of defects and queries related to operation of the software, well defined modifications are made to it. Also, software support information is prepared i.e., troubleshooting information, installation procedure, user guides etc., and the product is moved to the final phase.

5. **Production Phase**

Production phase reflects the deployment activity to certain extent. Here, the software team makes a close observation of the software by exercising it through various levels. Again, any of the defects encountered are reported and are modified. Software is deployed in the machine and is checked to ensure that it remains compatible with the surroundings.

While dealing with unified processing phase, one has to remember that, a new software development activity gets initiated when software moves from one phase to another. Hence, software development activity remains concurrent.

Merits

1. It covers the complete software development life cycle.
2. Even though the unified process model came into existence after a true hardwork of over 20 years, it is available on internet in the form of an electronic guide (available to everyone located anywhere around the globe).
3. Most of the modern techniques and approaches use the unified process model as a set of guidelines.
4. The best practices for software development are supported by unified process model.
5. Unified process model does not result in a frozen product. Rather the product is ever evolving and is constantly maintained.
6. Its process architecture can be tailored as per requirement.
7. It encourages UML as the best process-oriented languages protocols.

Demerits

1. It does not provide any guidelines for determining appropriate use cases.
2. It cannot determine objects.
3. Even though architectures in the development process are focused the development method does not specify the useful architecture for the analysis models. Moreover, it does not provide any guidance on how these architectures can be found.

Q53. What is meant by unified process? Elaborate on the unified process work products.

OR

Explain various work produced in elaboration phase of the unified process.

(Refer Only Topic: Elaboration Phase)

Answer : Dec.-11, Set-4, Q2

Unified Process

Unified Process refers to a methodology of extracting the most essential activities of conventional software development phases (communication, planning, modelling, construction and deployment). It also characterizes the activities so that they can be applied in development of highly valued software.

Work Products

The outcome at each phase results in the generation of few documents. These documents are called "work products". The work products at different phases of the unified process model are,

1. **Inception Phase**

The inception phase must,

(a) Produce a business case.

(b) Identify the business requirements, business and process risks.

(c) Give the overall vision for the project as its output. These outputs result in various documents/work products.

❖ **Vision Documents**

The overall objective and scope of the project is specified in the vision document.

❖ **Initial Use-case Model**

The initial use-case model gives the user diagrams that are necessary at this stage. Thus, at inception phase, only 10-20% of the use-case model is obtained.

❖ **Initial Business Case**

Financial forecast, market conditions, revenue model and business difficulties, etc., are specified in the initial business case.

❖ **Project Plan**

The phases and iterations required for the projects development are mentioned in the project plan document.

❖ **Business Model**

A brief description about cash flow threads, "time to market" or deadline and marketing strategies are given in the business model document.

In addition to these, documents like initial risk assessment and initial project glossary are also generated. The optional document in this phase is one or more prototypes giving the clear understanding of the project.

2. **Elaboration Phase**

By the end of the elaboration phase,

(a) The architectural foundations must be established.

(b) Further designing of the use-case model is done.

(c) The development environment, infrastructure and process are elaborated.

(d) Component selection for the project. In few cases component selection is not possible.

❖ **Work Products Use-case Model**

In the elaboration phase, the use-case model for the product is completed by 80% or 90%. The possible use-cases and actors required for the project are determined and specified.

❖ **Supplementary Requirements Document**

This document gives the supplementary requirements along with the nonfunctional requirements of the project.

❖ **Prototype**

An executable architectural prototype which gives an overview of the end product.

❖ **Development Plan**

This document gives the development plan along with workflow, iterations etc.

❖ **Preliminary User Manual**

The optional document in elaboration phase is the preliminary user manual. The manual gets refined in the latter phases.

❖ **Revised Risk Document**

The risks related with the development process are revisited to check their validity. This information is specified in the revised risk document.

3. **Construction Phase**

The objectives of the construction phase are,

(a) Project implementation

(b) Development cost minimization

(c) Resource management and optimization

(d) Product testing

(e) Comparing the product releases with the acceptance criteria.

Work Products

Various documents are generated as the outcome of construction phase. These are,

(i) User manuals, documentation manuals and operational manuals

(ii) Software product

(iii) Test suite

(iv) Test outline

(v) Documentation of the current release.

4. **Transition Phase**

In the transition phase,

(a) Beta testing is initiated

(b) User's views are analyzed

(c) Users are trained

(d) Tuning activities are performed

(e) Customer satisfaction is assessed.

Work Products

(i) Beta test reports

(ii) User feedback

(iii) Product release.

UNIT 2

Software Requirements, Requirements Engineering Process, System Models

Syllabus

Software Requirements: Functional and Non-functional Requirements, User Requirements, System Requirements, Interface Specification, The Software Requirements Document.

Requirements Engineering Process: Feasibility Studies, Requirements Elicitation and Analysis, Requirements Validation, Requirements Management.

System Models: Context Models, Behavioral Models, Data Models, Object Models, Structured Methods.

PART-A SHORT QUESTIONS WITH SOLUTIONS

Q1. Classify various functional and non-functional requirements.

Answer : Dec.-19(R16), Q1(d)

The system requirements of a software can be classified into functional and non functional requirements. Here, functional requirements describes the functionality of system and is not classified further. However, non-functional requirements address different issues of a system and is classified into various types.

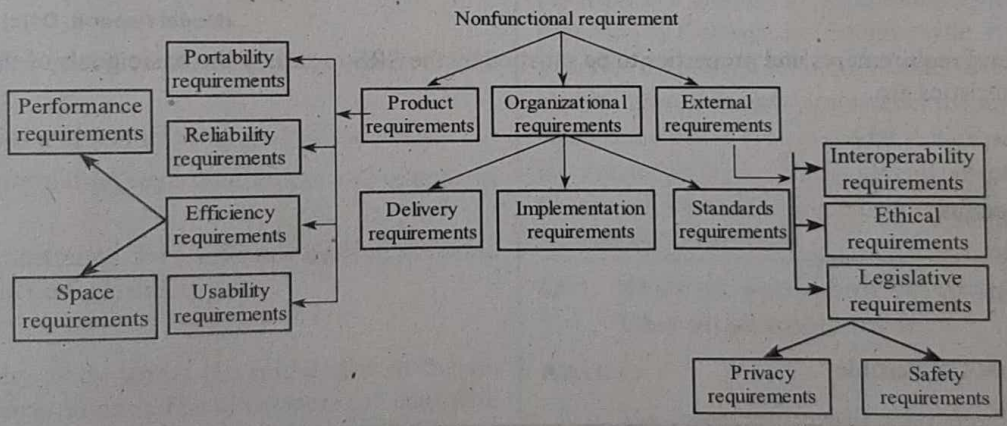


Figure: Hierarchical Tree Depicting Types of Non-functional Requirements

Q2. What are the differences between functional requirements and non-functional requirements?

Nov./Dec.-18(R16), Q1(d)

OR

What are non-functional requirements?

(Refer Only Topic: Non-functional Requirements)

Answer : (Model Paper-I, Q1(c) | March-17(R13), Q1(d))

Functional Requirements

"Functional requirements of a system" describes the functionality of a system. The functionality of any system depends on the purpose of its manufacturing to a greater extent. It is less concerned about people using it and the organization developing it. Hence, it describes the detailed scenario of the system functionality.

Non-functional Requirements

Non-functional requirements address different issues of a system, like its availability, reliability, performance, security and various other related facts. Hence, they are critical than functional requirements. Using these requirements, a specimen can easily avoid the errors occurring due to mis-implemented functional requirements. For example, no one would prefer of purchasing a water heater if its operation is unreliable i.e., if the heater is shock prone. While dealing with non-functional requirements, one has to remember, it not only deals with the end product or the system (being developed) but also its process of manufacturing. There are many reasons for the outcome of non-functional requirements.

Q3. Write any three metrics defined to measure non-functional requirements.

Answer :

The metrics defined to measure non-functional requirements are as follows,

- Property : Speed
  Metric : It is measured in terms of screen refresh time, user event response time, number of transactions being processed per second.
- Property : Portability
  Metric : (i) Percentage of statements directly dependent on the target system.
  (ii) Total number of target systems.
- Property : Ease of use
  Metric : (i) Total number of help frames required.
  (ii) Time spent in training.

Q4. What are the steps to be followed while performing the domain analysis process?

Answer :

The following are the steps carried out while performing the domain analysis process,

(i) The domain that needs to be analyzed must be defined.
(ii) The items retrieved from sources of domain knowledge must be categorized.
(iii) The sample of application defined in the domain must be collected.
(iv) Every individual application in the sample is analyzed and based on this, analysis class must be defined.
(v) An analysis model for the class must be developed.

Q5. What are the characteristics of a good SRS document?

Answer :

(Model Paper-II, Q1(c) | Nov.-15(R13), Q1(d))

There are several requirements and properties to be satisfied by the SRS to satisfy the basic goals of the proposed system. The desirable characteristics are,

1. Correct
2. Complete
3. Unambiguous
4. Verifiable
5. Consistent
6. Stability
7. Modifiable/Convertible
8. Traceable.

Q6. Differentiate between user requirement and system requirement.

Answer :

May/June-19(R16), Q1(c)

	User Requirements		System Requirements
1.	User requirements are mainly concerned with "What" the system should do.	1.	System requirements are concerned with "what" as well as "how" software tasks should be performed.
2.	They are abstract.	2.	They are precise and detailed.
3.	They are statements in the form of natural language and also in the form of diagrams.	3.	They are more detailed descriptions of the software system's functions and operational constraints.
4.	They are usually read by the system end-users, client engineers, and system architects.	4.	They are usually read by software developers and system architects.

Q7. What is the intent of requirements validation?

Answer : Nov.-15(R13), Q1(c)

This is the final step in the requirements engineering process, where all the proposed and refined requirements are verified. This concludes that, they are laid in accordance to the user specifications. Hence, the end product of this phase will be the requirements documents.

In short, all the above mentioned activities can be partitioned under three activities.

- Discovery
- Documentation and
- Checking.

However, when the developer visualizes the practical conditions, the scenario changes drastically. Human requirements change depending on the time factor, the organization purchasing the software changes, the hardware on which the given software being implemented changes etc. Hence, the software developed should be well acquainted to satisfy all these anomalies.

Q8. Discuss software scope and feasibility study.

(Model Paper-II, Q1(c) | Dec.-19(R16), Q1(c))

OR

What is feasibility study? Nov./Dec.-18(R16), Q1(c)

(Refer Only Topic: Feasibility Study)

OR

What is the significance of feasibility study?

(Refer Only Topic: Feasibility Study)

Answer : Nov./Dec.-16(R13), Q1(d)

Software Scope

Software scope specifies the following aspects of the software,

1. Functionalities and features required by the users.
2. Complete information regarding inputs and outputs of software.
3. Reliability constraints over software project, software interfaces and performance.

Feasibility Study

The feasibility study forms the initial step in the requirements engineering process. The developers can start with feasibility study process whenever the information related to the following aspects are obtained.

- The business requirements.
- The extent to which current system is capable of satisfying the requirement.
- A brief description of the system.

Hence, whenever feasibility study process is completed, the developers will be ready with the documents specifying whether the current system is capable/incapable of satisfying complete business requirements. Based on the result, the decision on whether to set aside, move further or refine the current system is taken. Therefore, when the feasibility study reports are developed, it should contain data related to the following aspects.

- Does the current report satisfy the business aspects of the organization involved in developing the system?
- Is it possible to implement the system using the current technology within given cost and schedule constraints?
- Can system be integrated with certain other well defined systems?

Q9. List the various types of feasibility studies.

Answer : May/June-19(R16), Q1(d)

The various types of feasibility studies are,

1. Technical feasibility study
2. Managerial feasibility study
3. Economic feasibility study
4. Financial feasibility study
5. Cultural feasibility study
6. Social feasibility study
7. Safety feasibility study
8. Political feasibility study
9. Environmental feasibility study
10. Market feasibility study.

Q10. Write short notes on VORD method.

Answer :

The Viewpoint Oriented Requirement Definition (VORD) is a method of requirements elicitation and analysis proposed by Kotonya and Sommerville. It has been designed as a service framework for requirements discovery and analysis. It also includes different steps in order to translate the analysis into an object-oriented system model. A viewpoint-oriented analysis has a major ability to identify several perspectives and provides a framework for discovering process in the requirements designed by various stakeholders.

Q11. What do you mean by viewpoint structuring? Give an example for it.

Answer :

Viewpoint structuring is an important phase of VORD method, which involves combining of similar view points into a hierarchical form. The similar viewpoints based on the specific services are arranged in a structural form for easy understanding and analysis purposes.

In technical aspect, viewpoint structuring is an iterative process of dividing the target system into a hierarchy of functional sub systems. It provides structure to the viewpoints, which are placed at the same level as the target system. Therefore, each functional subsystem can be represented as a viewpoint. Viewpoint structuring analyzes the target system as well as the functional sub systems in order to understand the requirements of the software system.

A simple example of viewpoint structuring of a college is shown below.

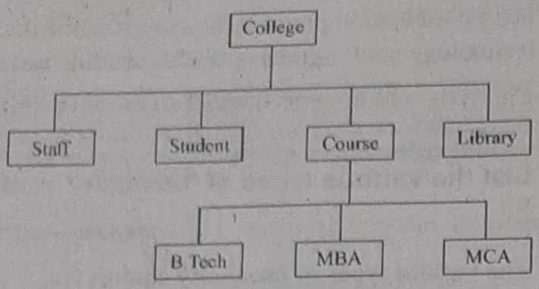


Figure: Viewpoint Structuring

Q12. Define Brainstorming.

Answer :

Brainstorming is defined as a group creativity technique that is designed to generate a large number of ideas as an optimal solution to a problem. During requirements elicitation and analysis, brainstorming may be used to understand the requirements in an effective and efficient manner. It is particularly helpful when a problem with many issues arises, for generating new ideas, creative thinking and new opportunities to develop highly creative solution to a problem.

Q13. What is meant by requirement management?

Answer : **May-18(R15), Q1(c)**

Requirement management is an approach which addresses the issues that arise during requirements gathering. Requirements management appeals to customize the software with an ability to understand and effectively control the changes to system requirements.

Initially, all the independent requirements are taken into account. Later, the possible requirements which may arise on the advent of these requirements are analyzed and accessed.

Q14. Discuss the statement, "requirements management needs automated support".

Answer : **Model Paper-I, Q1(d)**

Automated support is necessary for requirements management. The CASE tools that are required for this support must be selected during the planning phase. The automated tool must support the following functions,

(i) Storage for Requirements

The storage for requirements must be secured and well managed. Data should be easily accessible to all people involved in the requirements engineering process.

(ii) Management for Traceability

This feature allows the discovery of related requirements. Relationships between the requirements may be discovered using tools that use natural processing techniques.

(iii) Management for Changes

Change management process can be simplified with the help of available active tool support. Changes in the requirement can be documented and well managed by using automated tools.

Small projects does not require the use of special requirement management tools support can be provided hereby using available word processors, databases and spreads sheets. However, for large projects special requirements management tools are required.

Q15. What is ATC?

Answer :

Air Traffic Control (ATC) system is one which separates the air crafts in order to protect them from collisions, thereby organizing or stream lining flow of traffic. Here the process of preventing the collisions is called separation.

When air craft flies with the assistance of air traffic control system, it is controlled in air space, otherwise uncontrolled air space. In controlled air space air traffic controllers are provided for separately guiding the air crafts. All pilots follow the instructions issued by ATC system.

Q16. Define high level design matrix.

Answer :

A matrix that correlates any two documents requiring many-to-many relationship between them for determining the completeness of the relationship is called as high level design matrix. It is also known as traceability matrix or traceability table. It is used with high-level and detailed requirements of the software product that matches with the high-level design, detailed design, test plan and test cases.

Q17. Write the purpose of context model.

Answer : **Nov./Dec.-16(R13), Q1(c)**

Context Model

A context model refers to a model that describes the way in which context data is arranged and maintained. Its significance lies in providing efficiency in managing the context data.

In the process of software development, a decision is made regarding the system boundary and its environment at an early stage in the requirements elicitation and analysis process. With this, a simple architectural model is brought up. For example, following is an architectural structure representing the context of Bank ATM system.

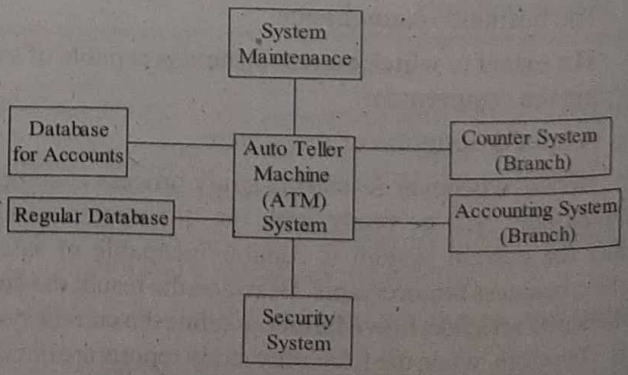


Figure: High Level Representation of Bank ATM System

Above figure represents only the block diagram which is nothing but a high level representation of the ATM system. The rectangle represents specific subsystem and embedding the name of the subsystem within it. The lines connecting these rectangles represent the collaborations existing between these subsystems. However, figure (1) depicts only the high level view of the system context. It does not express any details of other independent systems working with it. Other external systems should be considered since they may produce or intake the data produced to or from the current system. Their presence may have drastic impact on the current system and they may be directly connected to the current system.

Q18. Explain the need of behavioural models for software development.

Answer : (Model Paper-III, Q1(d) | Dec.-11, Set-1, Q5(a))

The behavioural model is needed in the software development process because,

- It depicts the dynamic behaviour of the system.
- Its first type, data flow models, depicts the data processing in the system where as its second type, state machine models, depicts the reaction of the system towards the events.
- It represents the states of analysis classes and the system as a whole.
- It uses input from scenario, flow oriented and class based elements.
- It identifies the states, events and actions of a scenario.
- It indicates how a software responds to external events.
- It provides the facility of reviewing the accuracy and consistency of the system.

Q19. Differentiate between data flow diagram and state transition diagram.

Answer : (Model Paper-II, Q1(d) | May-18(R15), Q1(d))

Data Flow Diagram

Data flow model depicts processing of data at each stage of system development. The model reflects the procedure used in data processing when considered along with analysis prospective.

State Transition Diagram

The state machine model is used to model the real time behaviour of a given system. It shows the states and events that result in change of the system's current states. However, it does not show the data flow within the system. Hence, state machine model consists of states, events (external/internal) and transitions between these states.

Q20. Explain about data models.

Answer : Nov./Dec.-17(R15), Q1(d)

We know that, database forms one of the essential aspects of any of the systems being developed (irrespective of their sizes). Also we have models to represent the processing of logical data belonging to these systems and such models are often referred as semantic data models. Most of the software development organizations today rely on entity relation attribute modelling for modelling database systems. The main aspects of these models are the entities, attributes and their relationships that exists among these entities. This gained wider recognitions even in object oriented database modelling.

PART-B ESSAY QUESTIONS WITH SOLUTIONS

2.1 SOFTWARE REQUIREMENTS

2.1.1 Functional and Non-functional Requirements

Q21. What are functional requirements of a software? Discuss in detail. Model Paper-I, Q4(a)

OR

"The functional requirements specification of a system should be both complete and consistent". Substantiate this statement with relevant examples. May/June-19(R16), Q4

Answer :

"Functional requirements of a system" describes the functionality of a system. The functionality of any system depends on the purpose of its manufacturing to a greater extent. It is less concerned about people using it and the organization developing it. Hence, it describes the detailed scenario of the system functionality.

An example of a system which depends on the purpose of its manufacturing to a greater extent, is the on-line pizza ordering system. In this example, the system should be capable of providing three major functional requirements i.e.,

- It should be easily browsable.
- The users should be capable of making right selection of pizza.
- At the end of ordering, the users should be provided with a unique id.

These functional requirements can also be considered under user requirements specification. The above requirements only provide an abstract information which can be further elaborated as,

- The system can provide facilities of opting favorite ingredients in the pizza.
- It should facilitate users to provide his/her credit card number in a secured manner.
- The system should accept user's delivery address in the form of house number street number or road number etc.

Inappropriate requirements often lead to unpredictable errors which can effect the end users in using the system. One of the reasons for generating such an error prone application, would be the omission of essential requirements for the sake of developer's own interest. This may also affect the project schedule thereby, increasing its cost deliberately.

Hence, the functional requirements are specified in terms of completeness and consistency. These two aspects though look quite similar, but they do have different meanings. Completeness in functional requirements specifications reflects that the specification includes all the requirements (small or large) essential for developing a given application. On the other hand, consistency ensures that the requirement supplied will not contradict with each other. Practically, it is not possible to have a system satisfying these two aspects. The reasons are,

- ctification of errors made in writing the specifications can be easily done for only large and complex systems.
- ne needs of the system stakeholders differ largely and are usually inconsistent.

---

iscuss the classification of non-functional requirements.

er : Nov./Dec.-12(R09), Q2(b)

n-functional Requirements of a System

Non-functional requirements address different issues of a system, like its availability, reliability, performance, security and various other related facts. Hence, they are critical than functional requirements. Using these requirements, a specimen can easily avoid the errors occurring due to mis-implemented functional requirements. For example, no one would prefer of purchasing a water heater if its operation is unreliable i.e., if the heater is shock prone. While dealing with non-functional requirements, one has to remember, it not only deals with the end product or the system (being developed) but also its process of manufacturing. There are many reasons for the outcome of non-functional requirements. They are listed below,

- Budget constraints.
- Inadequate user needs.
- If there is requirement of one system to be operable over other software/hardware.
- Strong privacy and safety constraints.
- Strong organizational policies etc.

Following is a hierarchical tree depicting classification of non-functional requirements,

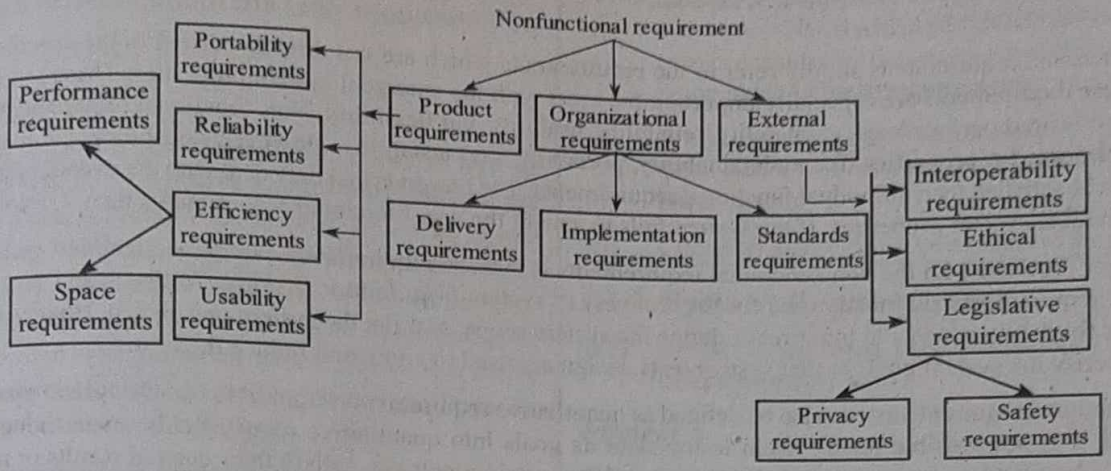


Figure: Hierarchical Tree Depicting Types of Non-functional Requirements

From the above tree, it can be concluded that there are three major nonfunctional requirements ascertained as product requirements, external requirements and organizational requirements respectively. Details on these requirements are as follows,

(a) Product Requirements

Behavioural aspects of the product are supplied in this case. For example,

(i) Usability requirements.

(ii) Expected or probable requirements.

(iii) Speed of execution of a system and its memory requirements.

(iv) The reliability values which can comprise to the system failures, if the failures are within the scope.

(b) External Requirements

Few external requirements are listed-below,

(i) Interoperable capabilities of the system.

(ii) Predefined deadlines or limits within which the system is required to operate.

(iii) The system operational capability which should be within the scope of a common user etc.

(c) Organizational Requirements

Requirements obtained from the organization's (both customer's and developer's) policies and procedures are,

(i) Implementation of methodologies used.

(ii) Delivery requirements.

(iii) Standards following which the current process is implemented etc.

Q23. Differentiate between functional and non-functional requirements with suitable examples.

Nov.-15(R13), Q5(a)

OR

Differentiate between functional and non-functional requirements.

Answer :

Nov./Dec.-16(R13), Q4(a)

Functional Requirements

For answer refer Unit-II, Q21.

Non-Functional Requirements

For answer refer Unit-II, Q22.

Q24. Explain the problems with non-functional requirements.

Answer :

Non-functional requirements simply refer to the requirements which are not directly related to the specific functions of the system. These requirements are especially important for users with the emergent system properties. These properties include specifications of desired performance, availability, reliability, usability and flexibility. Non-functional requirements are also important for developers for properties like maintainability, portability and testability. However, non-functional requirements are more difficult to be satisfied than individual functional requirements. The reason is that they deal with the overall system properties rather than individual system properties. If the system fails to satisfy the non-functional requirements then it becomes unusable.

The major problem with the non-functional requirements is that the requirements cannot be identified easily. Users indicated that these requirements are mostly used for the recovery of system from failure, fast user feedback and ease of use. These create problems for the developers to interpret or define the system scope, making the system unsatisfied. However, it is difficult to objectively verify the system goal, so that system tests design are used to count and reduce the errors.

Non-functional requirements can also be defined as quantitative requirements required to test the system goals objectively. But in practical, it is not possible for a system to translate its goals into quantitative requirements. Even though quantitative specification is possible, the users can not understand that these requirements can satisfy their desired results or not. Moreover, the cost of verifying the quantitative non-functional requirements may go beyond the expectations of the users.

Non-functional requirements may also have problems that need interaction with other functional and non-functional requirements. These problems may be related to the memory constraints, programming languages and cost etc.

Q25. Define the metrics for specifying the non-functional requirements.

Answer :

Following six metrics are defined to measure the non-functional requirements quantitatively.

- Property : Speed
  Metric : It is measured in terms of screen refresh time, user/event response time, number of transactions being processed per second.
- Property : Portability
  Metric : (i) Percentage of statements directly dependent on the target system.
  (ii) Total number of target systems.
- Property : Ease of use
  Metric : (i) Total number of help frames required.
  (ii) Time spent in training.
- Property : Robustness
  Metric : (i) Estimation of chances that data gets corrupted during the time of failure.
  (ii) Percentage of events which may account for failures.
  (iii)Time required in order to start the system after failure.
- Property : Size
  Metric : (i) Count of RAM chip utilized.
  (ii) Memory in K bytes
- Property : Reliability
  Metric : (i) Availability
  (ii) Rate of occurrence of failure
  (iii)Mean time to failure
  (iv)Chances of unavailability.

These metrics may be used fruitfully during system testing in order to ensure that the system meets all its specified non-functional requirements.

2.1.2 User Requirements

Q26. Discuss in detail about the user requirements.

Answer :

User Requirements

User requirements include both functional as well as non-functional details. The details should only provide the external behavioural aspects of the system. Hence, it should not include its implementation details which may frustrate the nontechnical users. The details should be written in terms of diagrams, forms, tables etc., to a major extent. Apart from this, there may be certain other problems which may evolve due to the usage of natural language in writing these specifications. Few of these problems are listed below,

Problem 1: Requirements Amalgamation

This problem usually arises when several requirements are clubbed together and are specified as one.

Problem 2: Lack of Clarity

When the requirements are not specified in a precise and unambiguous way, then clarity is said to be lacking.

Problem 3: Requirements Confusion

This problem usually arises whenever the developers do not specify a clear distinction between non-functional requirements, functional requirements, design information and system goals.

Moreover, while writing user requirements, one has to remember that it should not be included along with the details of the system requirements. The user requirements should be precise. If it covers large amount of data then the software developer cannot include solutions to the frequently occurring problems and it becomes difficult to read. Hence, it should provide only the essential facts, while avoiding the unnecessary details. Whenever any such details are provided, a supporting reason (behind its inclusion) should be associated. This reason describes its importance and it can be considered whenever suitable changes are made to the system.

Following are certain important, principle guidelines recommended to be followed while preparing the user requirements.

(i) It is a good practice to highlight the useful information. Highlighting can be done by either making the data italic, underline, bold or by changing its colour.

(ii) The user requirements documents should not get mixed up with the system using the information.

(iii) While writing the requirements, it is a good practice to follow standard notations throughout the document. For example, the requirements can be in a bold format while the reason of its inclusion in normal text etc.

(iv) Further, in order to make the document more realistic, the source from which the given reason is acquired can also be associated with a particular reason. This provides ease in considering the source when suitable changes are being made to these requirements.

(v) While specifying the requirements, there should be clear distinction between the mandatory and desirable requirements.

(vi) Computer jargons must be avoided to the possible extent.

2.1.3 System Requirements

Q27. What is system requirements? How would you specify them?

Answer : Model Paper-II, Q4(a)

System Requirements

'System Requirements' is often the initial step in the system designing process. They provide the implementation details of the user requirements. Hence, they are more detailed than user requirements. The data within the system requirements should be complete as well as consistent.

System requirements should only consider the description of the operational as well as behavioral aspects of the systems. It should not include the implementation as well as the design details of the system. But, whenever large and complex systems are considered the design details are avoided in the system requirements. There are many reasons supporting this occasion. Few of them are quoted below.

- If the system is distributed in nature (i.e., the system supporting interoperability), the design of a given system should be focused on the system requirements.
- In order to ensure that the current system satisfies non-functional requirements, system architecture need to be considered.

However, there many other factors which can create complexities while documenting the system requirements. The fundamental factor is the usage of natural language while documenting these requirements. They not only make the text difficult to understand, but they are often confusing. Following is a list of reasons supporting the illustration.

(i) As the requirements are specified by a developer in his own notion, it rests with the end user to properly make the distinction between the available requirements from the given volumes of text. This is because, given requirement may be grammatically analogous to other available text or given requirements may look similar but refer to two different facts etc.).

(ii) It is often impractical to associate the requirements under different side headings. Hence, a single requirement will drive through the entire system requirements set.

(iii) If the writer is addicted to using certain or desired phrases on different occasions, then readers may misunderstand the concept. This may deviate them from the actual requirements. Sometimes these facts are analyzed during final phases of the software development. On those occasions the costs of development will eventually rise. When these problems were analyzed by the scientists, they concluded to use a language (while writing the requirements) which can be easily grasped by all the individuals involved. This provides a full opportunity to the writers to express their views in the most appropriate manner. Using the language, the requirements are framed in a well planned format with suitable models used based on the requirements. The structure can also have models exchanging dialogues and other perspectives. Following are the set of notations which can be used for the requirement specifications,

Specifying System Requirements

Notation : Graphical Notations

Illustration : This refers to representation consisting of certain graphical objects along with appropriate text inscribed at suitable locations. Good examples of such notations are SADIT, use case diagrams and sequence diagrams.

Notation : Structured Natural Language

Illustration : This notation facilitates the writers to utilize certain predefined formats as well as templates while expressing a scenario in the form of requirement.

Notation : Mathematical Specifications

Illustration : These notations use the mathematical concepts like sets or finite state machines. Even though these notations minimize customer - contractor arguments, customers hesitate to accept it as a system contract.

Notation : Design Description Language

Illustration : In this case, the writers can utilize certain programming languages, in narrating the given requirements in terms of an operational model, if the model reflects only the abstract notion of the requirement. Now-a-days the approach is rarely used.

Q28. Explain about structured language specifications for writing system requirements.

Answer :

Structured Natural Language Specifications

Structured Natural Language was initially proposed by Heninger with an intention of providing certain standards while writing the system requirements for an aircraft software system. According to him, structured natural language is a decent style of writing system requirements which privileges the writers with well defined standards. It also helps in curbing writers independency of using their own visualizations. Hence, there will be high degree of expressibility but this expressibility can be adhered with equal measures of uniformity.

Structured natural language helps in decreasing the number of words used in describing the context of the system. It allows to look forward for usage of templates whenever necessary. Also focus on some of these requirements (which remains extremely essential), structured natural languages highlight them. Hence, in other words, the structured natural language expresses the system requirements in certain standard forms,

While dealing with these form-based writing of system requirements, following illustrations must be considered,

(i) There should be significant description corresponding to a function or a given entity.

(ii) Significant description corresponding to the inputs of the system as well as the source producing them. Also the description of outputs and the entities using them.

(iii) Illustrations related to the actions to be performed.

(iv) Illustration corresponding to generation of side effects while performing a given operation.

(v) Illustration on initial, as well as final conditions existing before and after a given function, if the applications consisting of functions.

Example

Consider an example of narrating different consequences observed during an ATM withdrawal using a sequence diagram.

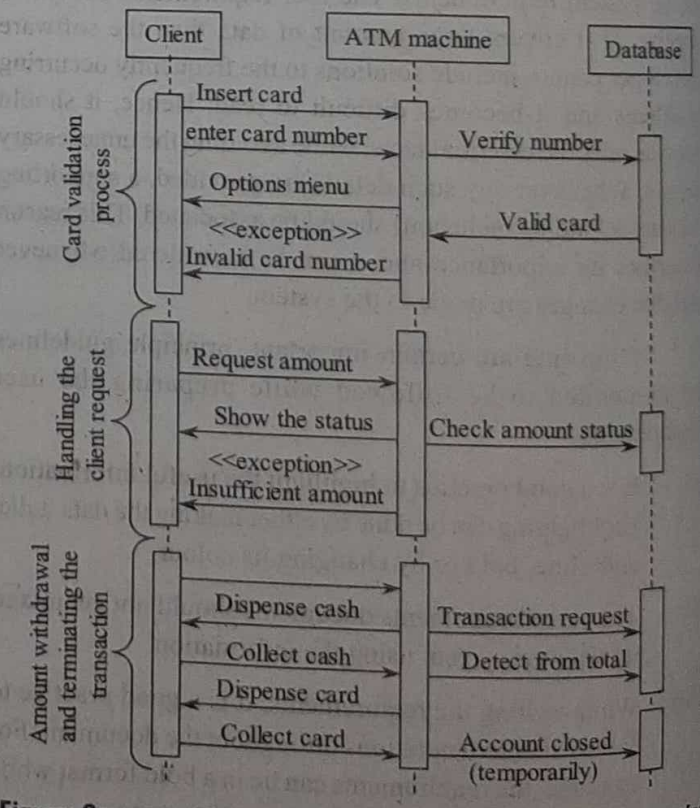


Figure: Sequence Diagram Depicting ATM Withdrawal Transaction

Q29. What is the purpose of domain analysis? Explain how it is related to the concept of requirements patterns?

Answer :

Domain Analysis plays an important role in specifying whether the system can function properly or not. This is because domain requirements focus on the system's application domain and not on the user's and system's requirements. While performing requirement engineering, the analysis pattern is performed repeatedly across several applications with respect to a particular business domain. If analysis pattern can be defined or categorized in a way that helps in identifying and reusing the pattern, then it is possible to create an analysis model. The impact of such a creation is that the use of reusable design pattern and executable software component increases. This inturn enhances the time-to-market and decreases the cost associated with the development of the pattern.

Domain Analysis contains the information required for recognizing, defining and categorizing the analysis pattern. According to Firesmith, software domain analysis is defined as a process of identifying, analyzing and specifying the common reusable requirements from a specific application domain. Firesmith also defined object oriented domain analysis as a process of identifying, analyzing and specifying the common reusable capabilities present in a specific application domain. These capabilities are defined in terms of classes, objects and frameworks.

The specific application domain can be of wider range i.e., it basically ranges from avionics to banking or from multimedia video games to software embedded systems. The objective of domain analysis is to identify, create analysis classes, functionality that are mainly applied for making them reusable. The information about expert advice, customer surveys and existing applications are extracted from sources of domain knowledge and are provided as inputs to domain analysis. Once the given inputs are processed, the outputs generated are class taxonomies, reuse standard and domain languages. These outputs are inturn supplied as input to domain analysis model. The main purpose of using domain knowledge is to identify reusable objects across the specific application domain.

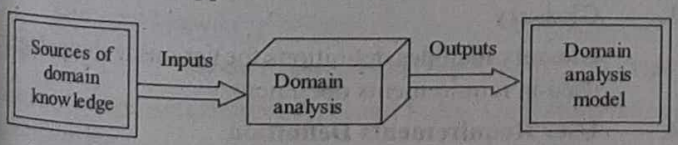


Figure: Domain Analysis Process

The entire process of domain analysis is characterized based on the context related to the object oriented software engineering. This process can be applied not only to the software engineering model but also to the traditional object oriented design models. The following are the steps carried out while performing the domain analysis process,

1. The domain that needs to be analyzed must be defined.
2. The items retrieved from sources of domain knowledge must be categorized.
3. The sample of application defined in the domain must be collected.
4. Every individual application in the sample is analyzed. Based on this, analysis class must be defined.
5. An analysis model for the class must be developed.

2.1.4 Interface Specification

Q30. Explain the importance of interfaces in a multi-system environment.

Answer :

Interface Specification

Software systems must be able to work with systems that are already implemented and installed. Thus, interfaces between them must be precisely specified. Specifications related to interfaces must be included in the requirements document as early as possible.

Types of Interfaces

1. **Procedural Interfaces**

   Application Programming Interfaces (API) are the procedural interfaces. Any service offered by a program or sub-system can be accessed by invoking an interface procedure.

2. **Data Structures**

   A sub-system may pass data structures to another sub-system. To describe these structures, Java or C++ automatically generated descriptions or graphical data models are used.

3. **Representations of Data**

   The way data is represented, for example, bit ordering, can be used as an interface. Most of the real-time embedded systems use these interfaces. Moreover, Ada also supports them.

2.1.5 The Software Requirements Document

Q31. Explain the structure of software requirements document. Nov./Dec.-17(R15), Q4(a)

OR

Explain how a software requirements document is structured.

Answer : (Model Paper-III, Q4(a) | March-17(R13), Q5(b))

Software Requirements Document

Software requirements document also called as software requirements specifications directs what the software engineering team must develop. It comprises of precise forms of customer requirements and a detailed illustration on system requirements. Software requirements are a combination of customer as well as system requirements specifications, which can be achieved in one of the following three ways,

(i) Both customer as well as system requirements specifications can be integrated to form one document.

(or)

(ii) Customer requirements specification can be provided in the initial phase or at the introduction of the document. This step is followed by detailed system requirement specification.

(or)

(iii) If the application is complex and probably large then system requirement specifications can be included at separate locations or document. Usually, in each software requirements document the details of large sets of stakeholders involved in the development are included. Following is a list of stakeholders and illustrations related to their involvement in the development,

(a) System Customer and End User

These are the people to whom software is being delivered. They are authenticated to supply their requirements and also read the current document (software requirement document) in order to ensure that the (document) is built in accordance to their needs.

(b) System Managers

System managers usually consider the document to decide on the cost of development and to plan out the consequences of entire project.

(c) Development Team or Engineers

They usually decide on the tasks to be accomplished.

(d) Testers

They use the documents in order to develop suitable test cases.

(e) Maintenance Engineers

They consider the document as a mode to understand as well as generalize the relationship existing between various modules of the system.

In general, the level of details to be provided in the document depends on the type of system being developed and also on the process being implemented on this occasion. By considering various aspects, IEEE has launched following format for generating the software requirements documents,

1. Introduction

- Purpose of requirements document
- Product scope
- Definitions, acronyms and abbreviations
- References
- Overview of rest of the requirements document.

2. General Description

- Product perspective
- Product functions
- User characteristics
- General constraints
- Assumptions and dependencies.

3. Specific Requirements

- Functional
- Non-functional
- External interface requirements
- System functionality and performance
- Logical database requirements
- Design constraints
- Emergent system properties
- Quality characteristics.

4. Appendices

Here, the details of the application are included. The fundamental issues which can be addressed in this regard are hardware requirements as well as database requirements.

5. Index

Indexes are usually included to provide ease in locating appropriate entities. The most fundamental index is given in alphabetical order. There are many ways to represent indexes and any of them can be followed.

The above mentioned format is a standard notation for developing the software requirement specification. But it is assumed that the above format defines only general characteristics when considered along with the organization developing the software. Hence, the above format is further matured and certain new specifications are added to it. This helps in directly applying preparation of SRS at the organization level. Hence, the improved format is given below,

1. Preface

Here, the fundamentals of the document are defined (say) the current version and its history. Also the details include the reasons for introducing the current documents along with the summary of changes made to the previous version in bringing up the current version.

2. Introduction

Summary of details under this side heading are given below,

- The need for the system
- The major functionality within the system and with other systems.
- The way the given system remains compatible with the business objectives of the organization involved in developing the given system

3. Glossary

Glossary includes definitions for various technical terms used in requirements document.

4. User Requirements Definition

Here, usually the services delivered by a given system to the users are specified along with non-functional requirements of the system. This illustration is usually made in most elucidating style using the formal language constructs, diagrams or user understandable notations. To make the specification more valuable, it is adhered to several templates, diagrams and highlighted data wherever necessary.

5. **System Architecture**

Here, usually the overall system architecture is presented in an abstract format. This includes the distribution of modules throughout the architecture along with the relationships existing between them. The components (within the architecture) which are used more than once, should be represented along with the same signature (representation) so that they can be recognized easily.

6. **SRS or System Requirement Specification**

In this case both functional as well as nonfunctional requirements are described in a more detailed manner.

7. **System Models**

One or more models depicting the relationship between the system, its environment and its components, along with their vicinity where they are highlighted. The most commonly used models in this case are semantic, data, data-flow and object models.

8. **System Evolution**

In this case usually the assumptions based on which the current system is brought up is summarized. Also, the probable changes which can be made to the system on the account of hardware trends and future user needs are also addressed.

9. **Appendices**

Here, the details of the application are included. The fundamental issues which can be addressed in this regard are hardware requirements and database requirements.

10. **Index**

Indexes are usually included to provide ease in locating appropriate entities. The most fundamental index is given in alphabetical order. There are many ways to represent indexes and any of them can be followed.

Q32. Describe desirable characteristics of a good software requirement specification document. What is the role of SRS in Software Engineering?

Dec.-19(R16), Q5(a)

OR

Describe five desirable characteristics of a good software requirement specification document.

(Refer Only Topic: Desirable Characteristics of a Good Software Requirement Specification)

Answer : **(Model Paper-I, Q4(b) | Nov./Dec.-18(R16), Q4(a))**

Desirable Characteristics of a Good Software Requirement Specification

A good SRS document must posses the following characteristics,

(i) **Concise**

The SRS document must be upto the point, unique and consistent because lengthy and irrelevant descriptions decrease the readability and increase the possibility of errors.

(ii) **Organized**

The SRS document must be well-organized because it becomes simple to understand and one can make the necessary changes easily as per the customer requirements.

(iii) **Black-box View**

The SRS document should represent only the physical behaviour of the system rather than describing its implementation issues. In other words the system which is being developed must be considered as a black box and its external behaviour should be defined. Due to this reason, the SRS document is also known as black-box specification of a system.

(iv) **Conceptual Integrity**

A good SRS document must possess conceptual integrity so that the contents of the document can be understood easily by a reader.

(v) **Response to Undesired Events**

The document must describe the system responses to exceptional conditions.

(vi) **Verifiable**

All requirements of the system must be verified in order to determine whether all the requirements are met during implementation or not. If any feature is not verifiable then it should be mentioned separately in the "goals of implementation" section of the SRS document.

Role of Software Requirement Specification in Software Engineering

Software Requirement Specification (SRS) document is the final outcome that comes after problem analysis. But, both problem analysis and SRS are carried out simultaneously due to which these activities overlap with movement from both the activities to the other. The information regarding the SRS is obtained from analysis therefore, specification activity follows the analysis activity.

The formal modeling performed in problem analysis is not treated as SRS as it lays stress on the problem structure than the external behaviour. The things like modeling of user interfaces, minor issues, performance, design constraints, recovery etc., are rarely modeled. But these things should be specified in SRS, these all constraints are needed to be modeled to design a system easily and clearly. On the other hand, it is difficult for some structures to be translated into external behaviour specification due to their limited use.

2.2 REQUIREMENTS ENGINEERING PROCESS

Q33. Discuss various steps in requirements engineering. What are the work products of engineering the requirements? Dec.-19(R16), Q4(a)

OR

Give an overview of various steps in requirements engineering process. May-18(R15), Q5

OR

Discuss about principal requirements engineering activities and their relationships.

(Model Paper-II, Q4(b) | March-17(R13), Q5(a))

Answer :

Steps Involved in Requirements Engineering

The primary objective of requirements engineering process is to design and maintain a document related to system requirements. The principal requirements engineering activities and their corresponding relationships are shown in the following figure.

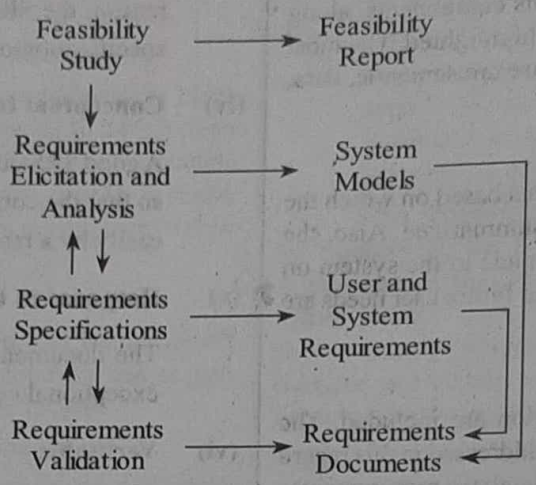


Figure: Various Stages of Requirements Engineering Process

During the requirement engineering process, four subprocesses are utilized. Their names and their relative illustrations are given below,

1. **Feasibility Study**

For answer refer Unit-II, Q34.

2. **Requirements Elicitation and Analysis**

For answer refer Unit-II, Q35.

3. **Requirements Specifications**

The knowledge gained during problem analysis becomes the starting point of requirements specification. Here, the main objective is to properly state the requirements address the issues, representations, tools, specification languages, etc. The redundant information and the knowledge generated from analysis is properly organized and described in the requirement specification activity.

4. **Requirements Validation**

This is the final step in the requirements engineering process, where all the proposed and refined requirements are verified. This concludes that, they are laid in accordance to the user specifications. Hence, the end product of this phase will be the requirements documents.

In short, all the above mentioned activities can be partitioned under three activities.

- Discovery
- Documentation and
- Checking.

However, when the developer visualizes the practical conditions, the scenario changes drastically. Human requirements change depending on the time factor, the organization purchasing the software changes, the hardware on which the given software being implemented changes etc. Hence, the software developed should be well acquainted to satisfy all these anomalies.

Work Products of Engineering the Requirements

The work products generated from each step of engineering the requirements are,

1. Feasibility report
2. System model
3. SRS document.

Spiral Model

This is an alternative method of the requirements engineering process. The spiral model is partitioned under three main activities i.e., requirements specifications, requirements validation and requirements elicitations. The time required in completion of given process depends on the type of system under development and also the process implemented.

The following figure describes the spiral model of requirements engineering process.

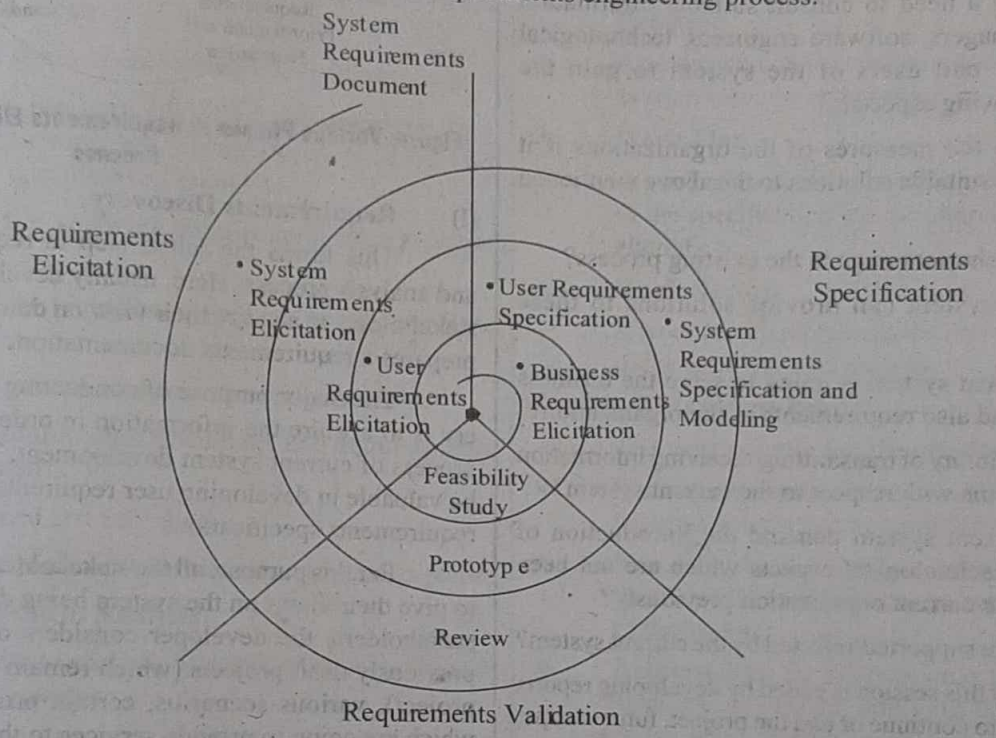


Figure: Spiral Model of Requirements Engineering Processes

As shown in the above figure, the core of all processes will be the determination of business, functional and non-functional requirements specification. Hence, the developers usually pay higher attention at this stage. While certain other intermediate processes become closer to the final phase like review.

2.2.1 Feasibility Studies

Q34. What are the feasibility studies for requirements engineering process?

Answer :

Nov./Dec.-17(R15), Q4(b)

Feasibility Study Process

The feasibility study forms the initial step in the requirements engineering proces. The developers can start with feasibility study process whenever the information related to the following aspects are obtained.

- The business requirements.
- The extent to which current system is capable of satisfying the requirement.
- A brief description of the system.

Hence, whenever feasibility study process is completed, the developers will be ready with the documents specifying whether the current system is capable/incapable of satisfying complete business requirements. Based on the result, the decision on whether to set aside, move further or refine the current system is taken. Therefore, when the feasibility study reports are developed, it should contain data related to the following aspects.

- Does the current report satisfy the business aspects of the organization involved in developing the system?
- Is it possible to implement the system using the current technology within given cost and schedule constraints?
- Can system be integrated with certain other well defined systems?

Hence, once the information related to the above mentioned facts are obtained, half of the requirements are said to be completed.

Now, there is a need to consult suitable information sources such as managers, software engineers, technological experts and finally end users of the system to gain the information on following aspects,

- What will be the measures of the organizations if it does not have suitable solutions to the above mentioned aspects?
- What are the shortcomings in the existing process?
- How a new system can provide solutions to these problems?
- How the current system is going to serve the business necessities and also requirements of the organization?
- Is there a possibility of transmitting/receiving information to other systems with respect to the current system?
- Does the current system demand the introduction of certain new technological aspects which are not been utilized by the current organization previously?
- What items are supported/rejected by the current system?

Hence, finally this session is ended by developing reports and suggestions like to continue or end the project, future scope, schedule, cost, etc.

2.2.2 Requirements Elicitation and Analysis

Q35. What are the activities of requirements elicitation and analysis? Explain.

(Model Paper-III, Q4(b) | Nov./Dec.-16(R13), Q5(a))

OR

Elaborate on requirements elicitation and analysis process in detail.

Answer : **May/June-12, Set-2, Q6**

Requirements Elicitation and Analysis

Requirements elicitation and analysis technique is a stage that comes next to the feasibility study process. In this process, the developer usually consults various customers and discusses related issues such as,

- Hardware requirements
- Details related to the resultant application
- Details related to the performance of the systems respectively.

Different Phases in Requirements Elicitation and Analysis Process

Following is a diagram depicting various stages during the requirements elicitation and analysis phases of requirements engineering processes.

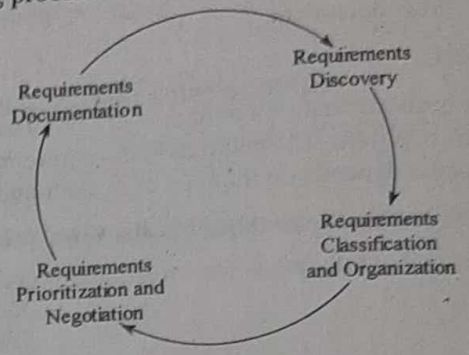


Figure: Various Phases in Requirements Elicitation and Analysis Process

(i) Requirements Discovery

This forms the initial step in requirements elicitation and analysis process. Here, usually developer consults various stakeholders to acquire their view on domain requirements and prepares a requirements documentation.

The major purpose of conducting requirements discovery is to acquire the information in order to proceed with the process of current system development. This information will be valuable in developing user requirements as well as system requirements specifications.

For this purpose, all the stakeholders are brought together to give their views on the system being developed. Apart from stakeholders, the developer considers other sources such as previously dealt projects (which remain similar to the current project), various scenarios, certain prototypes, the systems which are going to provide services to the current system, etc.

(ii) Requirements Classifications and Organization

As all the requirements are collected and documented, they are organized orderly. For this purpose all the related requirements are clubbed and several groups are framed. Often these groups are referred as clusters.

(iii) Requirements Prioritization and Negotiation

Whenever the developer consults various stakeholders to gain their requirements, often these requirements conflict. The developer assigns priorities to these requirements. Therefore, the main purpose of this phase is to provide priorities to these requirements (often referred as prioritisation) and also to resolve conflicts between them (often referred as negotiation).

(iv) Requirement Documentation

The final phase is to prepare requirement documents and proceed forward for next rotation of the cycle.

The entire scenario illustrated above reveals the consequences of requirements elicitation and analysis only during single rotation of this cyclic process. The cycle makes further rotations with an attempt to improve the process in each rotation. Hence, in this way the process proceeds.

Q36. What is the goal of requirements analysis phase? Give reasons why the requirements analysis phase is difficult one.

Answer : Nov.-15(R13), Q4(a)

Goals of Requirements Analysis

The various goals of requirement analysis are as follows,

1. To understand the problem for which the software system is to be developed.
2. To focus on what can be achieved from the system but not on how the system achieves a set of planned test activities.
3. To possibly reduce the communication gap between the developer and client.
4. To define the scope of the software to be developed.
5. To transform the user specified requirements into unambiguous, traceable, complete, consistent and stakeholder - approved requirements.
6. To produce the software requirement specification document.

The result of the requirement analysis phase comprises the following,

1. Definition of stakeholder-approved requirements.
2. A system requirements document and requirements traceability matrix.
3. A set of planned test activities.
4. An approval to proceed to the next phase i.e., design phase.

Reasons for Requirement Analysis

The requirements analysis is considered to be difficult due to the following reasons,

1. It has content related to the communication. Due to this likelihood in occurcence of misinterpretation or misinformation becomes high.
2. It has high ambiguity, which leaves the programmer in state of confusion, because the statements are repeated multiple times.
3. Here the errors go undetected in early stages and gets amplified in development phase.
4. It is difficult to detect and correct design errors that comes up in requirement analysis and they can only be detected after identifying its source.

Q37. What is a view point? Explain its significance. Discuss various types of view points.

Answer : Dec.-11, Set-2, Q5

View Point

View point is a system engineering concept. It refers to a loosely coupled and locally managed object which contains partial knowledge about the application domain and the process of software development. View points are used to organize requirements and requirements elicitation process.

A viewpoint is a combination of following slots (parts),

1. **Style**

   A style is a scheme of representation. The view point uses this scheme to represent whatever it could see.

   Example

   Data flow analysis, entity-relationship attribute modelling, equational logic etc.

2. **Domain**

   A domain defines the component of the "world" indicated precisely in the style and can be observed by the view point.

   Example

   An elevator-control system may have the domains such as user, elevator and controller.

3. **Work Plan**

   A work plan decides in which conditions the contents of the specification can be altered and how they can be altered.

4. **Work Record**

   A work record represents the account of the current state in the development process.

Significance of View Point

The view point is significant because,

- ❖ It aims at specific concerns related to the system. For example, a security view point aims at security concerns.
- ❖ It provides the conventions, rules and languages that can be used in the construction, presentation and analysation of the views.
- ❖ It encapsulates the information in the form of slots. These slots are helpful in, applying the general knowledge in the wide range of problems, representating specific knowledge regarding particular problem.
- ❖ View points also help in describing the current state of specification with respect to the development activities.

Types of View Point

There are three different types of view points. They are,

(i) **Interactor View Points**

Interactor view points represent the people or systems that use the system directly. They provide detailed system requirements such as system features and interfaces.

Example

In a banking system, the bank's customers, bank's account database are the interactor view points.

(ii) **Indirect View Points**

Indirect view points represent the stakeholders that use the system indirectly. They provide higher-level organisational requirement and constraints.

Example

In a banking system, management of the bank, bank security staff are indirect view points.

(iii) Domain View Points

Domain view points represent the domain characteristics and constraints that have a great impact on system requirements.

Example

Standards created for inter-bank communications.

Advantages of View Points

There are two major advantages of considering view-points. They are,

- Viewpoints can be used to design a framework which can act as a major tool for resolving conflicts between requirements acquired from different stakeholders.
- Viewpoints themselves can be used in classifying various viewpoints.
- The specifications of the system which may have its direct effect on the system under development.
- The nonfunctional requirements as well as the business sources of the system under development.
- The standards based on which the system is being developed, and also the rules being implemented.
- The specimens delivering the services to the system and also the specimens to whom the current system is going to deliver its service.
- The customer expectations on the system and also by valuing other features, like the scope of the organization which may exist, once the system is delivered.
- The requirements aspects of developers responsible to maintain, manage and develop the current system.

Q38. Discuss about VORD method in detail.

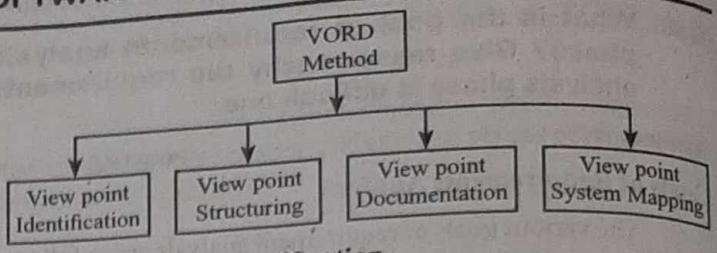
Answer :

VORD Method

The Viewpoint Oriented Requirement Definition (VORD) is a method of requirements elicitation and analysis proposed by Kotonya and Sommerville. It has been designed as a service framework for requirements discovery and analysis. It also includes different steps in order to translate the analysis into an object-oriented system model. A viewpoint-oriented analysis has a major ability to identify several perspectives and provides a framework for discovering process in the requirements designed by various stakeholders.

Stages of VORD Methods

There are four stages of the VORD method, which helps in the requirements discovery and analysis. The first three stages of the VORD method are concerned with viewpoint and service identification. The last stage deals with the mapping and transformation of analysis. These stages of VORD method can be defined as follows,



1. Viewpoint Identification

This stage involves discovering viewpoints in order to receive system services. It also identifies the specific services that are provided to each viewpoint. It analyzes the system services based on which the viewpoints can be designed. It is essential in requirements discovery and analysis to identify the viewpoints and services.

2. Viewpoint Structuring

This stage involves grouping similar viewpoints into a hierarchy form. In the first stage, the similar viewpoints based on the specific services are identified. Then these similar viewpoints can be arranged in a structural form for easy understanding and analysis.

3. Viewpoint Documentation

This stage involves defining the description of the identified viewpoints and services. In this stage, documentation is prepared in order to obtain the information of identified viewpoints and services. This information is very essential in the requirements discovery and analysis.

4. Viewpoint System Mapping

This stage involves transforming the analysis of the identified viewpoints and services to an object-oriented design model.

Q39. Define brainstorming. Explain where it is used with an example.

Answer :

Brainstorming

Brainstorming is defined as a group creativity technique that is designed to generate a large number of ideas as an optimal solution to a problem. During requirements elicitation and analysis, brainstorming may be used to understand the requirements in an effective and efficient manner. It is particularly helpful when a problem with many issues arises, for generating new ideas, creative thinking and new opportunities to develop highly creative solution to a problem.

Brainstorming has become an effective technique in requirements, which may be mostly used by the companies. It helps to develop creative thinking, new ideas and sharing opinions by the participants. During brainstorming sessions, there should be no criticism of ideas. Here, all participants try to open up possibilities and breakdown wrong assumptions about the limits of a problem. Here every participant has equal rights to share their own ideas whether it is useful or not. The brainstorming technique based on participants can be categorized as,

(i) Individual brainstorm

(ii) Group brainstorm.

(i) **Individual Brainstorm**

An individual brainstorm or participant will more intensionally produce a wider range of ideas than a group brainstorm. This is because he or she may not have to worry about other participant's egos or opinions. Therefore, he or she may make their own ideas freely to the highly creative solution of a program.

(ii) **Group Brainstorm**

Group brainstorm or participants can be very effective as it uses the experience and creativity of all members of group. When an individual member reaches to their limits for an idea, another member's creativity and experience can get the idea to the next stage. Therefore, group brainstorm can solve problems in more effective and efficient manner.

For example, a company has an idea to launch a new product but they are unaware of promoting the product to the public. Marketing team members may use brainstorming innovative marketing ideas that will ensure that the product will be successful in the market.

Q40. Write the structure and standard form of every viewpoint template and service template forms.

Answer :

The structure of standard form for each viewpoint template and service template are shown below,

Viewpoint Template		Service Template	
1.	**Reference:** The name of a viewpoint on which reference is given.	1.	**Reference:** The name of a service on which reference is given.
2.	**Attributes:** The properties or attributes provide specific information of viewpoint.	2.	**Rationale:** Issue based on which service is provided.
3.	**Events:** A reference to a set of event scanners, which describe how the system effects to viewpoint events.	3.	**Specification:** A reference to a list of service specifications, which may be expressed in different notations.
4.	**Services:** A reference to a set of service descriptions.	4.	**Viewpoints:** A reference to a list of viewpoint names, which receive the service.
5.	**Sub-viewpoints:** The names of the viewpoints on which references are provided.	5.	**Non-functional Requirements:** A reference to a set of non-functional requirements, which help to define the service.
		6.	**Service Provider:** A reference to a list of system objects, which provide the service.

Q41. Define use cases. Explain their purpose.

Answer :

Use Cases

Use case is the essential feature of the UML notation used in object oriented system models. Use cases act as a scenario in the real world concept for describing the requirements. Analysis of a system use case is used by requirements engineers to describe the interaction of a user with the proposed software system.

A use case is simply defined as a set of sequences in which each sequence represents the interaction of actors and the system. An actor represents a set of roles and responsibilities towards the system. It may be a human, machine or information system that is external to the system model.

Example

An example of use cases for the library is shown in the figure,

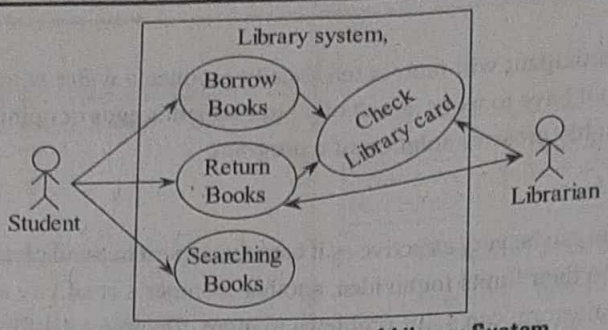


Figure: Use Cases and Actors of Library System

Purpose of Use Cases

The purposes of the use cases are as follows.

1. They specify the desired behavior of a system or a part of a system.
2. The jobs on what the user needs to do with the system rather than what they want the system to do.
3. They assist in the development of software system process by providing mechanisms for making clarity and consistency.
4. They help to validate the architecture and to verify the system as it evolves during software system development.
5. They provide common understanding with the system end user's and domain experts.

Q42. Using your own knowledge of how an ATM is used, develop a set of use cases that could be used to derive the requirements for an ATM system.

Answer :

Operation of ATM System

An ATM system works as follows,

Step1

To begin the transaction, the customer inserts his ATM card in the ATM system.

Step2

An ATM system reads the card and asks the customer to enter Personal Identification Number (PIN).

Step3

System checks for card authentication by checking the entered PIN.

Step4

If the PIN is valid, then system displays a transaction menu consisting options like cash withdrawal, transfer funds, balance enquiry, etc.

Step5

For instance, if the customer selects the cash withdrawal then system requests the customer to enter the amount.

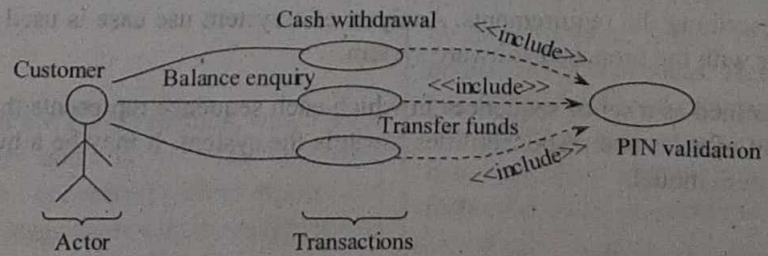
Step6

Then, the system gives out the entered amount with a print of receipt. Finally, customer collects the card by closing the transaction.

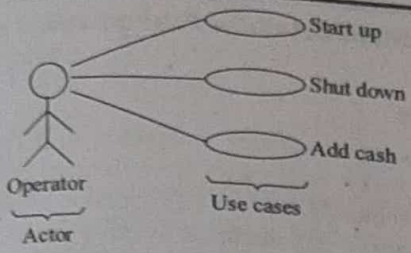
Step7

Customer takes the cash, bank card and the receipt.

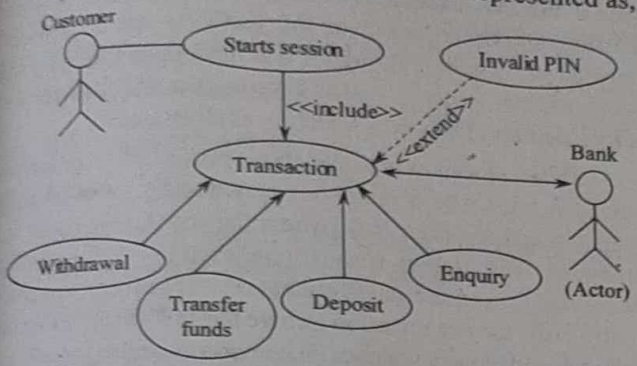
Use Case Model for ATM



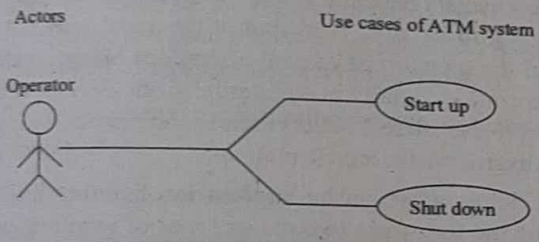
Here <<include>> is dependency relationship which specifies the source use case.



Set of use cases for ATM systems are represented as,



Here, <<extend>> is dependency relationship which specifies target use case.



These set of use cases are elaborated as follows,

(a) **System Start Up**

When the operator turns the switch to ON position, the system starts up. Then the operator will be requested to enter some amount existing in the cash dispenser, thereby establishing bank connection.

(b) **System Shut Down**

When the operator turns the switch to OFF position, the system shuts down. Now, the operator can remove the deposited envelopes and review the cash. Thereby disconnecting the bank connection.

(c) **Session Use Case**

When the customer places the ATM card into the machine, a session starts. A session is aborted by pressing CANCEL key.

(d) **Transaction Use Case**

When the customer wishes for a transaction from the menu display, a transaction use case is started.

(e) **Cash Withdrawal Transaction Use Case**

When the customer requests to withdraw from the existing account then a withdrawal transaction is started.

(f) **Deposit Transaction Use Case**

When the customer requests to deposit the amount, then deposit transaction use case is started.

(g) **Transfer Transaction Use Case**

When the customer requests to transfer, then transfer transaction use case is started.

(h) **Balance Enquiry Transaction Use Case**

When the customer requests to enquire the current balance then balance enquiry transaction is started.

(i) **Invalid PIN**

When the bank states that transaction is not approved due to invalid PIN, then an invalid PIN extension is started and the transaction is aborted by using CANCEL key.

Q43. Define a scenario. Write a sample use-case scenario for an article downloading in the library system.

Answer :

Scenario

Scenario describes the way user interacts with a software system. The information extracted from the scenario is used by requirement engineers so as to gain better understanding about the actual system requirements. Such scenarios that provide the description of system requirements are considered as an essential part of agile method like extreme programming. They are generally used to add details to an abstract description of the system. This description basically relate to the interaction activities. There are multiple ways of representing a scenario each depicting different sort of information at different abstraction level.

Scenario is initiated by first specifying the abstract description of interaction. Later, when elicitation is carried out, details are added in order to develop a complete description of interaction. Scenario includes the information about,

(i) User and system expectations (when the scenario initiate)

(ii) Flow of events

(iii) Other activities performed at same time

(iv) Situations that may go wrong and procedure for handling them

(v) State of system (when scenario terminates).

Scenario for Article Downloading in the Library System

This scenario describes the procedure of how a user of LIBSYS downloads a personal copy of an article in a medical journal.

1. **Initial Assumptions**

(a) User logs on LIBSYS system

(b) The location where the journal that consists of copy of desired article is found.

2. Normal Flow of Events

(a) The article that is to be copied is selected by the user.

(b) Once the selection is made, the user is allowed either to enter the subscriber information associated with the journal or to specify the payment method.

(c) After specifying the required details, the user is prompted to fill in a copyright form. This form contains the information about the transaction. Once the form is filled up, it is submitted to LIBSYS system.

(d) The LIBSYS system performs verification process after receiving the copyright form. If the details entered by users are correct, then the form is approved.

(e) Once the form is approved, PDF version of the selected article is downloaded and stored in the working space of LIBSYS. After the article is downloaded completely, the user is informed about the completion.

(f) If the user wants to print the article then he/she selects a printer that prints the copy of the article. While selecting the printer, if the user set the flag 'print-only', then the article is deleted from the working area.

3. Activities Carried Out at the Same Time

Downloading of other articles.

4. Situations that can go Wrong

(a) The request for downloading an article may be rejected, if the user enters incorrect details in the copyright form even though it has been resubmitted for correction.

(b) The payment method may be rejected by the system.

(c) While downloading the article, the system may get crashed due to which the session terminates. Despite of the termination, the cost of article is deducted from the user's account even if the article is not downloaded completely. The situation may get worsen if 'print-only' flag is selected, while printing the article. This is because the article is not available in the LIBSYS working area and the user must initiate from the beginning.

5. State of the System

(a) User is still logged on the LIBSYS system.

(b) If the flag is set to 'print-only', then after the completion of scenario, the article is deleted from the LIBSYS working area.

Use-case Scenario

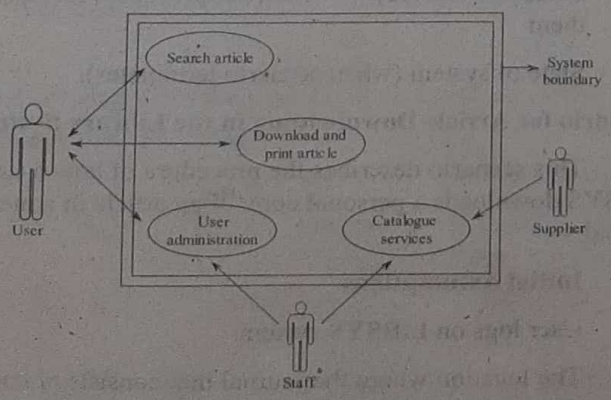


Figure: Use-case Diagram for Article Downloading

Q44. Explain in detail the Ethnography.

Answer :

Ethnography

Whenever a software is developed, it should initially gain the full fledge acceptance from the customers and at the same time it should be of very high quality. The developers of the software never pay attention towards the social as well as organizational factors, where the application has to be implemented. For this reasons, it will exercise its services throughout its life span. Often many of the software fail to provide its expected service even though it was developed through certain organizational experts, tested thoroughly and also reviewed (before delivery) by certain technical champions.

Hence, looking onto the above prospects, an observation strategy is used which motivates in acquiring social as well as organizational requirements (where the software is going to be deployed). This strategy is usually referred to as ethnography. It is a method for requirements elicitation and analysis. An analyst usually visits the organization (where the software is going to be deployed) and closely watches its scenario as well as the activities of its staff. Hence, in this way a report is generated which reflects the real time requirements of the organization as well as its staff. This usually happens as a given staff member may be very well known to his own work but at the same may not be able to reveal the scenario of overall operations being conducted in the organization. Finally in ethnography, the developer calls the end users to the office, collects their requirements and expresses certain diagrammatic representations.

Ethnography can be applied in obtaining information regarding the following two major types of requirements.

- Requirements that are gathered by observing how the people actually work.
- Requirements that are gathered by knowing about activities of other people.

Ethnography can be combined with prototyping. It informs about the prototype development. This reduces the number of prototype refinement cycles. Prototyping identifies the problems in the ethnography and discusses them with the ethnographer, who must find the solutions during the next phase of the system study.

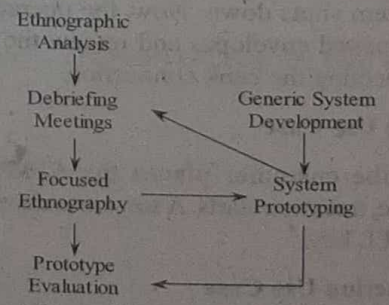


Figure: Different Steps Observed while Combining Ethnography along with Prototype

Limitations of Ethnography

- It cannot be used to acquire the domain requirements.
- It cannot be used to dictate modifications or improvements to be applied to the system.

Q45. Write in detail about interviewing. Explain with examples.

Answer : May-13(R09), Q3

Interviewing

Interview is a requirement elicitation technique conducted by requirement engineering team, who questions system stakeholders regarding the existing system being used and the new system that needs to be developed. Interviews (formal or informal) are considered as a part of requirement engineering process. They can be classified into following two types.

(i) Closed interviews

(ii) Open interviews.

(i) Closed Interviews

In this type of interview, the system stakeholder provides answers to a predefined set of questions.

(ii) Open Interviews

In this type of interview, the requirement engineering team discusses different issues with the stakeholders. The purpose of this interview is to gather significant information about the requirements of the stakeholders. In contrast to closed interviews, open interviews doesn't consists of any predefined set of questions.

The main intent of conducting interviews is to get better understanding about,

- The activities performed by stakeholders.
- Interaction between the stakeholder and the system.
- The problems encountered while using the existing system.

Disadvantages of Interviewing

The major drawback of interview is that, it doesn't provide information about the requirements from the application domain. This is because of the following reasons,

- It is difficult for application specialist to explore the requirements of application domain without using the terminology specific to that particular domain.
- The terminology is easily misunderstood by requirement engineers as specialists use the terminology in precise manner.
- Stakeholders might have prior knowledge about the domain. Therefore, they may find it difficult to explain the domain requirements or they might think it as a fundamental concept that doesn't need any explanation.

In addition to this, interviews fail to extract knowledge about the requirements of organizations as well as their constraints. This is because of the strong relationship between the stakeholders in the organization. Many people prefer to discuss issues (political and organizational) that may have impact on the requirements. Interviewers basically describe the abstract structure but not the actual structure of the organization.

Characteristics of Interviewing

The following are the characteristic features possessed by effective interviewers.

- Interviewers are open-minded i.e., they listen patiently regarding the new ideas/requirements of the stakeholders.
- If the requirements specified by stakeholders do not have any impact on organization's performance, then the interviewers manage to make the stakeholders understand the situation.
- Interviewers ask questions and allow the interviewee to initiate the discussion. Interviewers talk to the interviewee in a friendly manner by using defined context instead of using general terms.

Examples of Open Interviews

1. What types of problem do you encounter with existing process?
2. What measures you suggest to improve the existing process?
3. Mention any three factors that demotivated you from working in this environment?

Examples of Closed Interviews

1. Are you familiar with AutoCAD 3.X?
2. Do you need training in VisualPro 6.0?
3. Can you take additional responsibility of business analyst?

2.2.3 Requirements Validation

Q46. Discuss the significance of requirements validation and also discuss various requirements validation techniques. Nov./Dec.-12(R09), Q3(a)

OR

What is requirements validation? Explain different requirements validation techniques.

Answer : Model Paper-I, Q5(a)

Requirements Validation

The requirements validation activity uses the requirements specification. It examines the specification for consistency, omissions, errors and ambiguity of requirements.

It also examines that the work products produced as a consequence of requirements engineering conform to the standards established for the process, the project and the product.

The main objective of validation activity is to check that SRS presented is of good quality. Usually, there is a possibility of the following four errors to occur in SRS.

(i) Omission

(ii) Inconsistency

(iii) Incorrect fact

(iv) Ambiguity.

(i) Omission

When a requirement remains unspecified in SRS, omission error occurs. This error can have direct impact on the external completeness of the SRS. This is because omitted requirement might be related to either system's behaviour, its performance, constraints or any other factor.

(ii) Inconsistency

Inconsistency can occur due to the incapability of SRS to specify the requirements as defined by the client, due to the unfavorable conditions or if the requirements are not compatible with the environment in which the system will operate.

(iii) Incorrect Fact

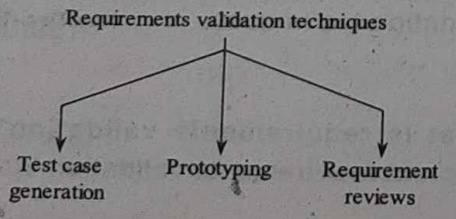
Such kind of errors occur due to the incorrect recording of facts in SRS.

(iv) Ambiguity

These errors usually occur due to multiple meaning possessed by a requirement that leads to inappropriate results. Therefore, the objective of validation activity should be the correction of such errors occurring in SRS along with the provision of good quality SRS.

Requirements Validation Techniques

Requirements validation techniques are divided into three categories,



1. Test Case Generation

In this technique, various test cases are generated for the requirements.

The nature of tests depends on following factors,

(i) If the tests are carried out as part of the validation, then it leads to requirement problems.

(ii) If the tests are very difficult to design, then obviously there is a difficulty in implementing requirements.

2. Prototyping

In this technique, requirements can be checked by availing an executable model or earlier model. This model is tested to determine whether it meets the requirements of end users and customers.

3. Requirements Reviews

For answer refer Unit-II, Q47.

Q47. Who should be involved in a requirement review? Draw a process model showing how a requirements review might be organized.

Nov./Dec.-18(R16), Q5(b)

Answer :

Requirements Reviews

Requirements reviewing is a process of inspection. In this case, a reviewing team is formed which consists of members belonging to development staff and certain end users. They usually review all the requirements before concluding that these requirements are free of inappropriate anomalies or unexpected requirements etc. In general, requirements reviewing process is divided into two stages, informal and formal reviews.

(i) Informal Reviews

This is just a direct process where developers call all the stakeholders and discuss their requirements on the given system.

(ii) Formal Reviews

In this stage, the developers presents all the requirements accepted in the informal reviewing process and collect relative opinions of the end users. Then the developers move forward to check the two important properties i.e., completeness and consistency issues of requirements documents. Reviewers also have to check the following factors,

1. Verifiability

To know whether the requirements that are clearly specified can be tested or not.

2. Comprehensibility

To know whether the system end-users or procurers can understand the requirements in a proper way.

3. Traceability

To know whether the base of requirement is defined in a simple and clear manner. By considering the base of the requirement, the impact of change on the requirements can be evaluated. Hence, traceability is considered as one of the most important factors because any changes made to the requirements can effect the rest of the system to be evaluated.

4. Adaptability

To know whether the requirements are adaptable or not i.e., whether they can change without having large impacts on the rest of the system requirements.

Reviewers must detect the errors, omissions, conflicts and contradictions that occur in the requirements. After the problems have been identified, they must be recorded in the review report. After the end of requirement review, it is the responsibility of procurers, users and developers of the system. to provide solutions to these identified problems.

Process Model of Requirement Review

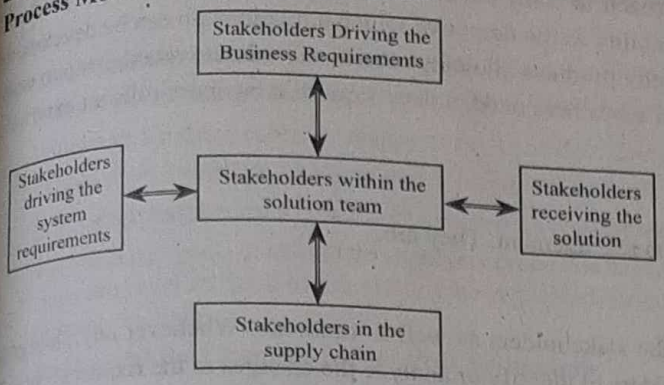


Figure: Process Model of Requirement Review

2.2.4 Requirements Management

Q48. What is requirements management? Why is it needed?

Answer :

The requirements for the software development of a large system keeps changing. This happens because the software development team usually addresses only the wicked problem, while implementing the requirements. But this session does not ends here. Moreover, whenever the end user installs and gets complete command over the software, they demand for further improvements. Following are the major issues favouring the above mentioned consequences,

- On implementing the software, it may happen that, several business as well as technical aspects of the organization get changed, certain new hardware may also get introduced, the end users may demand interfacing of current system developed to accept interfacing with various other systems, new rules or policies of the organization may be implemented.
- Sometimes it may happen that, the organization involved in developing the software adheres certain restrictions on the software with an intention to keep intact the budget and other financial requirements of the organization. But these restrictions will not be pleasing to end users working with the system.
- The organization accepting the system may have large set of man power and each person with different vision and requirements.

Requirement management addresses these issues in general, requirements management appeals to customize the software with an ability to understand and effectively control the changes to system requirements.

Initially, all the independent requirements are taken into account. Later, the possible requirements which may arise on the advent of these requirements are analyzed and accessed.

A document referred to as change proposals is prepared. These change proposals are matched with the system to check their compatibility.

Requirements management phase should start, just after the completion of requirement documentation.

Q49. Differentiate between enduring and volatile requirements. Give examples for each.

Answer :

During the software process, the requirements of the large software systems have been constantly changing. These requirements must be evolved to meet the needs of software systems. The process of developing the software requirements concentrates on software capability, business objectives and other business systems. These requirements have been changing over a time for developing a large system based on the needs of a user, whose purpose is to change the requirements.

In an evolution process, there are two different kinds of classes in the requirements process. They are,

1. Enduring system requirements
2. Volatile system requirements.

1. Enduring System Requirements

Enduring system requirements are defined as static requirements which cannot be changed over a period of time. They focus directly on the domain of a system as well as indicate the main activity of the organization. These requirements are helpful for the development of software systems for a long time.

Example

A college may always need stable requirements involving faculty, students, courses and library. These requirements may not change over a period of time and may be obtained from domain models, which provide entities and relations of an application domain.

2. Volatile System Requirements

Volatile requirements are defined as unstable requirements which can be changed over a period of time. It is used when the system is developing or the system has started its operational process. These requirements are frequently changed based on the needs of a user or a software system.

Example

Government releases the educational system policies that can be changed over a period of a time.

Q50. What is traceability? Explain different types of traceability.

Answer : Model Paper-II, Q5(a)

Traceability

Traceability is one of the essential activities of good requirements management, which helps to identify similar requirements. It is used to ensure that the right products are being developed at each phase of the software development life cycle. It helps to reduce the effort required to determine the impact of requested changes in order to trace the progress of the software development.

In a simple manner, traceability can be defined as an approach to trace the relationship between each unique product level requirement and its sources. The IEEE standard defines traceability as the degree of which a relationship can be developed between two or more products of the development process, especially products allowing master-subordinate relationship to one another. For example, a product requirement might be traced from a business need, a user request, a business rule, an external interface specification or some other sources.

Types of Traceability

There are three different types of traceability in requirements management. They are,

(i) Data Source Traceability

It helps to provide source information of requirements to the stakeholders as well as rationale. Whenever any changes occur, this source information become useful to the stakeholders to identify or analyze the changes in the requirements.

(ii) Requirement Traceability

It helps to determine whether the changes in requirements are necessary or not. It enables to analyze the number of requirements, which may be affected by the requirements changes. It also provides relation to the similar requirements within the requirements document.

(iii) Design and Implementation Traceability

It helps to determine the changes made in requirements that can be affected on the system design and implementation. It provides relation to the requirements of the system design, where these requirements are implemented.

Q51. Explain about high level design matrix.

Answer :

High Level Design Matrix

A matrix that correlates any two documents requiring many-to-many relationship between them for determining the completeness of the relationship is called as high level design matrix. It is also known as traceability matrix or traceability table. It is used with high-level and detailed requirements of the software product that matches with the high-level design, detailed design, test plan and test cases.

Traceability

For answer refer Unit-II, Q50, Topic: Traceability.

Construction of Traceability Matrix/High-level Design Matrix

A table is drawn with rows and columns such that the identifiers of one document are placed in the very first row and the identifiers of the other document in the very first left column. Upon identifying the relationship between two items, intersecting cell is marked. Finally, the number of markings are added for each row and for each column which indicates the relationship between the two items. Absence of the marking between a row and a column indicates that there is no relationship between the corresponding items.

Example

Important aspects of the system →

	Req 01	Req 02	Req 03	Req 04	Req 05	...	...	Req 11
Aspect 01		✓	✓					✓
Aspect 02	✓			✓				
Aspect 03		✓		✓				✓
Aspect 04	✓							✓
Aspect 05		✓	✓	✓	✓			
⋮								
⋮								
Aspect n	✓							

→ Requirements of the system

Figure: Traceability Matrix

Q52. Explain about requirements change management in detail.

Answer : Dec.-11, Set-4, Q3(b)

Requirements Change Management Process

Requirements change management is applied to all specified changes to the requirements.

Following are three essential stages of requirements change management process,

1. **Conducting Problem Analysis and Change Specification**

In this stage, requirements problem or sometimes a requirements proposal is initially considered which is then thoroughly analysed. The main perspective behind this analysis process is to validate a given requirements problem. Later, the outcomes of this stage are reverted back to the entity who requested for current change. Before proceeding to next step, a refined change proposal is made.

2. **Analysis of the Changes and its Cost Estimation**

In this stage, the effect of making changes to the requirements and the cost incurred are assessed.

The effects are analyzed by considering the general information of the system requirements and traceability information. The cost can be evaluated by considering the facts like expected improvements possibly applied to the requirements and to the designing and implementation of the current system.

3. **Change Implementation**

In this case, all the above assumptions are brought into reality. The change implementation begins with necessary alterations made to the requirements and also to the system design and implementation. Later, the requirements documents are partitioned into number of sections. The main purpose of such partition is that, alterations can be easily made.

Advantages

There are two main advantages of requirements change management. They are,

- All the proposals made in favour of change management can be equally treated.
- Changes can be applied to requirements documents in well organized manner.

Important Activities of Requirement Change Process

Some of the important activities of requirement change process are,

(a) Conducting problem analysis and change specifications

(b) Analysis of changes and its cost estimation

(c) Change implementation.

Q53. Elaborate on requirements management planning in detail.

Answer : May/June-12, Set-3, Q3

Requirements Management Planning

Requirements management planning is the first step in the requirement management process. Planning is necessary for managing the requirements in all the stages of the project. Some of the activities that should be planned during requirement management stage are,

1. **Requirements Identification**

Each of the requirements must be identified individually so that they can be referred by other requirement later and also to be used in traceability assessments.

2. **Change Management Process**

For answer refer Unit-II, Q52.

3. **Traceability Policies**

The traceability policies show the relationship that exists between the requirements and the system design and requirements which has to be recorded and maintained.

4. **Case Tool Support**

These are the tools (ranging from specialist requirements management system to spreadsheets/simple database systems) that are used to process large amounts of information about the requirements. There are many relationships that exist between the requirements and system design. These relationships correspond to the links between the requirements and the reasons behind the proposal of these requirements. The impact of the changes on other requirements and system design must be traced when they are proposed.

For remaining answer refer Unit-II, Q50.

2.3 SYSTEM MODELS

Q54. What is system modeling? Explain the process of creating models and the factors that should be considered when building models.

Answer : (Model Paper-III, Q5(a) | Nov./Dec.-18(R16), Q7)

System Modeling

System modeling refers to a process that creates a set of graphical representations which specify or explain the various aspects of the system. The graphical representations help the developers and stakeholders to better understand the system.

Factors that Needs to be Considered when Building a Model

The following are the restraining factors that has to be taken into consideration by the developer for creating a model.

1. Assumptions
2. Simplification
3. Limitations
4. Constraints
5. Preferences.

1. Assumptions

It allows a model to display the associated problems in order to reduce the possible numbers of permutations and variations.

For instance, consider the representation of a 3d human forms wherein system engineer imposes certain limitations on movement of human body in such a way that the range of input domain and processing will be limited.

2. Simplifications

It allows the development of a model at specified time.

For instance, consider a system engineer that not only model the requirement of a service organization but also understand the flow of data which the service order is produced.

Despite the fact that the service orders can be obtained from various origins but only two sources are classified by the engineers. These two sources are internal demand and external request.

These sources generate an easy method for input partitioning that is necessary to provide service order.

3. Limitations

It is generally used to restrict the system.

For instance, consider an aircraft avionics system that is developed for future purpose. As the aircraft is a two engine model, the monitoring domain for actuation will be designed in such a way that it contains atmost two engines along with its corresponding redundant system.

4. Constraints

It represents the way in which the model is developed and focuses on the approach that is considered while implementing the model.

For instance, consider a 3-D rendering system that uses a single G4-based processor. The complexity of problems should be forced to fit within the processing bounds by the processor.

5. Preferences

It specifies the desired architecture the is required for the entire data, functions and technology.

Q55. Give an overview of various system models.

March-17(R13), Q4

OR

Explain the following system models,

(a) Object models

(b) Structured methods.

(Refer Only Topics: Object Models, Structured Methods)

Answer : Nov./Dec.-17(R15), Q5

(a) Context Models

For answer refer Unit-II, Q56.

(b) Behavioral Models

For answer refer Unit-II, Q58.

(c) Object Models

For answer refer Unit-II, Q66.

(d) Structured Methods

For answer refer Unit-II, Q67.

(e) Data Models

For answer refer Unit-II, Q63.

2.3.1 Context Models

Q56. What is meant by context model? With a neat diagram explain the context model of ATM system. Model Paper-I, Q5(b)

OR

What is context model? Describe the importance of context model. Dec.-19(R16), Q4(b)

OR

Elaborate on context models.

Answer : Nov./Dec.-12(R09), Q3(b)

Context Model

A context model refers to a model that describes the way in which context data is arranged and maintained. Its significance lies in providing efficiency in managing the context data.

In the process of software development, a decision is made regarding the system boundary and its environment at an early stage in the requirements elicitation and analysis process. With this, a simple architectural model is brought up. For example, following is an architectural structure representing the context of Bank ATM system.

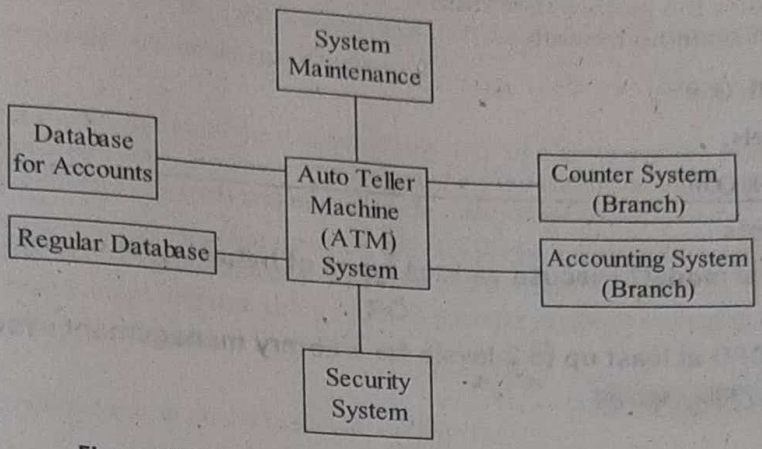


Figure (1): High Level Representation of Bank ATM System

The figure (1) represents only the block diagram which is nothing but a high level representation of the ATM system. The rectangle represents specific subsystem and embedding the name of the subsystem within it. The lines connecting these rectangles represent the collaborations existing between these subsystems. However, figure (1) depicts only the high level view of the system context. It does not express any details of other independent systems working with it. Other external systems should be considered since they may produce or intake the data produced to or from the current system. Their presence may have drastic impact on the current system and they may be directly connected to the current system.

Hence, process model should be augmented with the proposed architectural model. The process model includes all the activities supported by the system. Following model is an example process model of equipment procurement.

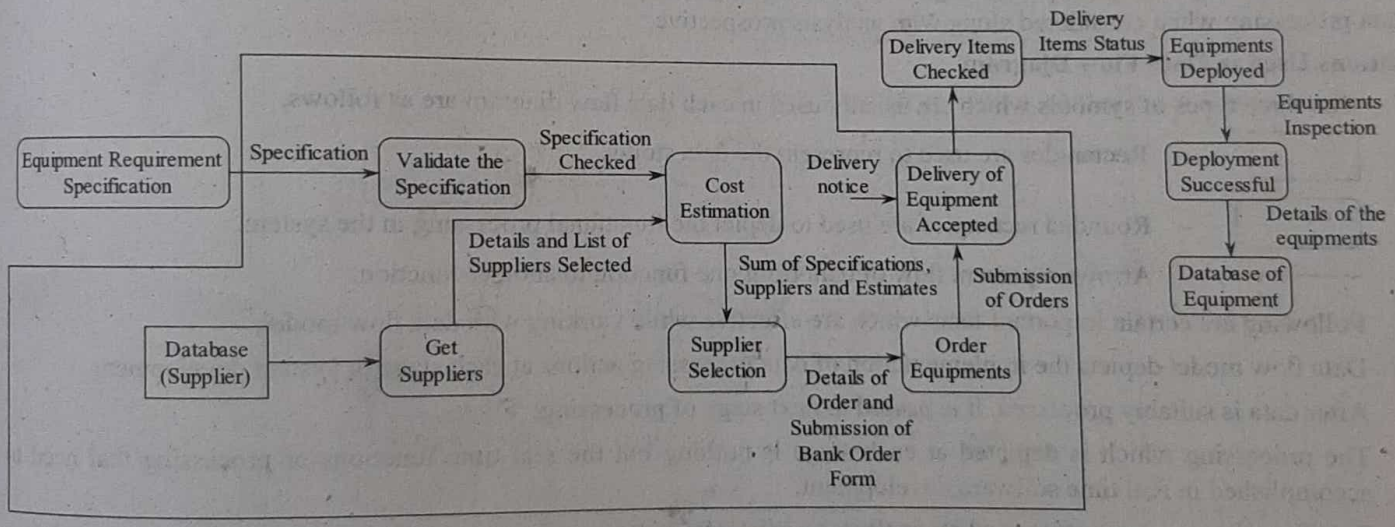


Figure (2): Process Model of Equipment Procurement

In figure (2), certain activities lie inside the system boundary, (as shown by the box) while the other activities lie outside the system boundary. Hence, when the computer support has been specified for the process model, certain decisions regarding the activities must be taken. The activities of the process model include the following.

(i) The equipments that are essential, need to be specified.

(ii) The suppliers need to be determined and selected.

(iii) The equipments should be ordered.

(iv) The ordered equipments need to be delivered.

(v) Finally, testing should be performed to check whether the equipments have been delivered or not.

Q57. Why is traceability an important aspect of requirement management? Why context system models are useful for requirements validation?

May-18(R15), Q4(a)

Answer :

Importance of Traceability

For answer refer Unit-II, Q50.

Importance of Context Models

For answer refer Unit-II, Q56.

2.3.2 Behavioral Models

Q58. What is a behavioural model? Discuss various types of behavioural models with examples for each.

OR

Draw the complete DFD at least up to 2-levels for a library management system.

(Refer Only Topic: Data Flow Model)

Nov./Dec.-18(R16), Q4(b)

Answer :

Behavioural Model

Behavioural model is a process of illustrating the entire functionality of the system. It provides the dynamic view of the system, where the changes in development of a software process are done dynamically. It is used to describe the behavioural activities of the system based on the states, events and time variant. Behavioural model is also defined as a function of specific events and time, which indicates how software will respond to these external events and state of changes during software development.

There are two kinds of behavioural models in software development which are,

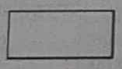
1. Data flow model
2. State machine model.

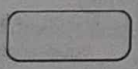
1. Data Flow Model

Data flow model depicts processing of data at each stage of system development. The model reflects the procedure used in data processing when considered along with analysis prospective.

Notations Used in Data Flow Diagram

The three types of symbols which are usually used in each data flow diagram are as follows,

 – Rectangles are used to represent the data stores.

 – Rounded rectangles are used to depict the functional processing in the system.

 – Arrows represent flow of data from one function to another function.

Following are certain important facts which are effective while working with data flow model,

- ❖ Data flow model depicts the implementation of data processing actions at each stage of system development.
- ❖ After data is suitably processed, it is passed to next stage of processing.
- ❖ The processing which is depicted at each stage is nothing but the real time functions or processing that need to be accomplished in real time software development.
- ❖ Data flow diagrams are often used by analysts to gain information on the scenario being described by these models.
- ❖ These are rather most simple and easy while developing.

Example

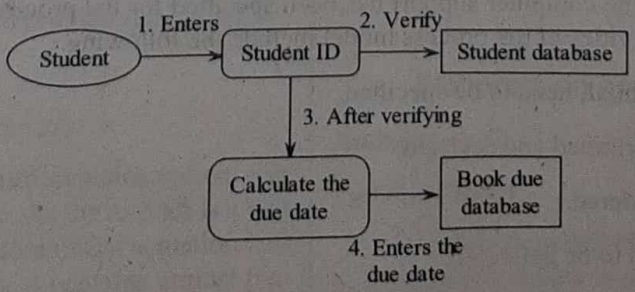


Figure (1): DFD of Library System

If a student wants to borrow books from the library then the following steps are performed,

(i) First, the student enters the student ID.

(ii) The student ID is verified by searching the details in student database.

(iii) If the verification is done successfully, then the due date on which the student must return the borrowed book is generated.

(iv) The due date value is entered into the books due database.

2. State Machine Model

The state machine model is used to model the real time behaviour of a given system. It shows the states and events that result in change of the system's current state. However, it does not show the data flow within the system. Hence, state machine model consists of states, events (external/internal) and transitions between these states.

State machine model assumes that, at a given instance, the process may remain at any state and whenever it receives an event it proceeds to the next state. For example, consider a computerized pressure controlling system in a reactor. The pressure in the reactor may be lowered from current pressure value when an operator performs an operation. Hence, lowering of pressure is an example of event generation.

Example

Consider a simple microwave oven. It contains several buttons to,

(i) Set power

(ii) Set time

(iii) Start the system and

(iv) Cancel the operation.

The steps required to cook food in microwave oven are,

(i) Set the level of power to be consumed. It can be either full or half.

(ii) Set the time required for the food to be cooked.

(iii) Press the start button to start cooking.

The state machine model of a microwave oven is shown below. It contains rectangular boxes that depicts states and arrows represent state transitions. Each state contains a small description within it as 'do' statement.

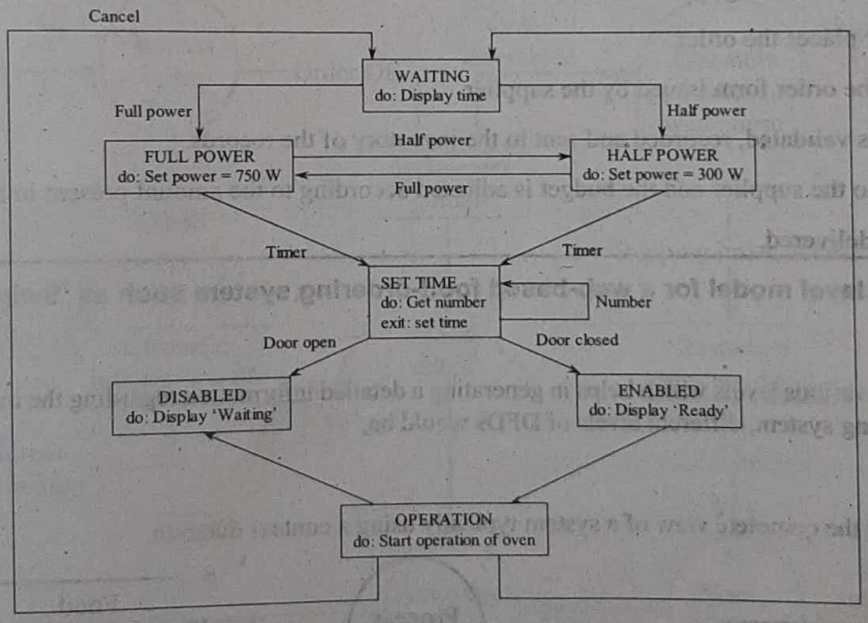


Figure (2): Microwave Oven's State Machine Model

Initially, the microwave oven will be in 'waiting' state. To use the microwave oven, firstly, the level of power to be consumed is set. It can be set either to 'full power' or 'half power'. The 'waiting' state will then be changed either to the full power or half power state. User is allowed to change the power from full to half power and half to full power at any time. After the power is set, the time needed to cook food is set in 'set time' state. Then, the door of the microwave should be closed. If it is not closed, then state will change from 'set time' state to 'disabled'' state. If it is closed, then it will be changed from 'set time' state to 'enabled' state. When microwave oven is in 'enabled' state, the start button can be pressed to start its operation. When this operation is carried out, it can either be cancelled or let to complete. In both the cases, the state of the microwave oven will change from 'operation' state to 'waiting' state.

Q59. What is data flow diagram? Draw a data flow diagram for order processing system.

Answer : Dec.-11, Set-3, Q6(a)

Data Flow Diagram

For answer refer Unit-II, Q58, Topic: Data Flow Model (Exclude Topic: Example).

Example

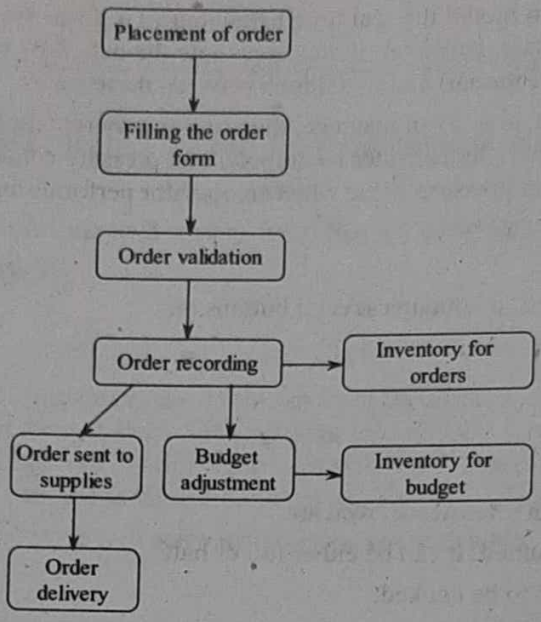


Figure: Order Processing System

The steps for order processing system are as follows,

1. First the customer places the order.
2. He/she then fills the order form issued by the supplier.
3. The signed form is validated, recorded and sent to the inventory of the records.
4. The order is sent to the supplier and the budget is adjusted according to the amount present in the customer's account.
5. The order is then delivered.

Q60. Draw a context level model for a web-based food-ordering system such as 'Swiggy".

Answer : May/June-19(R16), Q5(a)

A DFD consists of various levels which helps in generating a detailed information regarding the input, output and processing of input. For a food ordering system, different levels of DFDs would be,

1. Level 0 DFD

This level provides the complete view of a system typically using a context diagram.

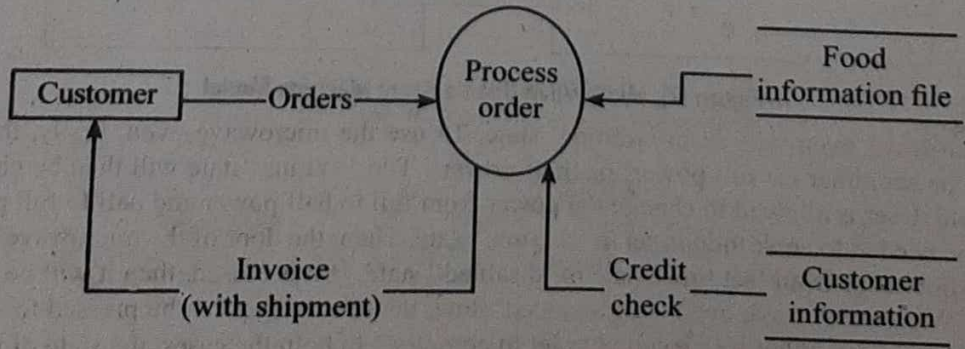


Figure: Level 0 DFD

2. **Level 1 DFD**

This level divides the system into number of processes.

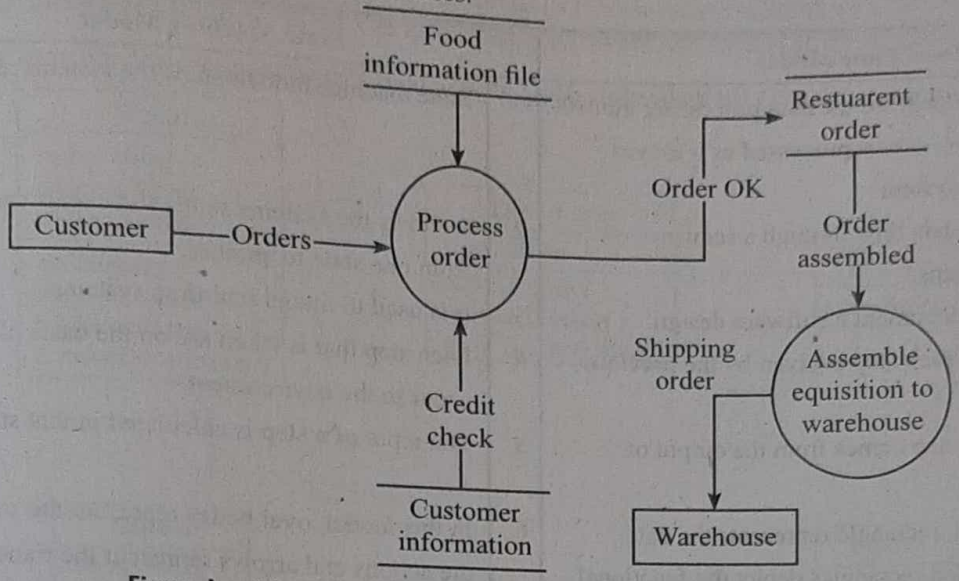


Figure: Level 1 DFD Which Explains the Present Food Ordering System

3. **Level 2 DFD**

This level describes each of the processes generated from the Level 1 of DFD.

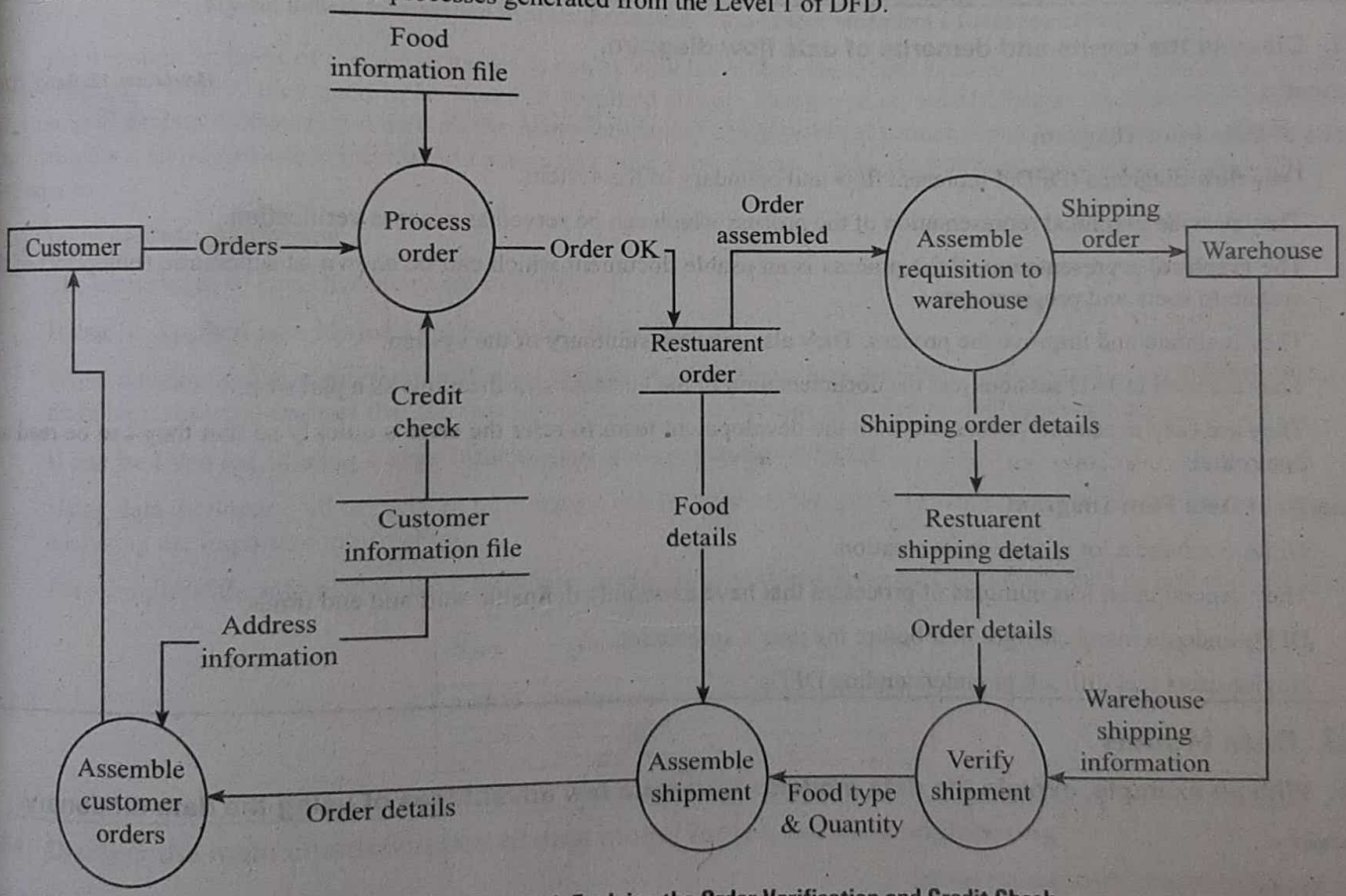


Figure: DFD Level 2 Which Explains the Order Verification and Credit Check

4. **Level 3 DFD**

This level describes the output generated from level 2 of DFD in detail.

Note that, the levels of a DFD do not explain about the internal structure and behaviour of a system. The process will be clear as the level of DFD is increased.

Q61. Differentiate between data flow and state machine models.

Answer : Dec.-11, Set-3, Q6(b)

	Data Flow Model		State Machine Model
1.	Data flow models are the data processing models that show how data is processed as it moves through the system.	1.	State machine models show the systems response to events.
2.	It shows the data flow through a sequence of processing steps.	2.	It shows the systems states and events causing transition from one state to another.
3.	It is used to document a software design.	3.	It is used to model real-time systems.
4.	The input of each step is given by the previous steps.	4.	Each step that is taken are on the basis of the events that occur in the environment.
5.	Input of one step comes from the output of another step.	5.	The input of a step is calculated in that step only.
6.	In this model, rectangle represents the data stores, rounded rectangles depict the functional processing of a system and arrows represent the data flow from one function to another.	6.	In this model, oval nodes represent the states that define the actions and arrows represent the transitions from one state to another.

Q62. Discuss the merits and demerits of data flow diagram.

Answer : May/June-12, Set-1, Q2(b)

Merits of Data Flow Diagram

1. Data flow diagrams (DFDs) represent flow and boundary of the system.
2. They provide graphical representation of the process which can be served as process verification.
3. The graphical representation of the process is an usable document which can be shown as schematic (blueprint) of the system to users and programmers.
4. They maintain and improve the process. They also give the summary of the system.
5. They are used in JAD sessions and the documentation of the business specifications as a part of it.
6. They are easy to identify errors and help the development team to refer the error's quickly so that they can be read and controlled.

Demerits of Data Flow Diagram

1. DFDs consume a lot of time in its creation.
2. They depend upon less numbers of processes that have extremely definable start and end times.
3. DFDs undergo many changes in it before the user's agreement.
4. Novice users feel difficult in understanding DFDs.

2.3.3 Data Models

Q63. With an example, explain the data models. Also state few advantages of using the data dictionary.

Answer :

Data Models

Database forms one of the essential aspects of any of the systems being developed (irrespective of their sizes). Also the models used to represent the processing of logical data belonging to these systems are often referred as semantic data models. Most of the software development organizations today rely on entity-relation attribute modelling for modelling database systems. The main aspects of these models are entities, attributes and their relationships that exist among these entities. This gained wider recognitions even in object oriented database modelling.

A new language referred as unified modelling language has acquired the position of entity relation attribute modelling. UML directly did not support the database modelling, but has spread its roots in encouraging models for semantic data modelling. UML adopted objects, classes and strategies in each of its models. Classes are represented as the entities of era model.

The following example depicts semantic data model using UML class notations,

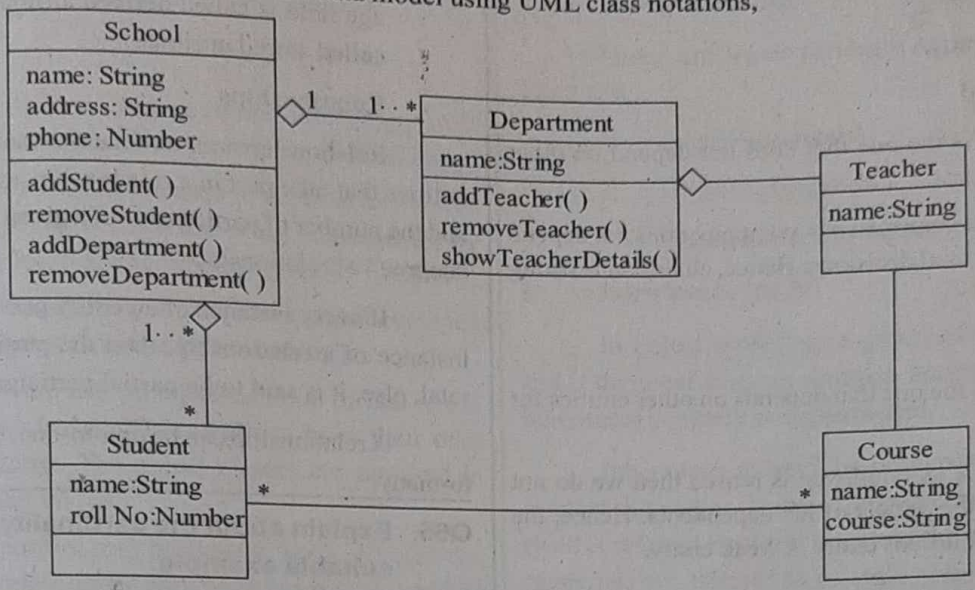


Figure: School Information System (Example of Data Model, Modelling a Database Schema)

On thorough analysis of the above model, it can be concluded that, the model exposes most of the details. As details are often a necessary aspect, they need to be stored in specified storage location in an orderly format. Such storage locations are often referred as data dictionaries where all the names are stored in alphabetical format. Apart from names, the dictionary can also maintain a variety of other information associated with these names. There are immense advantages of maintaining data dictionaries.

Advantages of Data Dictionary

The advantages of data dictionary are stated below,

- **It can be Applied as a Major Tool for Name Management**

While developing large enterprise software, usually the developers may introduce new name for a given entity. The data dictionary software ensures that, no two similar names exist because of which no confusion occurs.

- **It can be Used for Storing Large Information at Organizational Level**

Using data dictionary, all the related information can be stored at one place providing ease while adding, retrieving and analyzing the important information.

For example, if the information about the library system is to be stored in the data dictionary, then its probable field can be,

Name	Description	Type	Date
Authors	Description of entities	Attribute	17.07.07

Figure: Data Dictionary

Q64. Discuss the main characteristics of data model for requirement engineering.

(Model Paper-II, Q5(b) | May/June-19(R16), Q5(b))

Answer :

The main characteristics of data models for requirements engineering are,

1. Entities
2. Attributes
3. Relationships.

1. Entities

An entity is a real-world object that can be uniquely identified. There are two types of entities.

(i) Strong entity

(ii) Weak entity.

(i) Strong Entity

Strong entity is the one that does not depend on other entities.

For example, a chairman of a company does not depend on anyone for final decisions. Hence, chairman is strong entity.

(ii) Weak Entity

Weak entity is the one that depends on other entities for existence.

For example, if an employee is retired then we do not need to store the details of his dependents. Hence, the dependents (children) entity is weak entity.

2. Attributes

Each entity and relationship has a property called attributes. These attributes can be,

(i) Simple Attributes

These attributes are also called as atomic attributes which cannot be subdivided further. For example, the attributes like roll number and class of 'student' entity cannot be further subdivided.

(ii) Composite Attributes

An attribute which can be further divided into smaller components are called composite attributes.

For example, the attribute name can further be divided into first name, middle name and last name. Similarly, the attribute address can further be divided into house number, city, street, country.

(iii) Single-valued Attributes

Certain attributes take only a single value in all instances. For example, the age of a person is a single-valued attribute as man cannot have two age.

(iv) Multi-valued Attributes

Attributes that can have more than one value at a time for an instance are called multi-valued attributes. For example, the color of the product. A product might be multicolored in this case it takes more than one value at a time.

(v) Stored and Derived Attributes

Some attributes need not be stored, but can be derived from available other attributes.

For example, the total number of students in a class can be calculated by counting the number of student records. Similarly, the age of a student can be calculated, by subtracting the date-of-birth field from present date. The age field is called derived attribute and date-of-birth is called stored attribute.

3. Relationships

Relationship is an association among various entities. The entities that take part in a relationship are called "participants" and the number of participants of a given relationship are called "degree" of relationship.

If every instance of an entity participates in at least one instance of a relationship, then the participation is said to be total, else, it is said to be partial participation.

A relationship can be one-to-one, one-to-many or many-to-many.

Q65. Explain about the cardinality and modality with suitable example.

Answer : May-18(R15), Q4(b)

Cardinality

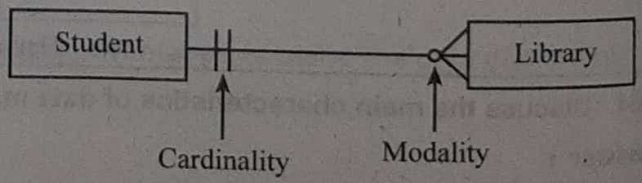
Cardinality states the occurrences that occur between the objects in a relationship. It shows 1:1 relationship between objects which means one object is related to only one other object. It represents the number of occurrences that occur between objects like 1: N relationship i.e., one object is related to many other objects. M : N relationship shows that many objects are related to many other objects. It also states the maximum number of object that can take part in the relationship.

Modality

Modality indicates that there is no compulsion for a relationship to occur between the objects by providing the integer 0. And if participation of the objects in the relationship is necessary it provides the modality as 1.

Example

Consider a library management system where a student borrows books.



In this example, a single student can borrow multiple books from library. This creates a one-to-many relationship resulting in cardinality. However, it is not mandatory for every student to borrow books from the library which is nothing but modality.

2.3.4 Object Models

Q66. What is object model? Explain various types of object models with examples.

Answer :

Object Model

Generation of object models are carried out for the system incorporating the object-oriented programming methodology.

Basically, object models are used to develop interactive systems. This means, these systems are being implemented by using object oriented programming languages like C++ or JAVA.

During requirement analysis, object models developed can represent both system data and its processing.

Object models are natural representation of real world entities called object classes. These entities have their own attributes and operations. The object classes are required in information processing system.

For instance, entities may be student, book, account etc. The attributes of a student entity may be name, roll no, student id etc.

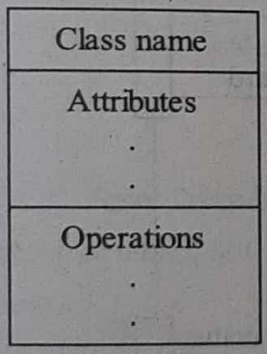
An object class consists of three sections,

(i) Class name

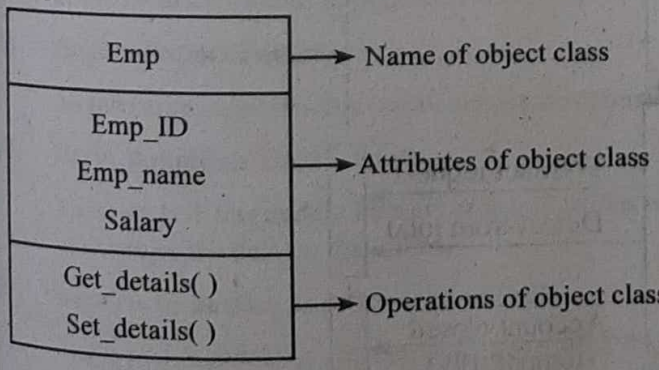
(ii) Attributes of class

(iii) Operations of class.

In UML, it is graphically represented by a rectangle as,



Example



All these things can be represented by using object model. In order to simplify the system design, object model is combined with data flow model.

Different types of object models can be created depending on the activities of object i.e., object aggregation, object interaction etc.

Types of Object Models

There are three different types of object models. They are,

1. Inheritance model
2. Object aggregation
3. Object behaviour model.

1. Inheritance Model

In object modeling, a group of classes are identified and if there exists some common attributes and services then inheritance property is implemented.

Inheritance reflects the property of deriving one class property into other classes. The property of the class being derived is referred as parent class and the classes acquiring these properties are referred to as child classes. Consider a simple diagram depicting the animal kingdom.

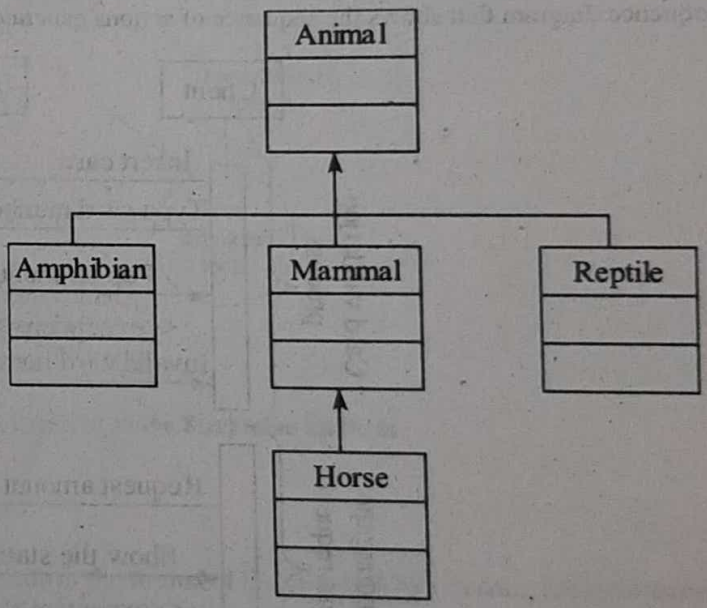


Figure: An Example of Inheritance Hierarchy

In the figure, inheritance is shown by upward arrow.

Here, 'Animal' is a base class through which three child classes are inherited namely, 'amphibian', 'mammal' and 'reptile. Here, further 'mammal' acts as a base class through which one child class is inherited i.e., 'Horse'.

2. Object Aggregation

Sometimes certain situations may arise where, a single object may have multiple objects associated with it. To model such situation object aggregation model is used. Following is an example depicting the modelling of various components of a computer system.

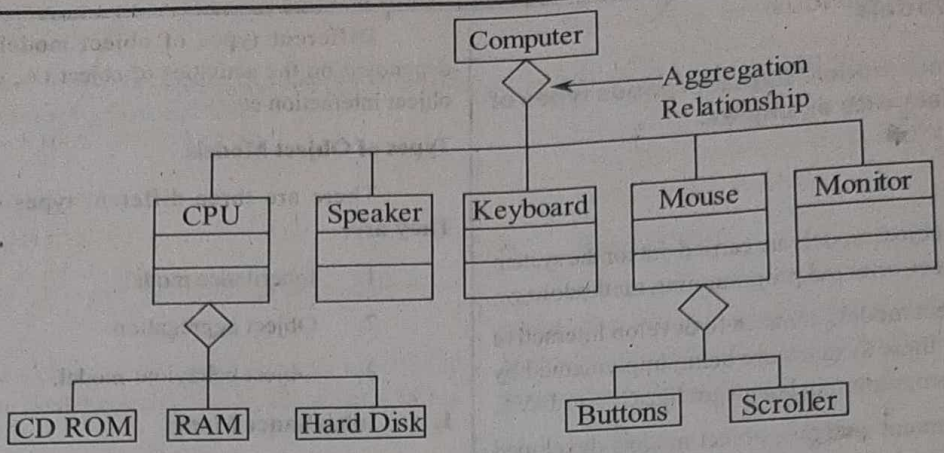


Figure: Aggregation Relationship Existing Among Various Objects

3. Objects Behaviour Model

In object behaviour modelling, the behaviour of objects is represented by the sequence of actions. In the UML, this is modeled using the sequence diagram.

In a sequence diagram, actors and the objects are placed at the top of the diagram and labelled arrows represent the operations of system. The vertical rectangle specifies the time span of the particular operation. The following is an example of sequence diagram that shows the sequence of actions generated during ATM withdrawal transaction.

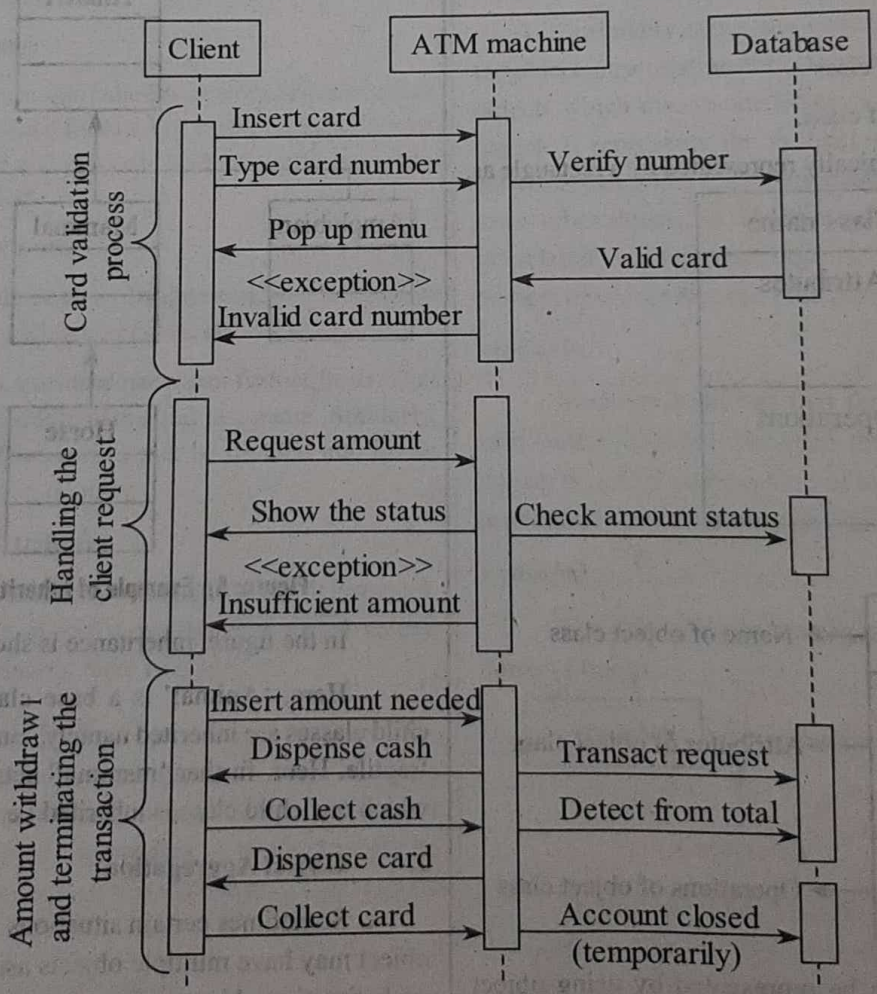


Figure: Sequence Diagram Depicting ATM Withdrawl Transaction

2.3.5 Structured Methods

Q67. Discuss about different structured methods used in software development.

Answer : (Model Paper-III, Q5(b) | Nov./Dec.-16(R13), Q5(b))

Structured Methods

Structured methods are used to generate models in an organized manner. These methods find their applications in requirements elucidation and analysis phase of the software development. They provide a core framework which can be successfully applied in developing detailed and specifically large models. They consist of a sets of processes and rules/regulations adhering to which sophisticated models are developed and they can produce standard documents for the system.

These methods also encompass several CASE tools. These tools can be effectively used in,

- Modifying the existing models
- Developing code
- Generating reports
- Validating the available models upto some extent.

The following figure depicts various components of CASE tools supported by structured methods.

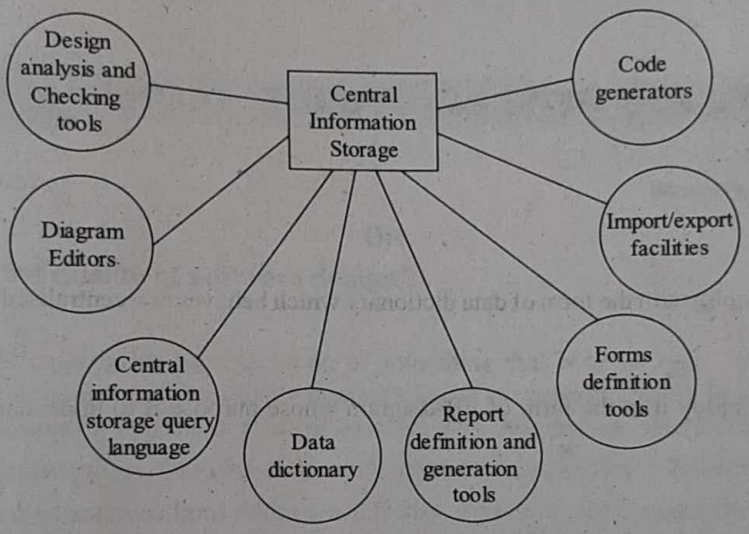


Figure: Components Under the CASE Tools Provided by the Structured Methods

A brief illustration on these CASE tools are as follows,

(i) **Code Generators**

They usually facilitate us by automatically introducing the code or the format of the code just by referring the information available in the central information storage.

(ii) **Import/Export Facilities**

As the name suggests, this component causes transfer of data to/from the central storage and other external data sources.

(iii) **Form Definition Tools**

These tools define certain formats or specifications which remain extremely useful in developing reports or forms and also to structure the data on the screen.

(iv) **Reports Definition and Generation Tools**

These tool facilitate in automatically generating the documents by accepting the data from the central storage.

(v) **Data Dictionary**

This tools stores large volumes of available information related to various entities of the system in an organized manner.

(vi) Central Information Storage Query Language

Using this query language, a given user can easily acquire design and their related information stored in the central information storage.

(vii) Diagram Editors

These editors help in generation of several models (say behavioural models, object-data models etc.) and storing the information on various entities available in these models in the central information storage.

(viii) Design Analysis and Checking Tools

These tools are used for exposing as well as evaluating report and documentation in order to determine errors as well as limitations.

Q68. Briefly explain the models used for structured analysis.

Nov.-15(R13), Q4(b)

Answer :

The various models used in structured analysis are as follows,

1. Data
2. Entity-relation model
3. Data modeling
4. State transition
5. System
6. Modeling language.

1. Data Model

The structural analysis deploy it in the form of data dictionary which behaves as a centralised repository of data and objects.

2. Entity-Relation Model

The structural analysis deploy it in the form of ER-diagram whose purpose is to understand the relationships types and nature.

3. Data Modeling

The structural analysis deploy it as data structure diagram. Its role is to understand and represent the data stored in the form of structure and attribute.

4. Data Flow

The structured analysis deploy it in the form of data flow diagram. It shows the data flow though different functions.

5. State Transition

The structured analysis deploy it in the form of process flow charts. It shows the performance of function under the contact of external stimulus.

6. System

The structured analysis deploy it as system flow diagrams. Its role is to understand the context, position and interface.

7. Modeling Language

The structured analysis deploys it for basic diagramming notations specific for entity, process flow, storage, decision and visio modeling language.

UNIT 3

Design Engineering, Creating an Architectural Design, Conceptual Model of UML

Syllabus

Design Engineering: Design Process and Design Quality, Design Concepts, The Design Model.

Creating an Architectural Design: Software Architecture, Data Design, Architectural Styles and Patterns, Architectural Design, Conceptual Model of UML, Basic Structural Modeling, Class Diagrams, Sequence Diagrams, Collaboration Diagrams, Use Case Diagrams, Component Diagrams.

PART-A SHORT QUESTIONS WITH SOLUTIONS

Q1. Define design process.

(Model Paper-I, Q1(e) | March-17(R13), Q1(e))

OR

How do we assess the quality of software design?

Answer :

Nov.-15(R13), Q1(f)

Design is a meaningful engineering representation of something that is to be built. It can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria for "good" design. In the software engineering context, design focuses on four major areas of concern, data, architecture, interfaces and components.

Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software.

The design specification (documentation) addresses different aspects of the design model. It will be completed as the designer refines his representation of the software. First, the overall scope of the design efforts is described.

Next, the data design is specified, database structure, any external file structures, internal data structures and a cross reference that connects data objects to specific files are all defined.

The architecture design indicates how the program architecture has been derived from the analysis model. In addition, structure charts are used to represent the module hierarchy (if applicable).

Q2. What are the goals of the design process?

Answer :

May/June-19(R16), Q1(e)

The goals of the design process are as follows.

1. **Determination of Different Modules**

The design phase of the software development cycle should determine different modules in the solution. These modules carry a set of functions along with data shared among them. Every module must perform a specific job which is a part of the overall software functionality. Based on this job, names are assigned to the modules.

2. **Determination of Control Relationships**

If two modules carry functions between them, then there exist control relationship between them. The design process must determine all such relationships in the software.

3. **Developing Interfaces between Modules**

Design phase developer interfaces between modules that helps in the determination of data transferred among various modules during function invocation.

4. **Designing and Documenting Modules**

Every module of the system store same data that is needed to be shared for carrying out the functionality of the module. Design phase design and document data structures for storing this data.

5. **Designing and Documenting Algorithms**

In design phase, algorithms that are developed to carry out processing activities are designed and documented. This task is performed by focusing on accuracy of the output, space complexity and time complexity.

Q3. List the principles of software design.

Answer : March-17(R13), Q1(f)

Three things that leads to good software designs are,

1. The designers should design the software in such a way that it should reflect external and internal requirements as stated in the requirement analysis phase.
2. It should be easily understandable and readable for testers, developers at the same time should support the software.
3. It should show the perfect picture of the software, its functional, data and behavioural domains.

Q4. What do you mean by software design quality? Explain.

Answer : Nov./Dec.-16(R13), Q1(f)

The quality of design that is under development can be assessed by conducting formal technical reviews or design walkthroughs . However, McGlaughin suggested three guidelines for the evaluation of a good design,

1. The requirements must be designed well. The design must include all the requirements of the customer. It must implement the explicit requirements of the analysis model.
2. The design must be properly understandable and readable by the testes who tests the software, as well as by the programmers who develops the code for the software.
3. The design must cover all the important points such as data, address, functions etc. A good software is designed for obtaining a good quality.

Q5. Give few characteristics of good design.

Answer : Model Paper-II, Q1(e)

Following are few important guidelines often referred as characteristics of a good design.

1. The notion of design should reflect its meaning.
2. Design should be implemented based on the information obtained from the requirement analysis phase of the software development process.
3. The design should clearly represent the interfaces applicable in the system. This causes ease while providing connections between various modules of the software as well as between the software and its vicinity of implementation.
4. The design should consist of all the independent modules (i.e. the modules capable of functioning independently).
5. The design should exhibit clearly all the modules of the software.
6. Various elements such as data, architecture, interfaces and components, should be distinguished clearly.
7. Using the design, a software engineer should directly build data structure which is suitable for the implementation classes. Also the design should be the outcome of easily recognizable components.
8. A design should describe architecture, should encompass several components and should reflect dynamic behaviour etc.

Q6. Why should we not over modularize?

Answer :

Modularization is the process of dividing the software product into different modules. Before being assembled into an integrated product, each module is tested separately in order to improve the quality of software product. A module is simply a system component that provides one or more services to other modules for achieving the software improvements. But we should not over modularized because as many modules existed in the software system, the complexities can become more. Too many modules can lead to misunderstanding of software architecture.

Therefore, under modularity and over modularity should be avoided.

Q7. What are the quality parameters considered for effective modular design?

Answer : (Model Paper-III, Q1(e) | May-18(R16), Q1(f))

The quality parameters considered for the effective modularity design are,

1. Functional independence
2. Cohesion
3. Coupling.

1. **Functional Independence**

It is an effective property of modularity, which relates to the concept of abstraction and information hiding.

2. **Cohesion**

Cohesion is defined as a measure of the degree to which the elements of a module are functionally related with each other. The main principle of cohesive module is to implement the functionality that is related to a feature of the solution.

3. **Coupling**

Coupling is defined as a measure of the degree that shows the interdependencies among modules. An interface provides relationship between modules, where inputs are received and outputs are sent for the further processing.

Q8. Define functional Independence.

Answer :

Functional independence is an effective property of modularity, which relates to the concept of abstraction and information hiding. Functional independence is achieved by developing modules with the help of a unique function that does not need to interact with other modules. In simple terms, the software design for each module provides a specific sub-function of requirements that has a simple interface to view many parts of the program structure.

Independent modules are easier to maintain and test because the effects caused by design or code modification are limited, error propagation is reduced and reversible modules are possible. Functional independence can be measured using two qualitative criteria i.e., cohesion and coupling.

Q9. Explain the function independence consequences to software design.

Answer : Dec.-11, Set-2, Q7(a)

The consequences of functional independence are given as follows,

(i) **Error Isolation**

Error propagation is reduced because the degree of interaction of an independent module with the other modules is less. Therefore, if any error is found in the existing module it will not directly affect the other modules.

(ii) **Scope of Reuse**

A module can be reused because the interaction of each module with other modules is minimal and each module has its own well-defined and precise functions. Such modules are called cohesive and they can be easily reused.

(iii) **Understandability**

The design complexity can be minimized because different modules have less interaction with other modules and can be understood independently.

Q10. Discuss the statement "abstraction and refinement are complementary concepts".

Answer :

Abstraction and Refinement are Complimentary Concepts

The abstraction and refinement are closely related with one another.

Refinement

It is a process of elaboration. Modern software development is a complicated process especially when software system becomes large and complicated. Therefore, software developers must apply software refinement in order to proceed from higher levels of abstraction to a final executable software system by appending essential details.

Refinement provides even the low-level details as the (information) design proceeds with the progress in refinement every time it allows a designer to elaborate the original software development process and more and more details are added over time.

Abstraction

To increase the efficiency and to decrease the complexity in a software development process we makes use of abstraction.

Abstraction is defined as the process of hiding all the irrelevant data and provides only the essential or important data. It enables the designer to specify procedure and data thereby suppressing low-level details.

Q11. Write about any three types of design classes.

Answer :

The three different types of design classes are given below,

(a) **Process Classes**

These classes represent the low-level graded business abstractions for managing the business.

(b) **Persistent Classes**

These classes are used to store large information (usually a database).

(c) **User Interface Classes**

These classes are used to implement the abstractions for human computer interaction. Usually these classes provide the visual representation of the elements of the metaphor used in HCI.

Q12. Write short notes on architectural patterns.

Answer : Model Paper-I, Q1(f)

Architectural pattern is a bit analogous to architectural style. It transforms the design of an architecture. However, architecture pattern and architecture style differ with each other in number of ways.

- Pattern usually considers only single aspect of the entire architecture.
- It specifies certain rules which are to be obeyed by a given architecture. These rules define how the software will perform a part of its functionality at the infrastructure level.
- It exposes only certain behavioural facts of the architecture.

Architecture patterns can be combined with an architectural style in order to provide a suitable shape to the existing software structure.

Q13. Explain in brief the taxonomy of various architectural styles.

Dec.-19(R16), Q1(e)

Answer :

The taxonomy of architectural styles are,

(a) Data Centered Architecture

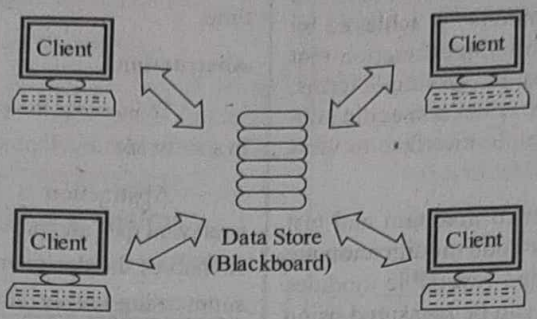


Figure: Data Centered Architecture

Important points related to above architecture are illustrated below.

- In the above architecture, essential data reside at the centre of the architecture.
- All the client software are authorized to access this data.
- These client software can easily manipulate the centered data i.e., they can delete, update, add etc., independently.

(b) Data Flow Architecture

Diagrammatic representation of data flow architecture is shown below.

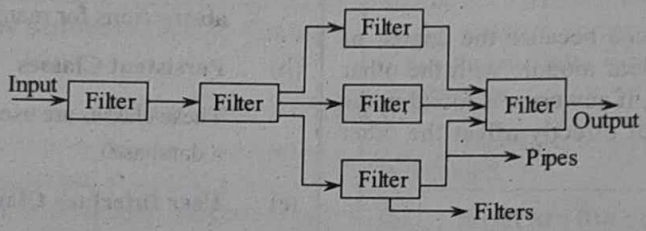


Figure: Data Flow Architecture

This architecture is applied when the given data is to be transformed into certain output by applying series of components (filters), this architecture consists of a set of components called filters, that are connected to each other by points for transmitting data.

(c) The Call and Return Architecture

The call and return architecture is best known to frame a given software such that the resultant architecture can be modified or shrunk depending on the requirements. This architecture has got two sub-architectures referred as main program/subprogram architecture and remote procedure call architecture.

(d) Layered Architecture

There are basically four layers in layered architecture. The rectangular small boxes in each layer are the components associated with that layer. Each of these layers defines a definite set of operations. The deeper layers are nearer to the machine instruction.

Q14. Distinguish between classes and components.

Answer :

	Classes		Components
1.	Each class is associated with operations and attributes.	1.	Components are associated with only operations.
2.	A class is a collection of objects, sharing similar properties and behavior.	2.	A component is a collection of logical elements. These elements can be classes or any other collaborations.
3.	Classes can be used to represent logical abstractions.	3.	These are used to represent physical elements, generally made of bits (say, data file, documents etc.).

Q15. Draw a software architecture for a human resource system.

Answer :

Software Architecture of Human Resource System

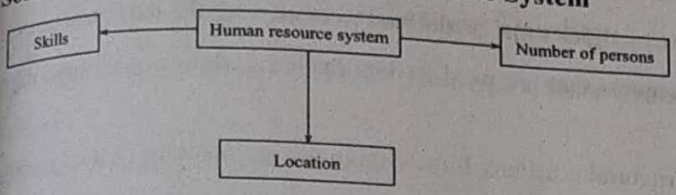


Figure: Software Architecture of Human Resource System

(a) **Skills**

Skills are the qualities that the planner selects for the system development.

(b) **Location**

Location plays a crucial role when the project is big and requires its project team members to be distributed across different locations to perform the software engineering tasks such as consulting with specialists etc.

(c) **Number of Persons**

It determines the number of persons required for the project.

Q16. Name the commonly used architectural styles.

Answer : **Nov./Dec.-18(R16), Q1(f)**

The following are the most commonly used architectural styles.

(i) Data flow architecture

(ii) Object-oriented architecture

(iii) Layered system architecture

(iv) Data-centered architecture

(v) Call-and-return architecture.

(i) **Data Flow Architecture**

This architecture is applied when the given data is to be transformed into certain output by applying series of components (filters), this architecture consists of a set of components called filters, that are connected to each other by points for transmitting data.

(ii) **Object-oriented Architecture**

It is an architectural style which divides the functionalities of an application or system into different independent and reusable objects. In other words, it divides a system functionality into different subsystems using object oriented concept. This style is used during application design stage. It represents the system in the form of different objects that work collaboratively rather than a set of procedures or sub-programs. The objects in this style are loosely coupled and interact with each other through interfaces or messages.

(iii) **Layered System Architecture**

There are basically four layers (i.e., core layer, utility layer, application layer, user interface layer). The rectangular small boxes represented in each layer are nothing but the components associated with that layer. Each of these layers defines a definite set of operations.

(iv) **Data-centered Architecture**

In data centered architecture, essential data reside at the centre of the architecture and all the client softwares are authorized to access this data

(v) **Call-and-Return Architecture**

The call and return architecture is best known to frame a given software such that the resultant architecture can be modified or shrunk depending on the requirements.

Q17. Explain briefly data design elements.

Answer :

Data Design Elements

Data design often leads to a systematic representation of data in most abstract form. Later by applying refining tools the given data is progressed into a descent format which can be easily processed by any of the computer based system. While dealing with data design, one has to remember that the format (rather design) of data (on which we are currently working) exerts significant impact on the architecture of the software which processes it. This can be analyzed by considering the following stages of development.

(a) **Component Level**

At this level, we generally concentrate on the available forms of data like data structures and its corresponding algorithms. This turns out to be necessary, since quality of end product (software) depends primarily on these aspects at component level.

(b) **Application Level**

At this level, our attention shifts onto storage aspects i.e., converting the given data model into a database. This remains important to achieve the commercial objectives which were estimated on the product.

(c) **Business or Commercial Level**

Once the data is stored in the database, now we concentrate largely on improving its storage aspects i.e. acquiring the required data with less effort applied. This can be achieved by data warehousing mechanisms.

Q18. Define software architecture.

Answer : **(Model Paper-II, Q1(f) | May/June-19(R16), Q1(f))**

Software architecture is a structure of system that consists of certain number of components, along with their visible features and relationships among them.

A software engineer uses software architecture in many ways as it provides specifications, through which he sets his goals, provides information related to modifications to be made to the design at the initial software development stages etc. Hence, he can get rid of potential risks easily at the beginning stages of software development.

Q19. List the guidelines for data design.

Answer : (Model Paper-III, Q1(f) | Nov./Dec.-18(R16), Q1(e))

The guidelines for data design are as follows,

1. Data must be organized orderly – It contains the data flow, its relationship and data objects.
2. Data must include the methods and the data structure. All the methods must be defined to work with the data structure.
3. Provides a data dictionary – Data dictionary defines the elements that are used in data design such as constraints, data objects and their relationship.
4. A data design provides low level design after defining the structural attribute. First, design the architecture of the system, which contains data structure before designing the low level design.
5. A data design provides information hiding – Information hiding plays a vital role in designing a quality software product which gives the comparison between physical and logical views.
6. A data design provides library which defines the methods and data structure which are used in designing the system. The library provides the reusability of data structure which decreases the time for defining the data.
7. Data of the system can be specified by using the programming languages along with software design – By using programming languages and designing model the data of the system can be identified.

Q20. Draw the architectural diagram depicting the 'SafeHome' security system.

Answer :

The architectural diagram depicting the 'SafeHome' security system is given below,

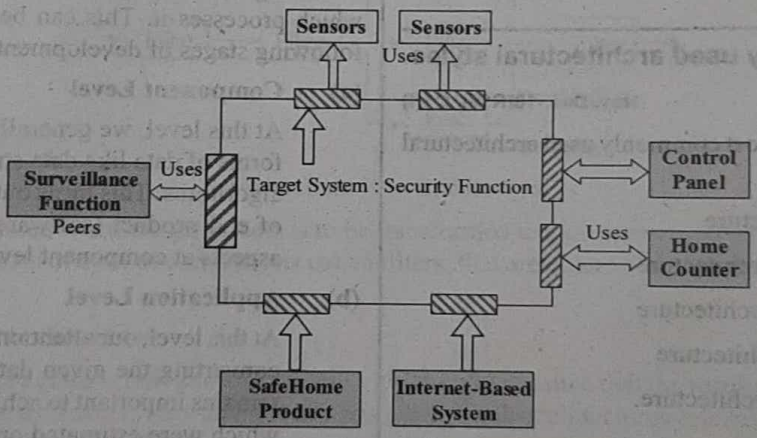


Figure: Architectural Diagram for SafeHome Security Systems

PART-B ESSAY QUESTIONS WITH SOLUTIONS

3.1 DESIGN ENGINEERING

3.1.1 Design Process and Design Quality

Q1. What is software design? Discuss where exactly design comes in software engineering.

Answer :

Software Design

The main objective of software design is to develop a model or representation of a software. It is an iterative process through which requirements are analyzed before the software development. Once the complete set of requirements is analyzed, it is transformed into a blueprint for building the software. Software design also ensures that the software should exhibit the properties of firmness, commodity, and delightness. The designer must also ensure that the developed product must be functional, reliable, easy to understand, modify and maintain. The trend of software design changes continually with the evolution of new methods.

Design in Software Engineering

Designing is one of the important phases of software engineering. Once a complete set of requirements have been analyzed and modeled, designing phase sets the stage for code generation and testing.

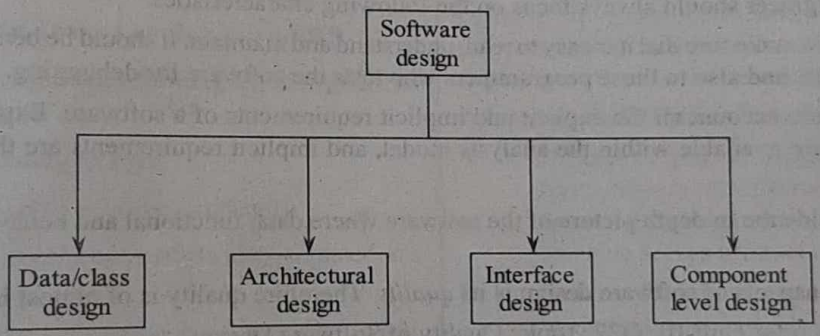
Designing can generally be classified into five types.

(i) Data/class design

(ii) Architectural design

(iii) Interface design

(iv) Component-level design.



(i) **Data/Class Design**

The main objective of the data/class design is to successfully transform analysis class models (that describe what the customer requires, establish basis for the creation of software design and define the set of requirements) into class realizations and the required data structures for implementing the software.

(ii) **Architectural Design**

Architectural design defines the relationship between data flow diagrams, control flow diagrams and processing narratives (also known as *structural elements of the software*). The architectural styles like class diagrams analysis packages and design patterns can be used to achieve the system requirements.

(iii) **Interface Design**

An interface design depicts how the software appear to its users and how will it communicate with the other system. It also shows the flow of information i.e. data and control among the system components.

(iv) **Component-level Design**

The component-level design is used to transform structural elements such as data flow diagrams, control diagrams into a procedural format of software components. Class-based models, flow model and behavioural models provide the ground for component-level design.

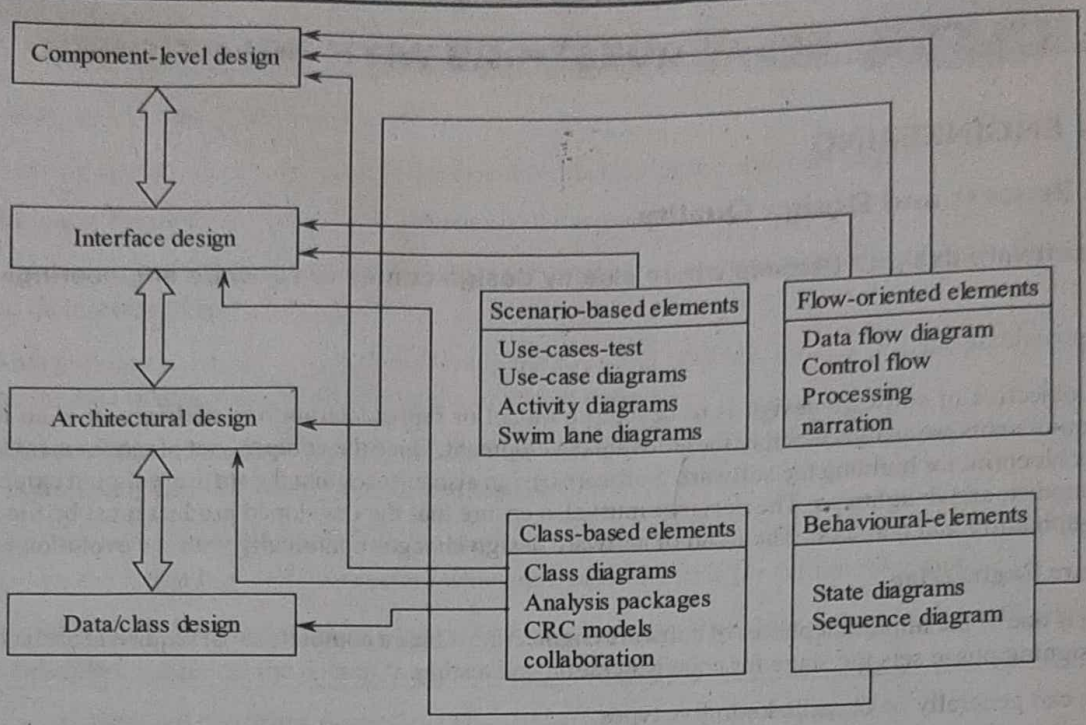


Figure: Transformation of Analysis Model into Design Model

Software design offers representation of software that can be used for measuring the quality. A design is one of the best ways to find out the customer's requirements. It also provides the base for all software engineering and software support activities. A good software design is highly recommended. Without proper designing, there can be a risk of building an unstable system where implementation of new methods or procedures is not possible.

A software design engineer should always focus on the following characteristics.

(i) A software design must make sure that it is easy to read, understand and maintain. It should be beneficial to those programmers who generate the code, and also to those programmers who tests the software for debugging.

(ii) A design must take into account all the explicit and implicit requirements of a software. Explicit requirements are those requirements which are available within the analysis model, and implicit requirements are those which are designed by the customer.

(iii) A design should provide the in depth picture of the software where data, functional and behavioural attributes are easy to implement.

One of the important aspects of software design is its *quality.* Therefore quality is of utmost importance.

For remaining answer refer Unit-III, Q22, Topic: Quality of Software Design.

Q22. How is the quality of a good software design is assessed?

Answer :

Quality of Software Design

Design is a meaningful engineering representation of something that is to be built. It can be traced to a customer's requirements and at the same time assessed for quality against a set of predefined criteria for "good" design. In the software engineering context, design focuses on four major areas of concern, data, architecture, interfaces and components.

Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software.

The design specification (documentation) addresses different aspects of the design model. It will be completed as the designer refines his representation of the software. First, the overall scope of the design efforts is described.

Next, the data design is specified, database structure, any external file structures, internal data structures and a cross reference that connects data objects to specific files are all defined.

The architecture design indicates how the program architecture has been derived from the analysis model. In addition, structure charts are used to represent the module hierarchy (if applicable).

The design of external and internal program interface is represented and a detailed design of the human/machine interface is described. In some cases, a detailed prototype of a GUI may be represented.

The design specification contains a requirement cross reference. The purpose of this cross reference (usually represented as a simple matrix) is to establish that all requirements are satisfied by the software design and to indicate which components are critical to the implementation of specific requirements.

The first state in the development of test documentation is also contained in the design document. Once the program structure and interfaces have been established, guidelines can be defined for testing of individual modules and integration of the entire package. In some cases, a detailed specification of test procedures occurs in parallel with design. In such cases, this section may be deleted from the design specification.

Design constraints, such as physical memory limitations or the necessity for a specialized external interface, may dictate special requirements for assembling or packaging of software. Special considerations caused by the necessity for program overlay, virtual memory management, high-speed processing or other factors may cause modification in design derived from information flow or structure. In addition, this section describes the approach that will be used to transfer software to a customer site.

The final section of the design specification contains supplementary data. Algorithm descriptions, alternative procedures, tabular data, excerpts from other documents, and other relevant information are presented as a special note or as a separate appendix.

Principal Software Design Tasks Leading to an SDD

(i) Requirements-to-design: Transforming requirements into architectural elements.

(ii) Verifying and validating designs.

(iii) Performing risk analysis relative to designs.

(iv) Selecting software interfaces (module inter-connections, user interfaces)

(v) Algorithmic explanation of architectures (how they work, steps performed)

(vi) Detailed design (consider alternative architectures, incremental improvements of selected architecture and complete software design).

Q23. Write about the following,

(a) Design quality

(b) Product-line software.

OR

Elaborate on product-line software.

May/June-12, Set-1, Q6(b)

(Refer Only Topic: Product-line Software)

OR

Elaborate on design equality.

(Refer Only Topic: Design Quality)

Dec.-11, Set-3, Q2(b)

Answer :

(a) Design Quality

The quality of design that is under development can be assessed by conducting formal technical reviews or design walkthroughs . However, McGlaughin suggested three guidelines for the evaluation of a good design,

1. The requirements must be designed well. The design must include all the requirements of the customer. It must implement the explicit requirements of the analysis model.
2. The design must be properly understandable and readable by the testers who test the software, as well as by the programmers who develop the code for the software.
3. The design must cover all the important points such as data, address, functions etc. A good software is designed for obtaining a good quality.

(b) Product-line Software

A group of software intensive systems that share same features is called product line software. The features that are shared by this group of systems must fulfill the needs of a particular market segment or mission and they must have been developed from a similar set of core assets in a well-defined way.

Since it is a grouped entity, it allows an organization to manage and evolve its elements as a single or unified entity. It can be reconfigured which involves manipulation activities such as component addition or deletion, parameter and constraints defining for components and adding of business processes information to it. It can be configured at design-time and deployment time in the development process.

Advantages of Product-line Software

- Different organizations (small and large) use the product line software in different domains since it allows to promote the asset reuse through the software lifecycle and also serves product customization.
- It allows the organizations to improve productivity, time-to-market, cost and reliability.

Q24. Explain various software quality standards and discuss how to assure them.

Answer : **Nov.-15(R13), Q10(b)**

The attributes such as functionality, usability, reliability, performance and supportability are often referred to as software quality attributes. These were initially introduced by Hewlett-Packard which remain as target values to be satisfied by any of the software designs.

(i) Functionality

In order to determine functionality, usually the efficiency of the program and its maturity in performing various functions are assessed and finally its security aspects are analyzed deeply.

(ii) Usability

Usability of a given system can be determined by considering its consistency, design, documentation and various other human factors.

(iii) Reliability

Its given program depends on following factors.

- The mean time to failure.
- The potentiality to escape from errors and the ability to recover from failures.
- Frequency and severity of failure of a program.
- The predictability of the program etc.

(iv) Performance

It is computed based on the following attributes.

- Efficiency
- Throughput
- Response time
- Amount of resource utilization
- Processing speed etc.

(v) Supportability or Maintainability

Supportability or maintainability of a given program can be computed based on following attributes.

- Configurability
- Compatibility
- Testability
- Extensibility
- Serviceability
- Adaptability
- The simplicity in separating the defects
- The simplicity of installation of a program etc.

Q25. "Data modeling can be viewed as a subset of OOA." Comment on this statement and justify your comments.

Answer : Nov.-15(R13), Q5(b)

Object Oriented Analysis (OOA) is basically an extension of an older analysis technique called data modelling. Data modelling technique has been used for data intensive applications and mainly focuses on data represented as "data network" on a system. In applications where data and relationship governing data are complex, data modelling technique is very useful. Although data modelling and some of its graphical notations are similar to that of the Object Oriented Analysis, but their approaches are different from each other. For example, data modelling and OOA use the term "object" but data modelling definition is limited. Both define relationships between "Objects" but data modelling does not describes how relationship is established between objects.

If the entity-relationship notation is considered then it can be said that "data modelling can be viewed as a subset of Object Oriented Analysis". Using E-R diagram data modelling mainly focuses on defining data objects in which 2 objects relate each other (i.e., objects relationship). So, using E-R diagrams data modelling can be said as a subset of object oriented analysis.

Q26. Explain the following five component characteristics:

(a) Standardized
(b) Independent
(c) Composable
(d) Deployable
(e) Documented.

Answer : (Model Paper-I, Q6 | Nov./Dec.-17(R15), Q6)

Characteristics of Software Components

A software component should be independent, composable, deployable, standardized and documented.

(a) Standardized

A component used in Component Based Software Engineering (CBSE) must meet the standard of component model. This model defines metadata, interface, deployment and documentation of components.

(b) Independent

A software component should be independent of other software component. It should be generated independently from other components. It must also be deployable without depending on other components. However, when one component needs services of other components, this requirement must be set out in the interface specification.

(c) Composable

In a software component, all interactions must take place using interfaces that are defined as public. A component should also offer access to information such as its characteristics and methods to other components.

(d) Deployable

A component must be capable of getting operated as a single stand-alone entity and it should be self-centered. Further, a component does not need compilation before its deployment as it is in binary form.

(e) Documented

A component should be documented and this documentation should include the syntax and semantics of interface. Documentation helps potential users to decide if the documented component is useful for them or not.

3.1.2 Design Concepts

Q27. State and explain various software design concepts.

Answer :

Software Design Concepts

Different types of software design concepts include,

(i) Abstraction

(ii) Architecture

(iii) Pattern

(iv) Modularity

(v) Information hiding

(vi) Functional independence

(vii) Refinement

(viii) Refactoring

(xi) Separation of concerns

(x) Aspects.

(i) Abstraction

Abstraction is a mean of describing a program function, with an appropriate level of details. At the highest level of abstraction a solution is stated in the language of the problem environment (requirements analysis). At the lowest level of abstraction, implementation-oriented terminology is used (programming language). An abstraction can be compared to a model which incorporates details only to the extent needed to fulfill its purpose.

Abstraction is a very powerful concept which is used in all engineering disciplines. It is a tool that permits a designer to consider a component at an abstract level without worrying about the details of its implementation. Any component or system provides some services to its environment. An abstraction of a component describes the external behavior of that component without bothering with the internal details that caused the behavior.

Abstraction is used for existing components as well as the new components that are being designed. Abstraction of existing components plays an important role in the maintenance phase. To modify a system, the first step is, understanding what the system does and how. The process of comprehending an existing system involves identifying the abstractions of subsystems and components from the details of their implementations. Using these abstractions, the behavior of the entire system can be analyzed. This also helps in determining what are the effects of modifying a component on the system.

During requirements definition and design phases, abstraction permits separation of conceptual aspects of a system from (yet to be specified) implementation details. For example, the FIFO property of a queue or the LIFO property of a stack can be specified without concern for the representation scheme to be used in implementing the stack or queue. Similarly, the functional characteristics of the routines can be specified that manipulate data structures (Example, NEW, PUSH, POP, TOP, EMPTY) without concern for the algorithmic details of the routines.

Three widely used abstraction mechanisms in software design are functional abstraction, data abstraction, and control abstraction.

(a) Functional Abstraction

In functional abstraction, a module is specified by the function it performs. For example, a module to compute the log of a value can be abstractly represented by the function log. Similarly, a module to sort an input array can be represented by the specification of sorting. Functional abstraction is the basis of partitioning function-oriented approaches. That is, when the problem is being partitioned, the overall transformation function is partitioned into smaller functions, that comprise the system function. The decomposition of the system is in terms of functional modules.

(b) Data Abstraction

Data abstraction involves specifying a data type or a data object by specifying legal operations on objects. Representation and manipulation details are suppressed. Thus, the type "stack" can be specified abstractly as a LIFO mechanism in which the routines NEW, PUSH, POP, TOP and EMPTY are included. In data abstraction, data is not treated simply as objects, but is treated as objects with some predefined operations on them. The operations defined on a data object are the only operations that can be performed on those objects. From outside an object, the internal details of the object are hidden and only the operations on the object are visible. Data abstraction forms the basis for object-oriented design.

(c) Control Abstraction

Control abstraction is the third commonly used abstraction mechanism in software design. It is used to state a desired effect without stating the exact mechanism of control. IF statements and WHILE statements in modern programming languages are abstractions of machine code implementations that involve conditional jump instructions. A statement of the form "for all I in S sort files I" leaves unspecified sorting technique, the nature of S, the nature of the files, and how "for all I in S" is to be handled.

(ii) Architecture

The architecture of the procedural and data elements of a design represents a software solution for the real-world problem defined by the requirements analysis.

Software architecture describes two important characteristics of a computer program.

1. The hierarchal structure of procedural components
2. The structure of data.

Software architecture is derived through a partitioning process that relates elements of a software solution to parts of a real-world problem implicitly defined during requirements analysis. The evolution of software and data structure begins with a problem definition. The solution occurs when each part of the problem is solved by one or more software elements.

(iii) Pattern

A design pattern describes a design structure that provides a reliable solution to a recurring problem. Thus, it can be applied to another problem of the same type.

Pattern forms an instance of the original design which includes certain specifications through which its implementation and various other details can be judged. Hence, by developing patterns, a designer can determine,

1. Whether the specifications included in these patterns are satisfying the requirement.
2. Whether the given patterns is reusable in other applications.
3. Whether it can form as a sample for developing other applications etc.

(iv) Modularity

The main concept behind partitioning can be visualized, if a system is partitioned into modules so that the modules are solved, modified and compiled separately (then changes in a module will not require recompiling of the whole system). A system is considered modular if it consists of discrete components so that each component can be implemented separately, and a change to one component has minimal impact on other components.

Modularity is a desirable property of a system. It helps in system debugging, isolating the system problems, changing a part of the system and in system building. A modular system can be easily built by "putting its modules together."

A software system cannot be made modular by simply chopping it into a set of modules. For modularity, each module needs to support a well-defined abstraction and have a clear interface, through which it can interact with other modules. Modularity is where abstraction and partitioning come together. For easily understandable and maintainable system, modularity is clearly the basic objective; partitioning and abstraction can be viewed as concepts that help to achieve modularity.

(v) Information Hiding

Information hiding is a fundamental design concept for software. The principle of information hiding was formulated by Parnas. When a software system is designed using the information hiding approach, each module in the system hides the internal details of its processing activities and modules communicate only through well-defined interfaces.

According to Parnas, design should begin with a list of difficult design decisions and design decisions that are likely to change. Each module is designed to hide such a decision from the other modules. Because these design decisions transcend execution time, design modules may not correspond to processing steps in the implementation of the system. In addition to hiding of difficult and changeable design decisions, other candidates for information hiding include:

(a) A data structure, its internal linkage, and the implementation details of the procedures that manipulate it (this is the principle of data abstraction).

(b) The format of control blocks such as those for queues in an operating system (a "control-block" module).

(c) Character codes, ordering of character sets, and other implementation details.

(d) Shifting, masking and other machine dependent details.

Information hiding can be used as the principal design technique for architectural design of a system, or as a modularization criterion in conjunction with other design techniques.

(vi) Functional Independence

It is an effective property of modularity, which relates to the concept of abstraction and information hiding. Functional independence is achieved by developing modules with the help of a unique function that does not need to interact with other modules. In simple terms, the software design for each module provides a specific sub-function of requirements that has a simple interface to view many parts of the program structure.

Independent modules are easier to maintain and test because the effects caused by design or code modification are limited, error propagation is reduced and reversible modules are possible. Functional independence can be measured using two qualitative criteria i.e., cohesion and coupling.

(vii) Refinement

Step-Wise refinement is a top-down technique for decomposing a system from high-level specifications into more elementary levels. It is also known as "Step-Wise Program Development" and "Successive Refinement". Step-wise refinement involves the following activities.

(a) Decomposing design decisions to elementary levels.

(b) Isolating design aspects that are not truly interdependent.

(c) Postponing decisions relative to representation details as long as possible.

(d) Carefully demonstrating that each successive step in the refinement process is a faithful expansion of previous steps.

The step-wise refinement technique breaks the logic of design problem into a series of steps, so that the development can be done gradually. The process starts by converting the specifications of the module into an abstract description of an algorithm containing a few abstract statements. In each step, one or several statements in the algorithm (developed so far) are decomposed into more detailed instructions. The successive refinement terminates when all instructions are sufficiently precise and can easily be converted into programming language statements. During refinement, both data and instructions have to be refined. A guideline for refinement is that, in each step, the amount of decomposition should be such, that it (refinement process) can be easily handled and represent one or two design decisions.

(viii) Refactoring

Refactoring is a process of improving the existing software without altering its internal mechanisms. Hence, once a given software is said to be refactored, it means that all the loopholes existing in it such as inappropriate data structures or inefficient algorithms or certain redundant data etc., are corrected.

(ix) Separation of Concerns

A concern is the behaviour of a requirement model specification. When these concerns are divided into smaller manageable parts, a complex problem consumes less effort as well as time to solve. It also becomes easier to solve.

(x) Aspects

During the requirements analysis phase, many concerns (requirements) related to data structure, patterns, requirements and QoS issues uncover. A requirements model must be organized in a manner that separates an individual concern in order to be handled independently of other concerns. However, few concerns span over entire system and such systems cannot be compartmentalized.

Q28. Define modularity. Discuss the ways of achieving effective modularity. Write advantages and disadvantages of modularization.

Answer :

Modularity

For answer refer Unit-III, Q27, Topic: Modularity.

Effective Modularity

The different ways of achieving the effective modularity are,

1. Functional independence
2. Cohesion
3. Coupling.

1. Functional Independence

For answer refer Unit-III, Q27, Topic: Functional Independence.

2. Cohesion

Cohesion is defined as a measure of the degree to which the elements of a module are functionally related with each other. The main principle of cohesive module is to implement the functionality that is related to a feature of the solution. It measures how closely various steps in a module perform with least or no interaction with other parts of the system, subsystem or modules. Cohesion of a module gives a clear idea to the designer that whether different elements of a module belong to the same module or not.

3. Coupling

Coupling is defined as a measure of the degree that shows the interdependencies among modules. An interface provides relationship between modules, where inputs are received and outputs are sent for the further processing. When two modules with high coupling are strongly interconnected and dependent on each other then it is known as tight coupling. An interface developed for simple data transfer from one module to another is known as data coupling. The degree of coupling is lowest for data communication, higher for control communication and highest for modules that modify other modules.

Advantages of Modularization

The advantages of modularization are as follows,

(i) It makes an easiest way of understanding the software structure by separating the complete software architecture into different modules.

(ii) It provides fast delivery and increased quality of software because each module is tested before they are assembled into an integrated software product.

(iii) It reduces the development cost by shortening the development time. It also reduces risks by testing and debugging each module efficiently.

(iv) It removes redundancy and inconsistency by combining the similar modules of the system.

(v) It helps in architectural transformation where a centralized system proposed for a single computer is modified to run on a distributed platform.

(vi) Modifications to an existing software system can be made more easily understandable.

Disadvantages of Modularization

The disadvantages of modularization are as follows,

(a) Modularization design can be very complex because well defined methods are needed. Moreover, too many modules can lead to misunderstanding of a system structure. Each module takes its own time for execution.

(b) The designers have to adjust the modules frequently that leads to inconsistency and complexity.

(c) Some modifications to an existing software system from the same platform may not satisfy the needs of the customers.

Q29. Define refactoring. Explain it's intent. Also explain the advantages and disadvantages of it.

OR

Discuss about refactoring in detail.

Answer : Dec.-11, Set-2, Q7(b)

Refactoring

Refactoring is a disciplined technique for restructuring an existing body of code by altering its internal structure without changing its external behaviour or function. It is simply a process of changing the structure of a program without changing its functionality. Refactoring is a powerful technique, but it needs to be performed carefully. So, it is better to refactor the software system into small packages rather than a single attempt. At each refactoring phase, the code needs to be properly tested so that existing functionalities are not disturbed.

The intent of refactoring is to provide the better design of a software system with more understandable code. It improves the quality of a software system and allows developers to repair code that is hard to implement and maintain. It reorganizes the design of a software system without removing its existing source code. The techniques of refactoring are used to simplify the system design by removing unused code and adding flexibility for improvements.

Advantages

The advantages of refactoring are as follows,

1. It makes software system easier to understand.
2. It makes logic as simple as possible.
3. It makes the process of adding new functionality or extending the features of a software system much easier.
4. It helps to find potential errors.
5. It helps to improve performance.

Disadvantages

The disadvantages of refactoring are as follows,

1. It requires a new language or file format.
2. Changes are limited to refactorings.
3. A special tool must be used by developers to record refactorings.

Q30. Define design class. Describe its purpose. Explain different types of it.

Answer :

Design Classes

Design classes are a set of classes that perform the following activities,

(i) Refine Analysis Classes – In this activity, the analysis classes are refined. This refining process generates design details using which it is possible to implement the classes.

(ii) Create a New Set of Design Classes – In this activity, a new set of design classes are created. These classes support the business solution by implementing the required software infrastructure.

Usually the system classes, process classes, persistent classes, user interface classes and business domain classes are referred as design classes. These classes are created by designer.

Types of Design Classes

(i) System Classes

These are the classes which help in the implementation of software management and control functions. Hence, they enable the system to function and communicate with various items of its vicinity as well as with the items which do not correspond to its functional area.

(ii) Business Domain Classes

After completing the analysis phase, certain classes will remain. These classes are usually present in their abstract form. When they are refined further, the business domain classes are obtained. The attributes and methods associated with these classes favours the business domain of the system.

(iii) Process Classes

These classes represent the low-level graded business abstractions for managing the business.

(iv) Persistent Classes

These classes are used to store large information (usually a database).

(v) User Interface Classes

These classes are used to implement the abstractions for human computer interaction. Usually, these classes provide the visual representation of the elements of the metaphor used in HCI.

Q31. What are the characteristics of well-formed design class?

Answer :

There are four characteristics of a well-formed design class.

1. Complete and Sufficient

When the word complete refers to the design class, it means the design class should include all the entities (attributes and methods) essential to describe a given design class.

Sufficiency is a relative term which measures exact number of methods (neither more nor less) in a given design. Hence, a well-formed design class usually possess both the properties i.e. complete and sufficient.

2. High Degree of Cohesion

This feature reflects that whenever a given design class is implemented to represent a specific purpose, then the attributes and methods associated with that class should also reflect the same purpose. As long as this condition is satisfied, these classes are said to be maintaining the property of high degree of cohesion.

3. Low Degree of Coupling

It is often a fact that in a given model, the design classes should have collaboration. But this collaboration should not be extended to extreme values. This is because, a model of design classes possessing high degree of collaboration is difficult to test, maintain etc. Also in this case, the "law of diameter" need to be considered. It suggests that, when there are multiple subsystems, then various methods belonging to single subsystem should communicate by transmitting messages to the methods in neighbouring classes. At the same time, it restricts the methods of one class to communicate with the methods of another classes where the two classes belonging to different subsystems.

4. Primitiveness

It suggests that any method belonging to a given design class should focus only on a single purpose. Hence, once a method implements a given purpose, then there should be no other method in the design class that implements the same purpose.

Q32. Do you design software when you "write" a program? What makes software design different from coding?

Answer :

Designing software before writing the actual program often proves to be beneficial because it simplifies the programming tasks to a greater extent. Further, it eliminates the possibility of developing a wrong program and also ensures that the program works as per the programmer's expectation. Thus the programmer saves himself from writing the code that does not work. Following are the four aspects that must be considered while designing the program.

1. What is the motive behind writing the program?
2. Who will be the user of the program?
3. What are the prerequisites for running the program?
4. Can the program be written by a single programmer?

Designing of a program can be done using a flowchart or a simple sketch that describes the sequence of operations that are to be performed. Some people, however, develops the software directly by writing the code (i.e., without using a design). This practice may cause a severe damage even when there is a minor error in the code. A program can be designed simply by deciding what that computer does first, what it does next and so on. Coding on the other hand, is the translation of the program into the computer code. In simple terms, software designing is the process wherein an optimal solution to the problem is planned. It usually happens once the motive and specifications of the software are determined. Designing can be done either by the software developers (for simple programs) or by the designers (for complex programs).

A source code or the software code refers to the collection of programming language statements. These statements instruct the computer to perform specific operations or actions. The software coding takes place once the designing of the software is completed.

3.1.3 The Design Model

Q33. How to translate the analysis model into the design model? Explain with an example scenario.

(Model Paper-II, Q6(a) | May/June-19(R16), Q6)

Answer :

A design model can be represented in two different dimensions i.e. it represents process dimension along horizontal x-axis and abstraction dimension along vertical y-axis as shown in figure below.

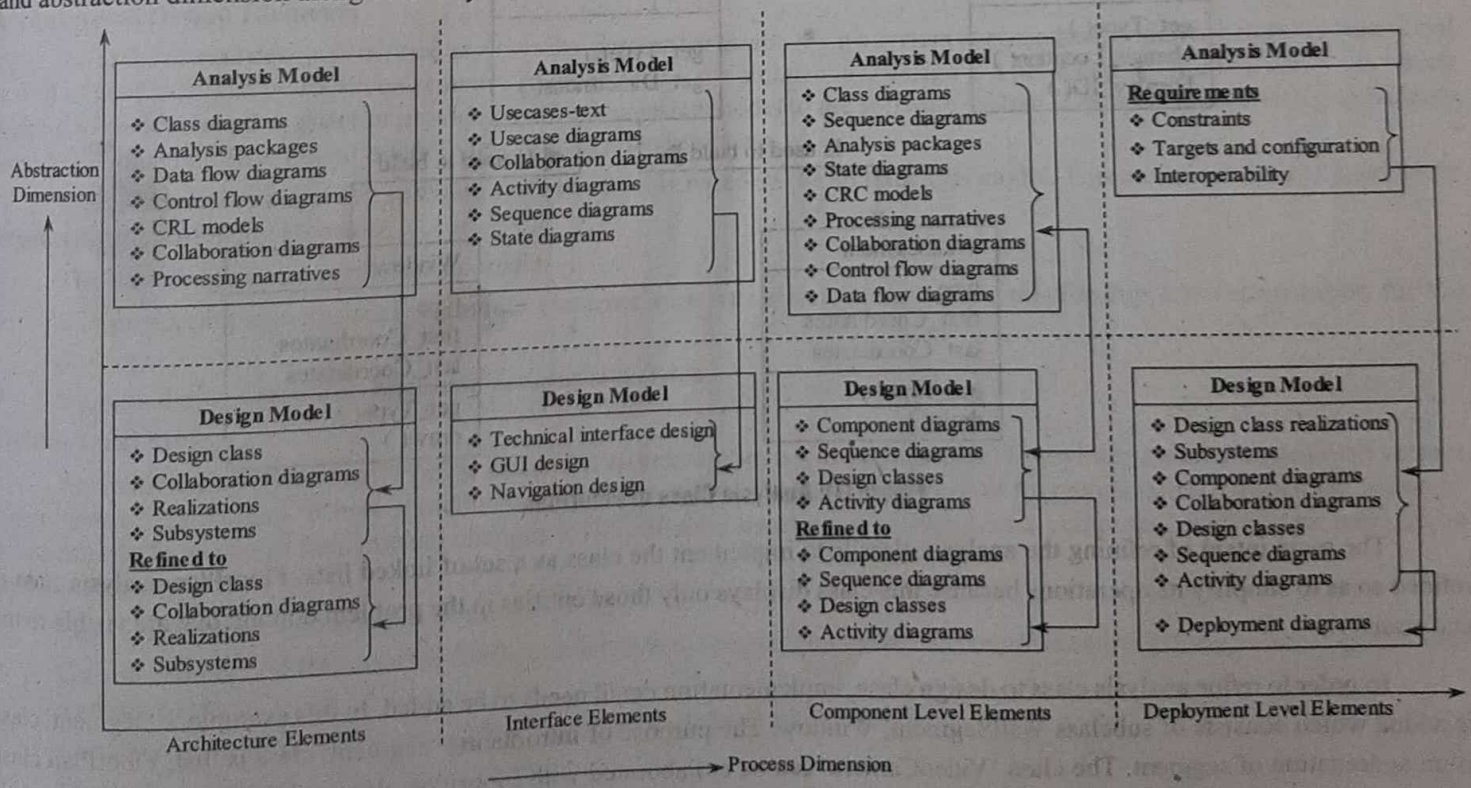


Figure: Design Model

The process dimension along horizontal direction depicts the stages of design development (i.e., architectural elements, interface elements, component-level elements and deployment-level elements) which is usually observed during software development process. The vertical view refers to abstraction dimension. Here, a short details on the transformation of elements from analysis model to design model are shown. Apart from this, the elements which undergo refinement with in the design model are also shown. Usually the refinement is performed in a sequential fashion. The horizontal dashed line appearing at the mid of the diagram is used to separate the analysis model from design model. Usually in both of these diagrams the UML diagrams are present. The UML diagrams appearing in the design models are completely refined when compared to the diagrams in analysis model. It emphasizes the architectural structure and style, components that reside within the architecture and interfaces.

Finally, while dealing with the process dimension, one has to remember that, the architectural design, interface design and component level designs are developed parallely. The deployment-level design is obtained at the end i.e., deployment-level design is considered when the entire design process is completed.

Example

Consider an analysis class of "FloorPlan".

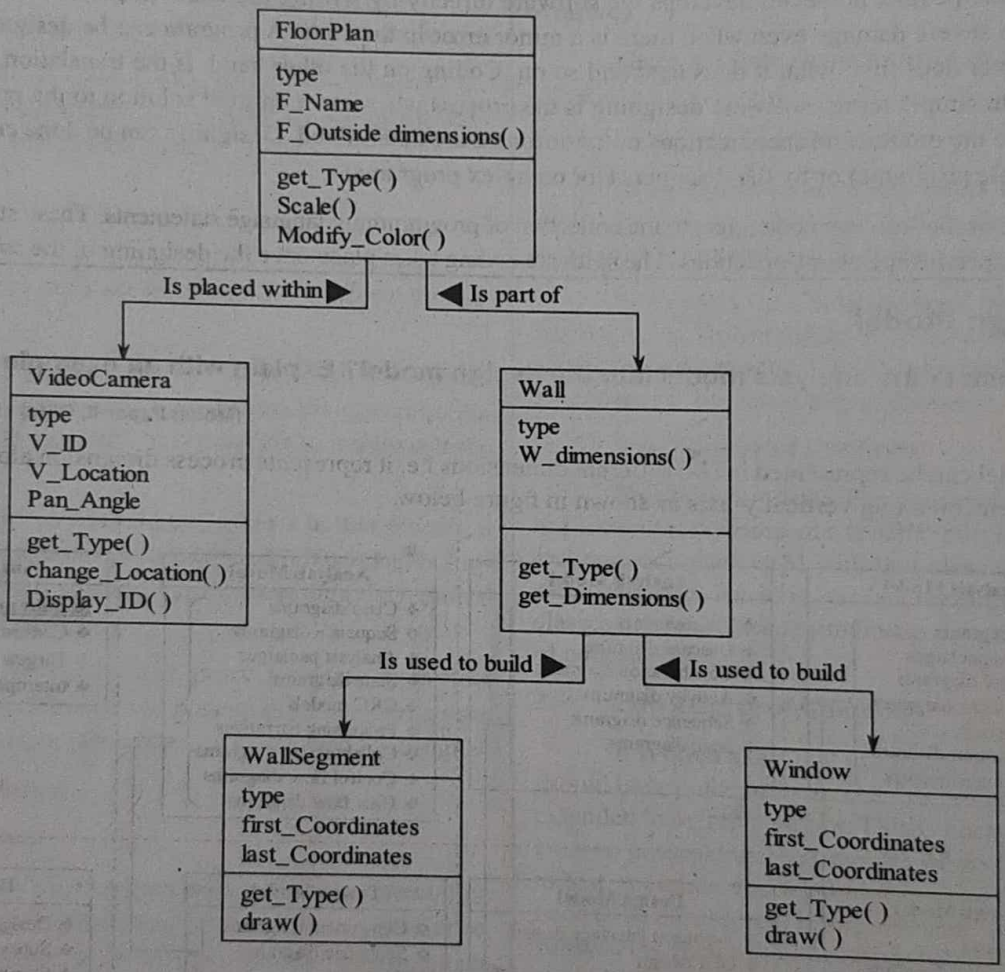


Figure (1): Analysis Class of FloorPlan

The main intent of refining the analysis class is to implement the class as a set of linked lists. FloorPlan analysis class is refined so as to simplify its operations because this class displays only those entities in the problem domain that are visible to the end-user.

In order to refine analysis class to design class, implementation detail needs to be added. In this example, a 'segment' class is added which consists of subclass WallSegment, Window. The purpose of introducing 'segment' class is that, FloorPlan class is an aggregation of segment. The class 'VideoCamera' can be collaborated with FloorPlan class and there can be one-to-many relationship between these classes. The design class for the FloorPlan is shown in figure (2).

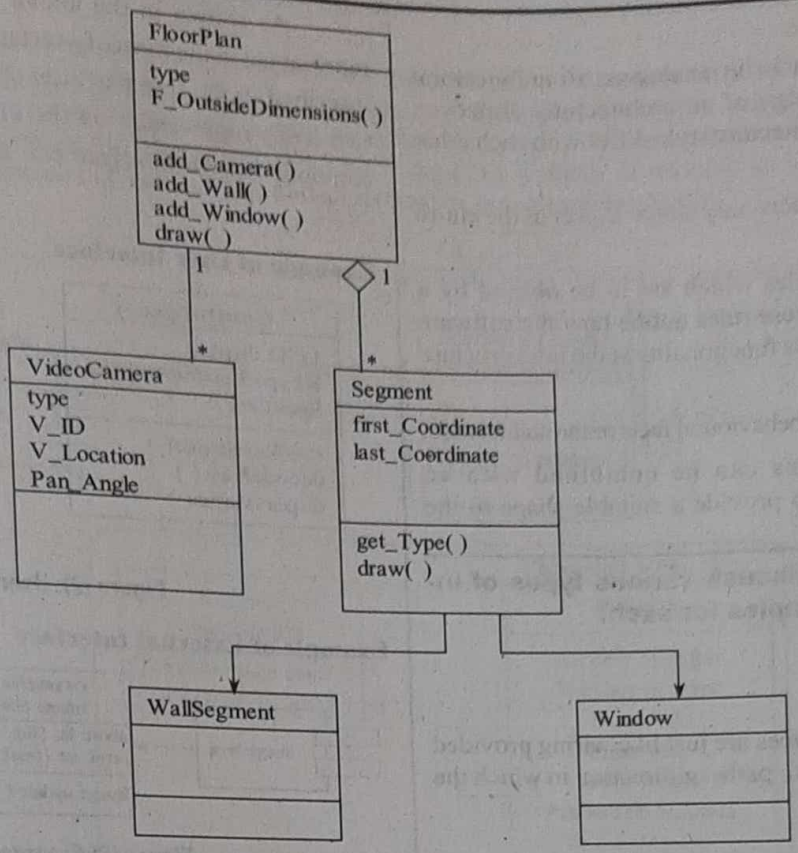


Figure (2): Design Class for FloorPlan

The advantage of refining the analysis class into design class is that it is very easy for the designer to add or delete new items to and from the list without any difficulty.

Q34. Elaborate on architectural design elements.

Answer :

Architectural Design Elements

An architectural design is defined as a creative process for establishing a system organization, which is helpful in functional as well as nonfunctional system requirements. In other words, the architectural design is a photocopy of the end software, which is going to be developed in order to provide an effective design model to the software system. The architectural design elements provide a simple and an efficient overall view of the software system.

The architecture design of a software system is mainly based on the architectural model. This model can be designed while considering the following elements,

- The knowledge about application domain.
- By considering specific analysis model elements such as analysis classes, their relationship and collaboration for the problem.
- By considering various architectural patterns and styles.

Architectural Style

An architectural style transforms the designs of an entire system with an intention of providing suitable structure to various components of that system. It may sometimes happen that the existing architecture is to be reengineered. In such situations an architecture style results in fundamental changes to the software structure. A software for a computer based system includes an architecture style.

Each architecture style consists of the following,

- Certain number of connectors facilitating coordination, cooperation as well as communication among various components forming a system.
- Certain number of components capable of providing definite functionality.
- Semantic models
- Constraints which refer to integration of a system etc.

Architectural Pattern

Architectural pattern is bit analogous to architectural style. It transforms the design of an architecture. However, architecture pattern and architecture style differ with each other in number of ways.

- Pattern usually considers only single aspect of the entire architecture.
- It specifies certain rules which are to be obeyed by a given architecture. These rules define how the software will perform a part of its functionality at the infrastructure level.
- It exposes only certain behavioural facts of the architecture.

Architecture patterns can be combined with an architectural style in order to provide a suitable shape to the existing software structure.

Q35. Define interface. Discuss various types of interfaces. Give examples for each.

Answer :

Interface

In general sense interfaces are just like wiring provided in a given circuit i.e., it refers to paths or direction in which the given information proceeds.

Interface Design Elements

Basically there are three types of interface design elements,

1. The user interface
2. External interface
3. Internal interface.

The user interface accommodates following elements.

1. Technical (e.g. reusable components, UI patterns).
2. Aesthetic (e.g. layout, color, graphics).
3. Ergonomic (e.g. metaphors, UI navigation, information layout and placement).

The external interface defines flow of information between the components of two independent systems. The information related to these interfaces is collected during requirement analysis phase. It is often recommended to check these interfaces before the initiation of interface design. As the information in this case usually flows into other system, it should include security as well as certain error checking measures.

Internal interface defines flow of information between various components of a single system.

Representation of interfaces is same as that used in UML. Hence, consider the following diagram.

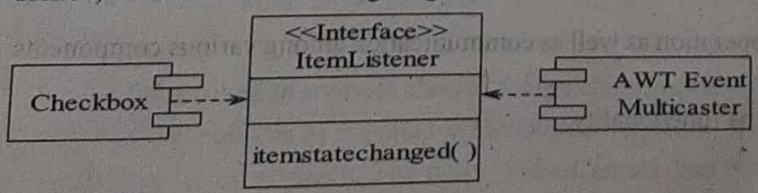


Figure (1): Representation of Interface in terms of UML Notation

As shown in the above figure, an interface is usually represented in the form of a rectangle with a keyword "interface" inscribed in the first partition of the icon. Apart from this, the two icons represented on the either sides of the interface are components. An interface can also be represented as a small circle.

Example of User Interface

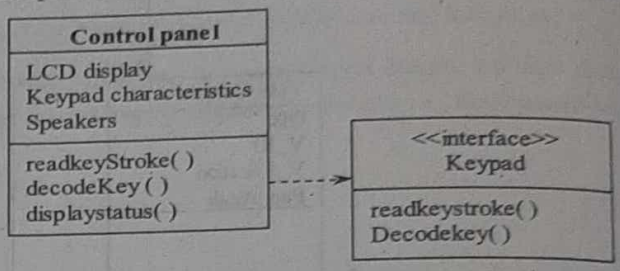


Figure (2): User Interface

Example of External Interface

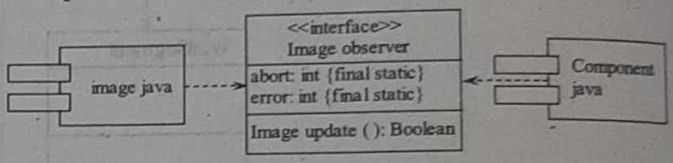


Figure (3):External Interface

Example of Internal Interface

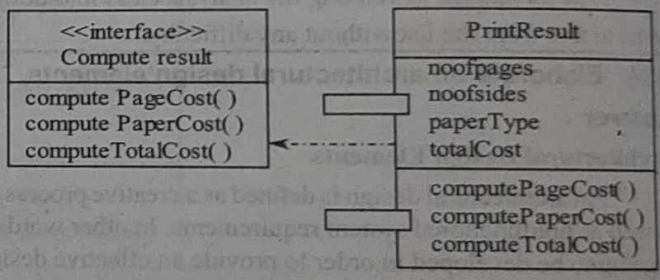


Figure (4): Internal Interface

Q36. Explain the component from traditional view with an example.

OR

How a component is designed based on function? Explain.

(Refer Only Topic: Conventional View)

Answer : May-18(R15), Q7(a)

Views in Defining Component

There are three important views used in defining the components,

1. Object-oriented view
2. Conventional view
3. Process-related view.

Object-oriented View

1. In an object-oriented view, a component can be defined as a collection of collaborating classes. Each of these classes are fully elaborated so that they include all attributes, operations that are used for implementing the component. It is necessary for the designer to define all the relevant interfaces during the design elaboration phase. These interfaces are defined so as to enable communication and collaboration among the design classes. This is done, by initially developing an analysis model and then elaborating the model such that it include all the analysis classes as well as infrastructure classes.

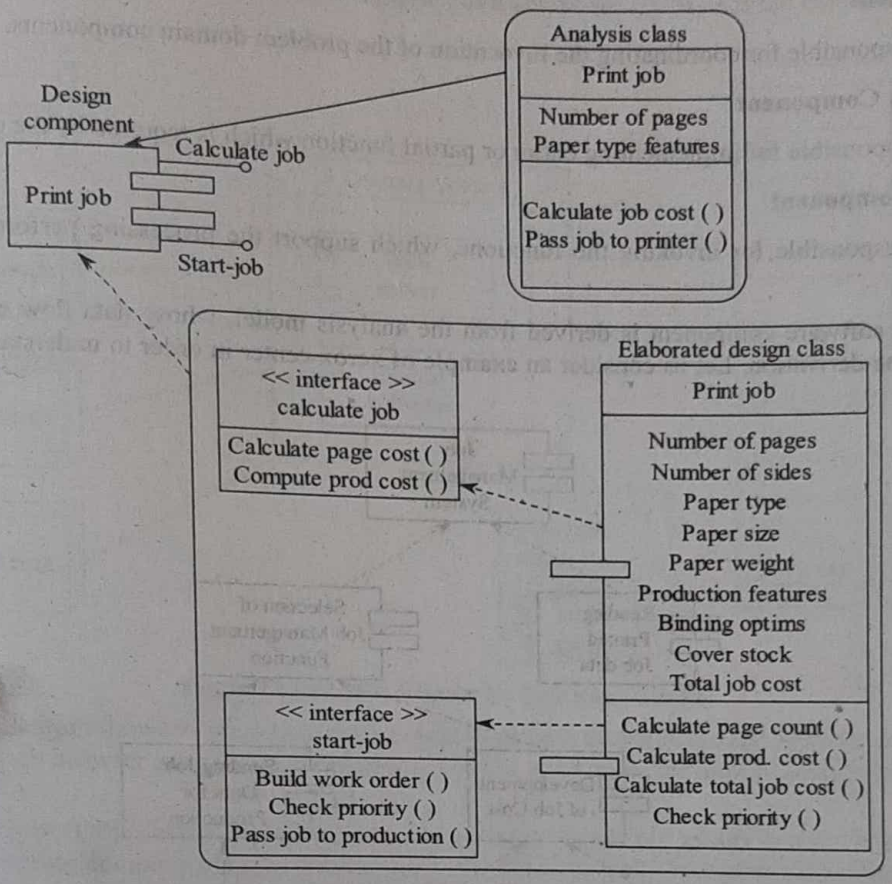


Figure: Design Component Elaboration

Let us consider an example of developing a "print job" software whose objective is to,

(i) Collect information about the customer's requirement

(ii) Determine the cost of print job

(iii) Transfer the job details to an automated production facility.

The designer passes the collected requirements to requirement engineering, where an analysis class called "print job" is generated. During this phase, all the necessary attributes and operations associated with this class are defined. The designer then creates an architectural design where the class "print job" is considered as a component. The following two interfaces are defined for this component, which are represented using "lollipop" notation.

(i) **Calculate-job**

It provides the job costing information.

(ii) **Start-job**

It passes the job to the automated production facility.

Now, the designer initiate the component level design, in which the component 'print job' is elaborated so as to include all the necessary information required for its implementation. The analysis class 'print job' is elaborated with all the attributes and operations in order to implement the class as a component. Then the design class "print job" is elaborated such that it includes all the detailed information about every attribute and operation.

After the elaboration of one component in the architecture, design all the remaining components are elaborated sequentially. Elaboration of every component, the attribute, operation and interfaces are individually elaborated. The elaboration of attribute is performed by defining the necessary data structure and operations are elaborated by defining the algorithmic detail which is required for implementing the processing logic.

2. Conventional View

In conventional view, the component can be defined as a functional element of a program that consists of (i) processing logic, (ii) internal data structure which are necessary for implementing the processing logic, (iii) an interface that enables the invocation of components (iv) data that is to be passed between the components. Generally, component is also referred to as a module that is present in software architecture. This module acts as any one of the following component,

(i) Control Component

This module is responsible for coordinating the invocation of the problem domain components.

(ii) Problem Domain Component

This module is responsible for implementing entire or partial function which is required by the customer.

(iii) Infrastructure Component

This module is responsible for invoking the functions, which support the processing performed in problem domain component.

The conventional software component is derived from the analysis model, whose data flow elements are used as the underlying concept for the derivation. Let us consider an example of xerox center in order to understand the process of design elaboration.

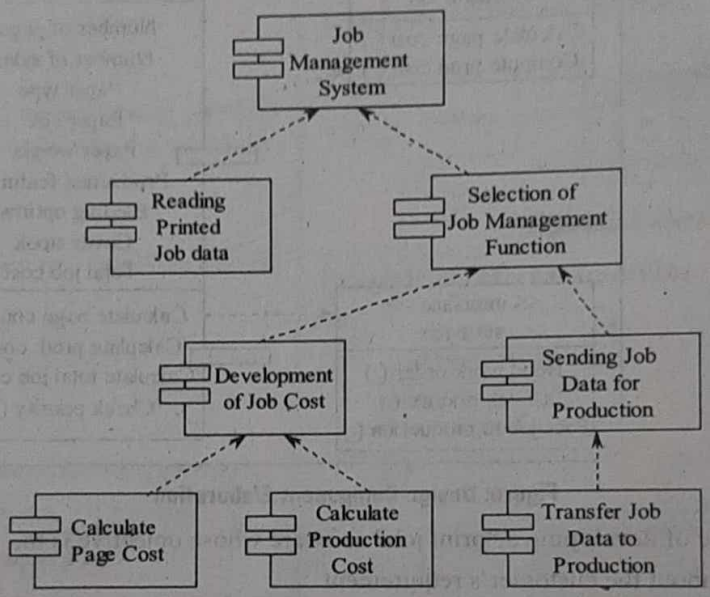


Figure: Structure Chart for a Conventional System

In the above figure, control components are placed at the top of hierarchy and the problem domain components are placed at the bottom of the hierarchy. Here, every component is represented using the boxes and all operations are represented using separate module. The other modules are used to control processing and hence referred as control components.

The designer during the component-level design, elaborates every module and explicitly defines the module interface such that every data or control objects that flow across the interface are represented. The elaboration is done by defining the data structures (used internal to the modules) and the algorithm (used for enabling every module to perform its intended function).

3. Process-related View

Process-related view performs the component level design by using the existing software components. That is, unlike object-oriented and conventional views, process related view does not design the component from the scratch. While the architecture of component is being developed software engineers use the catalog of the existing design for selecting the design patterns. As the components are designed by considering the erasability factor, the designer have complete information about,

(i) Interface of component

(ii) Function of component

(iii) Communication and collaboration between the components.

Q37. Discuss briefly on data design at deployment level.

Answer :

Deployment level design elements indicates the physical arrangement of computing environment. It shows how various software elements are linked with other subsystem in actual deployment site.

The deployment diagram is created during design phase and is refined in later stages. Each computing environment consist of several subsystems, which are extended to define its implementing component. Consider the example of a safe home system.

Example

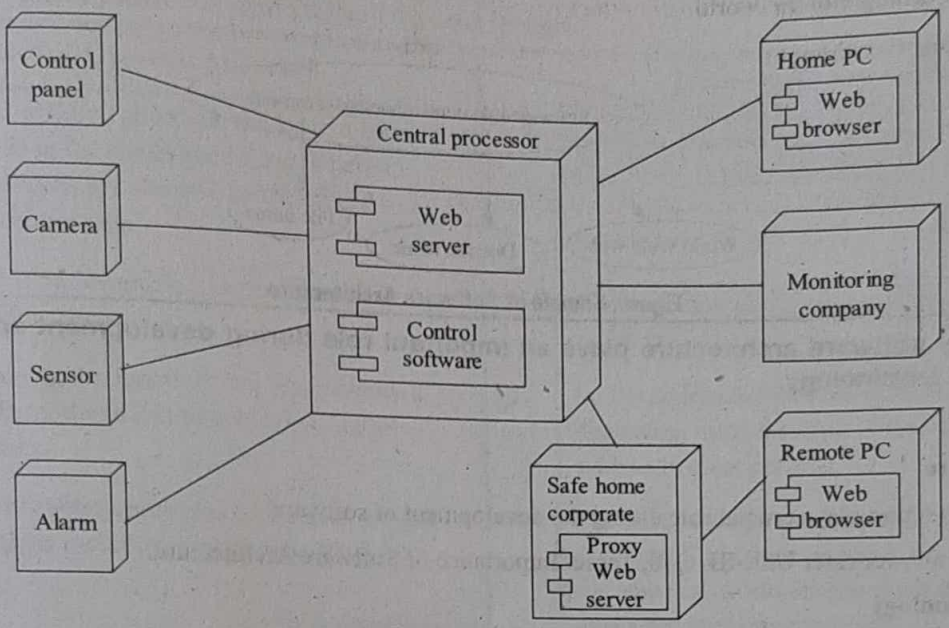


Figure: Deployment Diagram for Safe Home System

This deployment diagram shows 9 computing environments with both the hardware and software components in the system. The web server and the web browser are generic components. Control software is the important software component in the safe home system.

The Central Processor (CP) is at the customers site. The CP software can run on any system. It is connected to safe home corporate site and a monitoring company via a broadband connection or a modem. The central processor assures system security. It has cameras, sensors and alarms connected to it. There is a wireless communication between control panel and central processor. To access the complete functionality of the system, users connect to the central processor via web browser. The web server is run by computer processor. The safe home corporate website gives the information of the system when the user is travelling.

3.2 CREATING AN ARCHITECTURAL DESIGN

3.2.1 Software Architecture

Q38. What is software architecture? Why is it important?

Answer : **May/June-12, Set-3, Q8(a)**

Software Architecture

Software architecture is structure of system that consists of certain number of components, along with their visible features and relationships among them.

A software engineer uses software architecture in many ways as it provides specifications, through which he sets his goals, provides information related to modifications to be made to the design at the initial software development stages etc. Hence, he can get rid of potential risks easily at the beginning stages of software development.

Importance of Software Architecture

There are three reasons for the importance of software architecture in the real world software development.

- A software architecture describes a scenario, through which various stake holders having interest in the software can easily communicate with each other.
- It provides an overview of each software development aspect which remains important for overall success of the software being developed.
- Finally, it describes the structure and the working of various components which collaborates to form a single software.

Apart from above specifications, it has to be noted that the designs and patterns of the architecture can be interchanged. This means that these patterns can be applied to other system designs. This enables to describe software architecture in predictable ways.

Example for an Software Architecture

A Uniform Resource Locator (URL) is an example for a software architecture which displays a group of web pages. These web pages contain entire information about the desired site provided in the URL http:// www.google.com. Here, one can search the entire information throughout the world.

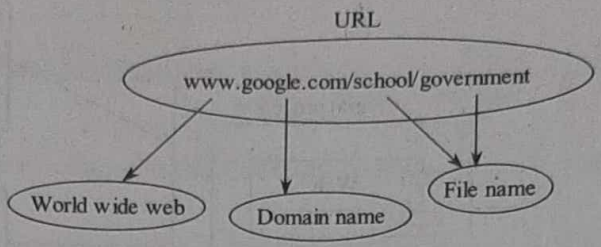


Figure: Sample of Software Architecture

Q39. Discuss why software architecture plays an important role during development and discuss various architectural terminology.

Answer : Nov./Dec.-12(R09), Q4(b)

Software Architecture

Software architecture plays crucial role during the development of software.

For remaining answer refer Unit-III, Q38, Topic: Importance of Software Architecture.

Architectural Terminology

1. Architecture
2. Architectural description
3. Architectural decision.

1. Architecture

A software architecture is a structure of a system that consists of certain number of components, along with their visible features and relationships.

A software engineer uses software architecture in many ways like use of specifications, through which he sets his goals, information related to modifications to be made to the design at the initial software development stages etc. Hence, he can get rid of potential risks easily at the beginning stages of software development.

2. Architectural Description

The description of architecture differs from one stakeholder to another stakeholder because stakeholders have their own perspective. Therefore, the architectural description is a set of work product that represents various view points of a system.

In software development, developers have their own perspective of architecture and testers have their own perspective. The IEEE standard proposed the following objectives of architectural description.

(i) Form a conceptual framework and vocabulary to be used for developing software architecture.

(ii) Offer guidelines to represent architectural description.

(iii) Support architectural design practice.

Further, architectural design is defined as a set of product in order to capture architecture in a document. The design is depicted in various forms where in each view represents complete system from a stakeholders view.

3. Architectural Decision

Architectural description focuses on a particular stakeholder's concern using views. It is developed by considering multiple alternatives and finally deciding a specification.

The view is selected according to the criteria specified by the user. Therefore, architectural decision are self considered as one view of the architecture. The decision is based on insight of the system and in regard to the concern of stakeholders.

Q40. Define software architecture. Explain why it may be necessary to design the system architecture before the specifications. Compare function oriented and object oriented designs.

Answer :

(Model Paper-III, Q6(a) | Nov./Dec.-18(R16), Q6(a))

Software Architecture and its Importance

For answer refer Unit-III, Q38, Q39.

Difference between Function Oriented and Object Oriented Designs

	Function Oriented Approach		Object Oriented Approach
1.	In functional oriented approach the system state information is in the centralized form, where in different functions are allowed to access and modify the central data.	1.	In object oriented approach, the state information is distributed between distinct objects of the system.
2.	The basic unit of designing functional a oriented approach is functions.	2.	The basic unit of designing object oriented approach is object.
3.	In this approach, the functions are represented in a grouped form, that helps in achieving higher-level functions.	3.	In this approach, the functions are grouped together depending upon the type of data they operate on such as in class person.
4.	The basic abstraction provided to the user are real world functions including sort, merge, track, display.	4.	The basic abstractions provided to the user are not real-world functions, instead data abstractions become real-world entities such as picture, machine, customer, student, employee.
5.	Functions appear as verb in problem description.	5.	Objects appear as nouns in problem description.

3.2.2 Data Design

Q41. What is meant by data design? Explain with an example.

Answer :

Dec.-14(R09), Q4(b)

Data Design

Data design refers to a criterion which is useful in transforming data objects into data structures at software component-level, further into a database architecture at the application level.

Steps for Constructing a Data Design

1. Data must be organized orderly – It contains the data flow, its relationship and data objects.
2. Data must include the methods and the data structure. All the methods must be defined to work with the data structure.
3. Provides a data dictionary – Data dictionary defines the elements that are used in data design such as constraints, data objects and their relationship.
4. A data design provides low level design after defining the structural attribute. First, design the architecture of the system, which contains data structure before designing the low level design.
5. A data design provides information hiding – Information hiding plays a vital role in designing a quality software product which gives the comparison between physical and logical views.
6. A data design provides library which defines the methods and data structure which are used in designing the system. The library provides the reusability of data structure which decreases the time for defining the data.
7. Data of the system can be specified by using the programming languages along with software design – By using programming languages and designing model the data of the system can be identified.

For remaining answer refer Unit-III, Q42.

Q42. Discuss briefly on,

(i) Data design at the architectural level

(ii) Data design at the component level.

Answer :

(i) Data Design at the Architectural Level

In today's world, the enterprises maintain large sets of databases irrespective of whether they are small or large. In such circumstances it often turns out to be crucial to acquire useful information from such databases especially when each of these databases are not correlated. Hence, to manage such circumstances, several researchers had put their efforts and concluded with data mining techniques also called knowledge discovery in databases (KDD). But due to many reasons, these researchers turned their attentions towards other techniques due to run time failure of KDD. This is because, due to the existence of large number of databases, their details of storage, their structures etc., differ significantly causing failure of KDD. Nowadays each of the modern business enterprises immensely depend on new type of technique called "data warehouse".

The data warehouse itself refers to a huge database that has the access to data that is stored in database which is required by a business.

(ii) Data Design at the Component Level

Whenever we deal with data designing at component level, it reflects the representation of data structures accessible by various components forming a given software.

There a set of principles for data specification and design.

Principle 1

The specification and realization of abstract data types should be supported by a software design and programming language.

At times the most sophisticated data structures for which its implementation goes to highest level of complexity. This usually happens since the programming language utilized for implementation of such data structures does not include mechanisms to support it.

Principle 2

Decisions on low-level data design should be made late in the design process.

We should use the sequential or stepwise refinement process while designing data. This can be done as follows.

- ❖ Begin the process of data organization at requirement analysis phase.
- ❖ Refine this organized data at data design phase.
- ❖ Include the details of refined data at final phase i.e. component level phase.

Principle 3

The data design should identify a useful set of data structures and operations that can be performed on them.

It is prescribed for an efficient data structure to possess information on various types of operations implemented on it. This can be achieved by using a class.

Principle 4

Only those methods that directly use the data within the structure should know the representation of a data structure.

The above principle refers to two important ideologies i.e. information hiding and coupling. These two concepts when implemented in current data designing, yield high quality software.

Principle 5

There should be a mechanism that defines the content of each object and allows to define data and operations to be performed on it.

This phenomenon can be observed in class diagrams where it defines data items of a class and various operations applied to these items.

Principle 6

The system analysis principle for data should be the same as applied to function ad behaviour.

It initially begins by developing and reviewing the representations consisting of data flow and various contents. Later, the data objects should be identified. If there exists any alternative data organizations, then they should be considered. Finally, the effect of data modelling over the software design should be determined.

3.2.3 Architectural Styles and Patterns

Q43. What is architectural style? Discuss various categories of it. Model Paper-II, Q6(b)

OR

Write the taxonomy of architectural styles and give a brief description of each style.

Answer :

Architectural Style Nov.-15(R13), Q7(a)

For answer refer Unit-III, Q34, Topic: Architectural Style.

Various Categories of Architectural Styles

(a) Data Centered Architecture

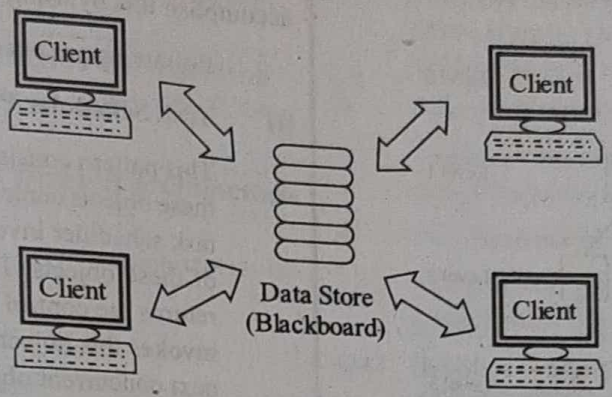


Figure: Data Centered Architecture

Important points related to above architecture are illustrated below.

- In the above architecture, essential data reside at the centre of the architecture.
- All the client software are authorized to access this data.
- These client software can easily manipulate the centered data i.e., they can delete, update, add etc., independently.
- As the data manipulation can be done independently, the centered data can be transformed into a blackboard which notifies the client when data of his interest changes.
- The architecture supports integrability which results from the fact that the architecture promotes independent access and manipulation to data store.
- All the client software are authorized to access several processes independently. The data store can also act as a blackboard, hence, clients can send and receive messages among themselves.

(b) Data Flow Architecture

Diagrammatic representation of data flow architecture is shown below.

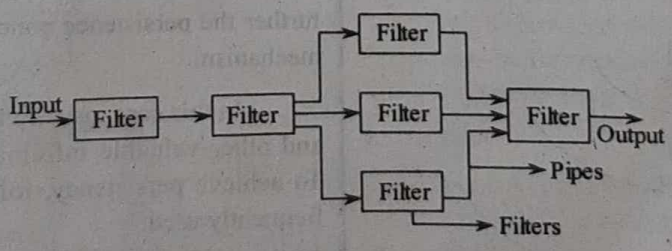


Figure: Data Flow Architecture

This architecture is applied when the given data is to be transformed into certain output by applying series of components (filters), this architecture consists of a set of components called filters, that are connected to each other by points for transmitting data. Each filter transforms the data received by it into certain output independently and delivers it to other filters. Also, each of these filters are independent of the functioning of all other filters. At times, it happens that a condition is met referred as batch sequential, in which the data flowing gets degenerated into a single line of transforms. In those conditions, the architecture accepts the data and transforms it into certain output using one or more filters.

(c) The Call and Return Architecture

The call and return architecture is best known to frame a given software such that the resultant architecture can be modified or shrunk depending on the requirements. This architecture has got two sub-architectures referred as main program/subprogram architecture and remote procedure call architecture.

Main Program/Subprogram Architecture

As the name suggests, this architecture decomposes main program into a number of constituent program components. These program components can further be decomposed into other components.

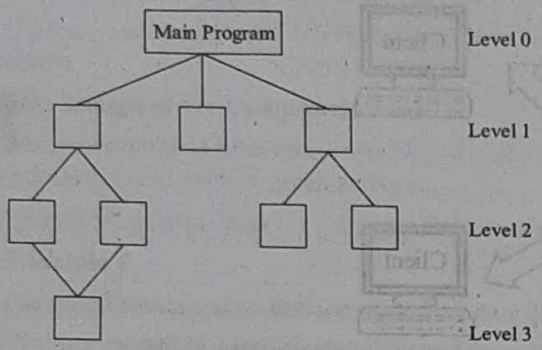


Figure: Main Program/Subprogram Architecture

The above diagram depicts the main program/ subprogram architecture. The level-0 consists of main program, the level 1 consists of controlled subprogram and level 2 and 3 consists of application subprogram.

(d) Layered Architecture

There are basically four layers in a layered architecture. The rectangular small boxes in each layer are the components associated with that layer. Each of these layers defines a definite set of operations. The deeper layers, are nearer to the machine instruction. The outermost layer is user interface layer, where several components perform operations related to user interface. By moving further we encounter application layer, utility and core layer are encountered.

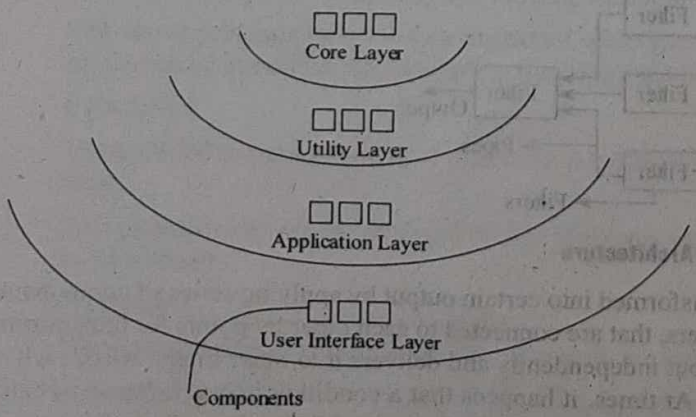


Figure: Layered Architecture

Q44. Explain the following terms with respect to architectural patterns,

(a) Concurrency

(b) Persistence

(c) Distribution.

Answer :

(a) Concurrency

In today's world, almost every software application is suitably crafted to achieve parallelism. An application can accomplish this by using different architectural patterns.

Following are examples of two architectural patterns.

(i) Task Scheduler Pattern

This pattern consists of multiple active objects. Each of these objects contain a special operation called tick. The task scheduler invokes this operation that activates one of these objects. The object performs its function and returns the control to the scheduler. The scheduler then invokes this object by executing tick (✓) and activates next concurrent objects.

In this way, all the objects gets executed periodically.

(ii) Operating System Process Management Pattern

In this pattern, operating system concepts are taken under consideration to achieve concurrency. Apart from this, the other features of operating system, such as establishment of communication between multiple processes, scheduling etc., are also implemented.

(b) Persistence

Any data is said to be persistent if it retains even after the completion of execution of a process that has created it. In any organization, the immediate way to store the persistent data is either in database or in the form of system files.

The introduction of object oriented terminology moved further the persistence concept by means of persistent object mechanism.

In this case, usually the state of objects, their attributes and other valuable information are stored to be used later. To achieve persistency, following architectural patterns are frequently used.

(i) Database Management System Pattern

Any application can implement the database management system pattern for orderly storage and retrieval of data.

(ii) Application Level Persistence Pattern

In this pattern, the persistence features are built into the application architecture.

(c) Distribution

In this pattern, the process of communication is implemented between various components of a given system (when they are far apart) is of prime focus. Hence, while implementing such processes, following two aspects are needed.

(i) Connection among components

(ii) The type of communication.

To deal with above two aspects, the most commonly used pattern is *broker pattern* where the broker is just like an intermediate component (usually like a software residing at server). It is capable of accepting, processing and redirecting the queries. Whenever it receives a query from any of the clients, it just processes and accordingly perform actions.

Q45. Write about architectural styles and patterns.

May-18(R15), Q6(a)

OR

List and explain different kinds of architecture styles and patterns.

Answer : **March-17(R13), Q7**

Architectural Styles

For answer refer Unit-III, Q43.

Architectural Patterns

For answer refer Unit-III, Q44.

3.2.4 Architectural Design

Q46. Draw the architectural context diagram. Explain different parts used in this diagram.

Answer :

Architectural Context Diagram

The architectural context diagrams are used to diagrammatically represent the scenario in which a given software performs interaction with various external components located far from its boundary. Following is the diagrammatic representation of the general architectural context diagram.

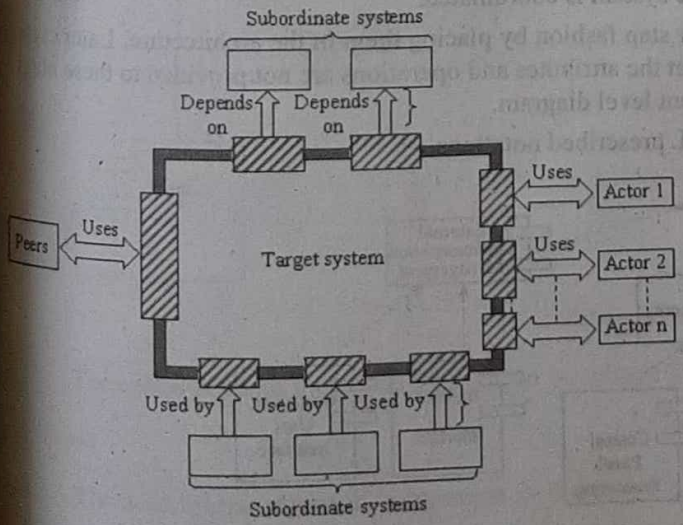


Figure: Architectural Context Diagram

Components

The description of essential components of the above system is given below.

1. **Actors**

   These are the entities that possess a definite set of roles and interacts with the system. During this interaction, an actor can either provide or accept information from the system.

2. **Subordinate Systems**

   These are the systems which function along with the target system. Thereby, supporting it in successfully completing its processing.

3. **Superordinate Systems**

   These are the systems used by the target system for completing few of its higher valued activities.

4. **Peer Systems or Peers**

   These are the systems which directly interact with the target system (same as client-server interaction).

Q47. What is meant by "archetype"? With the help of a diagram explain few archetypes.

Answer :

Archetype

In general terms an '*archetype*' describes a pattern which is essential in designing the final or target system. In the target system, there may be certain stable elements. However these elements may even be changed to represent certain other elements depending on the behavioural aspects of the system. Only few sets of these archetypes are enough to form the constituent entities of any critical architecture. Archetypes can be identified directly by considering the analysis classes which form the outcome of the analysis phase. Some of the archetypes for safe home system are as follows.

1. **Controller**

   It refers to an abstract representation. This representation usually refers to a mechanism using which a given node can be provided with or without the authority. They can also be represented on networks which assist them in communicating with each other.

2. **Node**

   It refers to an entity which depicts the cohesive collection of several input and output elements pertaining to 'Safe Home' system.

3. **Indicator**

   It depicts an abstract representation of several mechanisms that indicate an alarming condition.

4. **Detector**

   It is also an abstract representation that refers to a sensing entity which may deliver information to the end system.

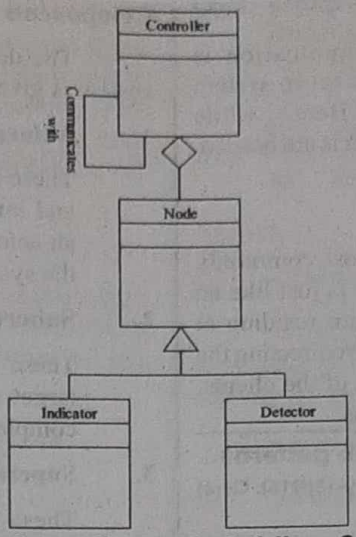


Figure: UML Archetype Notation for SafeHome Security System

As the above represented archetypes represents only abstractions, they can be refined into further components by refining these abstractions.

Q48. With the help of a diagram explain the process of refining the architecture into components.

Answer :

Process of Refining Architecture into Constituent Components

The process of refining the architecture into its constituent components marks the beginning of the structure for the end system. For this purpose, initially the classes which were acquired during the analysis phase of the software development process are considered. These classes form the major entities of an application domain and therefore they are the important aspects of an end system. The development of structure representing the system must also address the infrastructure domain i.e. the end system must also include infrastructure components which support the components of an application domain.

For example, if 'SafeHome' security system is considered, few highly valued components are defined as follows.

- **Alarm Processing**

  Capable of processing, detecting and also responding to all alarm conditions.

- **Control Panel Processing**

  Here, control panel related activities are managed.

- **External Communication Management**

  Here, the activities involved in providing a security function communicating with various external entities are coordinated and managed.

- **Detector Management**

  Here, usually the accessing of detector associated with the system is coordinated.

Now, the above mentioned entities are refined in a step by step fashion by placing them in the architecture. Later classes can be attached to them. While doing so, one has to remember that the attributes and operations are not provided to these classes directly. They are provided at the time of beginning the component level diagram.

Following is an example architecture designed using UML prescribed notations.

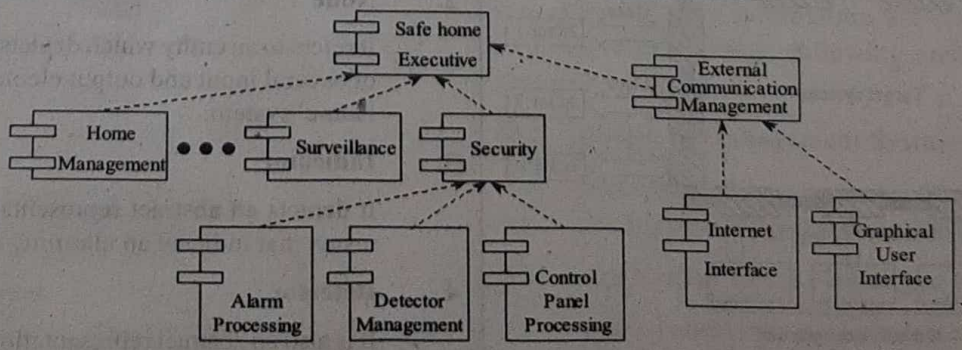


Figure: Refining of Architecture into Components

Q49. What is instantiation? How would you describe instantiations of the systems?

Answer :

(May/June-12, Set-2, Q2(a) | Dec.-11, Set-3, Q2(a))

Instantiation

From the programming perspective, instantiation is a process of creating real instance or realization of a computer process.

Instantiation of the System

Instantiating of an architecture is necessary whenever the following aspects of designing are completed.

- The context of the system is developed.
- The archetype has been defined.
- The overall system architecture is identified.
- Finally, all the important components belonging to the end software are recognized.

Now the instantiation process is initiated. In this, the product with an intention is displayed which it is well suited for the given problem, all its accessories are perfectly connected and the architecture structure so formed is ultimate in all aspects.

Following is an example diagram depicting the instantiation of architecture for the '*SafeHome*' security system architecture.

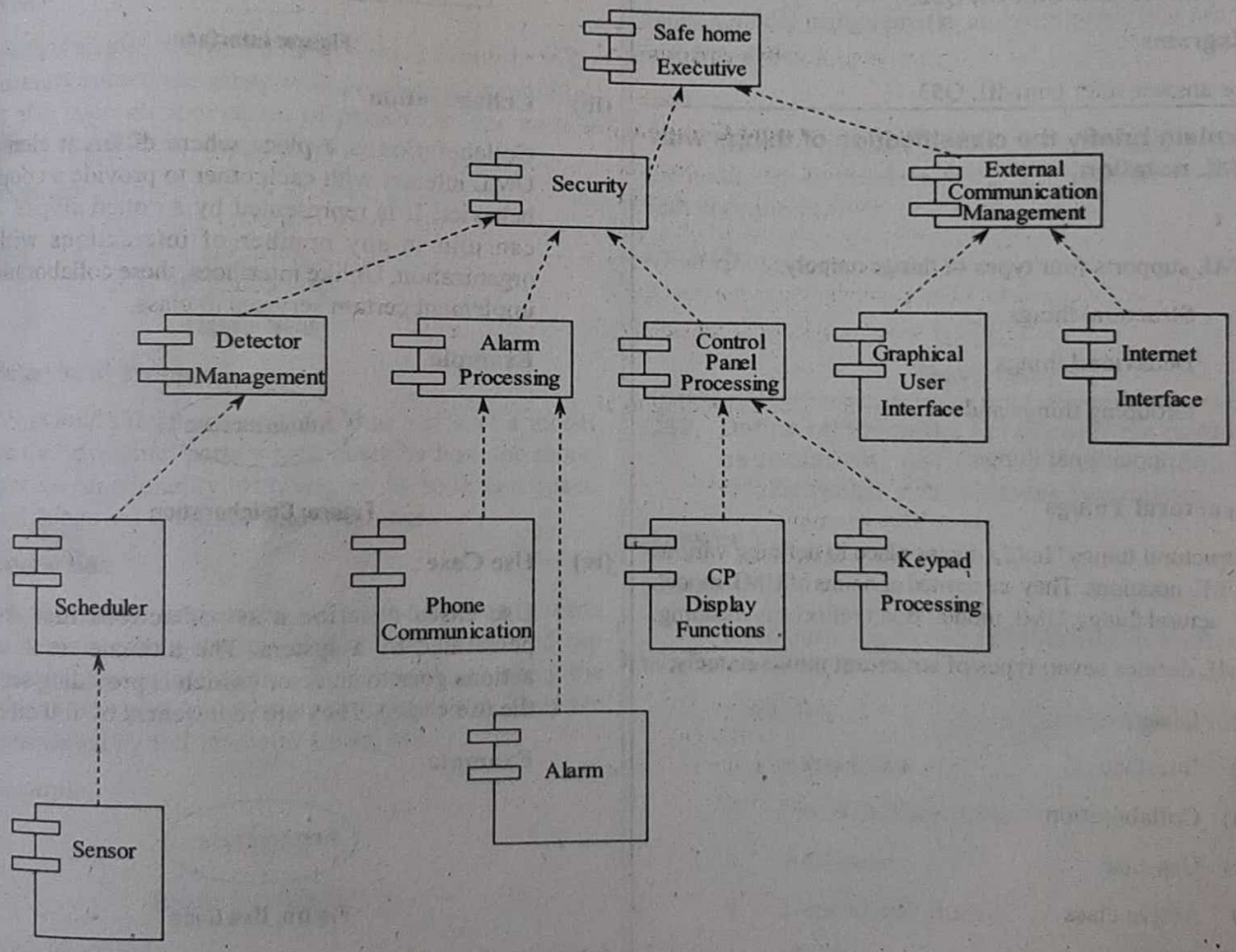


Figure: Instantiation Process

The components of the above figure must be refined further to elucidate the additional details.

Example

The detector management component in the above figure communicates with the sensor component for implementing concurrent polling of all the sensor objects.

3.3 CONCEPTUAL MODEL OF UML

Q50. What are the categories of building blocks in the UML? Explain any one category of building blocks.

Answer : Model Paper-III, Q7

UML is based on three major building blocks, namely,

1. Things or basic elements.
2. Relationships between these things.
3. Diagrams which are meaningful organization of these things.

1. Things

For answer refer Unit-III, Q51.

2. Relationship

For answer refer Unit-III, Q52.

3. Diagrams

For answer refer Unit-III, Q53.

Q51. Explain briefly the classification of things with UML notation.

Answer :

UML supports four types of things namely,

1. Structural things
2. Behavioral things
3. Grouping things and
4. Annotational things.

1. Structural Things

"Structural things" hold a major place in defining various
ir ML notations. They are termed as nouns of UML model.
uctural things, UML models don't reflect any meaning.

ML defines seven types of structural things namely,

(i) Class
(ii) Interface
(iii) Collaboration
(iv) Use case
(v) Active class
(vi) Component and
(vii) Node.

(i) Class

It can be defined as a collection of objects that share common properties and relationships. Every representation of class includes name, attributes and responsibilities.

Example

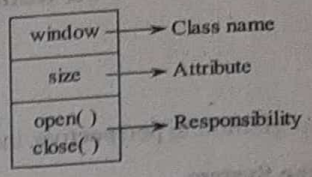


Figure: Representation of Class

(ii) Interface

It defines service to each class. However, implementing these services is not a responsibility of an interface. Its declaration is similar to class but includes <<interface>> in its header while excluding attribute function.

Example

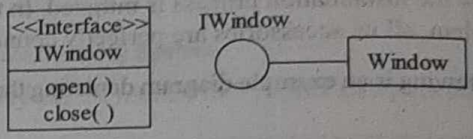


Figure: Interface

(iii) Collaboration

Collaboration is a place where different elements of UML interact with each other to provide a cooperative behavior. It is represented by a dotted ellipse. A class can join in any number of interactions within the organization. Unlike interfaces, these collaborations can implement certain services to class.

Example

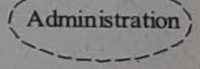


Figure: Collaboration

(iv) Use Case

Use cases describe a set of actions that are to be performed by a system. The ultimate result of these actions goes to an actor (which is providing services to the use cases). They are represented by full ellipse.

Example

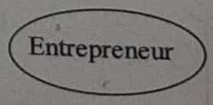


Figure: Use Case

(v) Active Class

Active class possesses objects which consist of processes and threads. These processes and threads are essential in initiating control over the functionality associated with class. The difference between a class and an active class is that its functionality depends on several other classes that are in contact with it.

Example

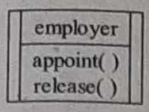


Figure: Active Class

(vi) **Component**

Components bind several classes, interfaces and collaborations by physical representation. The representation might include various connectors connecting these components to carry out a specific functionality.

Example

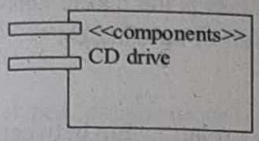


Figure: Components

(vii) **Node**

Node is an element which holds certain locations in the memory sometimes along with processing capabilities. It also supports movement of parts from one node to another node. It is always represented by a cube.

Example

Figure: Node

2. **Behavioral Things**

Behavioral things are considered as verbs of a model. These are the 'dynamic' parts which describe how the model carries out its functionality with respect to time and space. Behavioral things are classified into two types.

(i) **Interaction**

As the name indicates, interaction represents communication between objects that is carried out by exchange of messages. These messages carry the actions to be performed by a particular object. They are represented by full line with a dark head.

Example

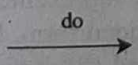


Figure: Interaction

(ii) **State Machine**

State machine represents various states through which an object passes in its life time to perform a particular task.

Example

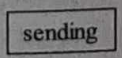


Figure: State Machine

3. **Grouping Things**

Grouping things deal with the "organizational" features of UML. They are represented by boxes and functionalities of models are decomposed into it. An example of grouping things is a "package".

Package

It organizes various structural and behavioral things into groups known as package.

Example

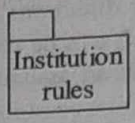


Figure: Package

4. **Annotational Things**

It provides the description of various parts of UML models. Annotational things provide additional information about a model using remarks and comments that are included as a stick note.

Note

A note is typically a symbol used for adding constraints and comments to the elements. It is represented as a rectangle with dog-eared corner.

Example

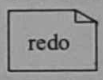


Figure: Note

Q52. Define relationship. Briefly explain dependency, association, generalization and realization relationships with suitable examples.

Answer :

Relationship

A bond existing among several elements is often referred to as a relationship.

UML defines four types of relationships as follows,

1. Association
2. Generalization (is-a)
3. Realization
4. Dependency (using).

1. Association

Association is a relationship among two or more objects. It is a structural relationship wherein users can navigate from an object of one class to an object of other class and vice versa. An association can connect back to the same class. It can be binary (or) n-ary where a binary association connects two classes and an n-ary association connects to more than two classes.

An association name is used to describe the nature of the association.

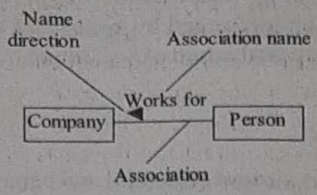


Figure (1)

The association "works for" between the class Person and Company has a name direction (a triangle) pointing from Person to Company which can be read as "Person works for Company". Every class in an association plays an important role.

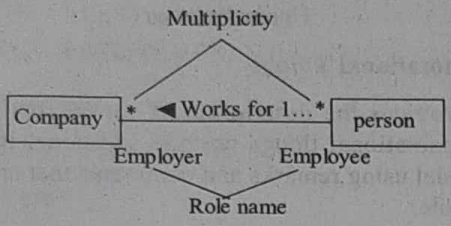


Figure (2)

Here the class 'company' plays the role of an employer whereas the class Person plays the role of employee. The roles are named exclusively. Multiplicity is used to indicate number of objects that can be connected across an instance of an association in the form of an explicit value or a range of values.

(0..1) Zero to one

(1) Exactly one

(*) Many

(1..*) One to many

(1..5) One to five.

2. Generalization

Generalization is a relationship between a more general element and a more specific element. The more general element is called the parent or super class and the more specific element is called the child or subclass. The parent describes a set of instances with common properties over the children. Whereas the child describes those instances which have the properties of the parent and additional properties peculiar to itself.

In a simplest manner, a class which has a single parent is called single inheritance. The child that inherits structure, behavior and constraints from all its parents is called multiple inheritance.

Moreover, the class that does not have parents is the root class and a class with no children or sub classes is called a leaf class.

When operations in child classes override the behavior of the operation of parent class it is called polymorphism, or if the same operation behaves differently on different classes. Generalization is also called as "is-a-kind-of" relationship.

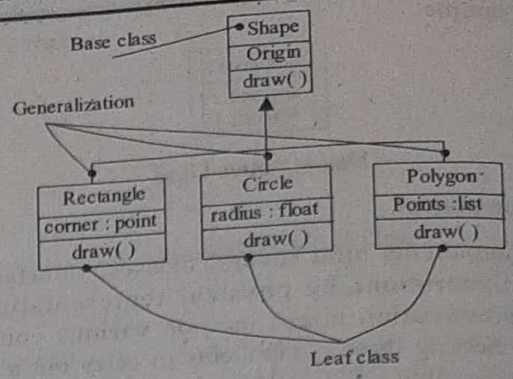


Figure (3): Generalization

Here, Shape class is the parent class and the other three classes are the children or subclasses.

3. Realization

Realization is a relationship between classifiers in which one classifier lays down a functionality and another classifier guarantees to carry out its functionality. In a realization relationship, usually the classifiers used are classes and interfaces, whereas an interface specifies the function that a class or component should carry out.

A realization relationship is shown by a dashed path with a closed triangular arrow head on the end adjacent to the element supplying specifications and its tail on the element supplying the implementation.

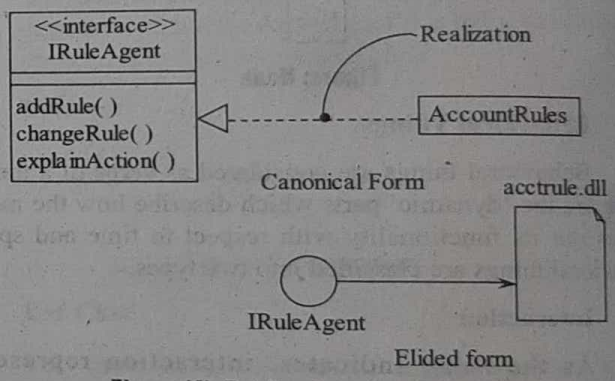


Figure (4): Realization of an Interface

In the given diagram, a class AccountRules realizes an interface IRuleAgent. Also, a component acctrule.dll realizes the interface IRuleAgent.

The realization is represented in two ways i.e., the canonical form and the elided form. In the canonical form, the interface Stereotype is used and a directed dashed line with a large open arrow head. Whereas in the elided form the interface is used in the lollipop notation.

4. Dependency

Dependency is a relationship between two elements in which a change in one element may effect the other element that uses it. This may not apply in the reverse manner. For example, Child is dependent on parent but not vice versa. It is represented as a dashed line in the direction of the element being dependent on.

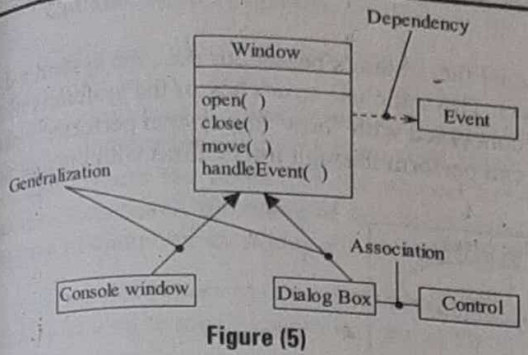


Figure (5)

From the diagram, any change in the class Event may affect the class Window but this might not be not possible in reverse way. The class window uses the class event, therefore if the used class event changes, the operation of the class Window is affected.

Whenever a dependency is modeled, it should be pointed from the class with the operation (window) to the class used as a parameter to an operation (event class).

Q53. List and explain various diagrams that the UML contains.

Answer :

UML Diagrams

UML diagrams reflect the graphical representation of a system to be developed. There are nine sets of UML diagrams, which are grouped into two categories.

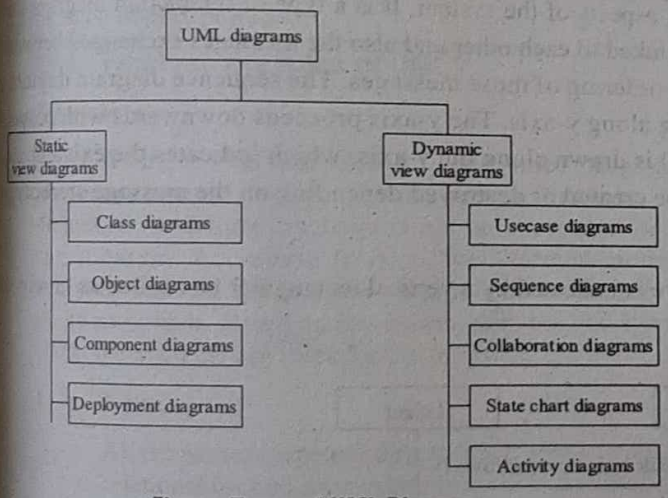


Figure: Types of UML Diagrams

1. Class Diagrams

These diagrams reveal the static design/view of a system and include few active classes which are used in capturing the static process of overall system. The class diagram consists of classes, interfaces and collaborations. The main purpose of class diagrams is to visualize, specify and document the structural model.

2. Object Diagrams

These are also the kind of class diagrams used in revealing static design view or static process view of a system. The uniqueness of these diagrams is that it illustrates data structures and the functionalities at a particular time of elements existing in class diagrams.

3. Component Diagrams

A component is a physically existing part (usually files) in a system. Executable files, .dlls, object files etc., can be represented as various components which can exist on a node. They are replaceable and supports openness and adaptability. The diagrams which incorporate these components along with their relationships to model the physical aspects of a system are called as component diagrams. The class diagrams show logical aspects of a system where component diagrams are used to decide which physical things would be available on a node like executable files, source codes, .dlls, tables etc. It is just a collection of vertices and arcs.

When a decision is required to decide the way in which the source code can be organized or the way the database is to be laid or the manner in which the executable files of a project are to be implemented, component diagrams are used. A sample component diagram is shown in figure.

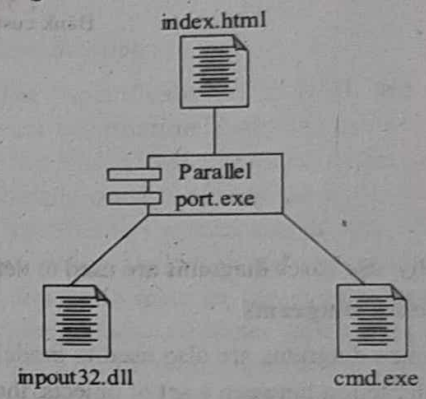


Figure: Component Diagram

4. Deployment Diagrams

A deployment diagram is a special type of class diagram that is used to represent the configuration of nodes along with their residue components.

These diagrams may include deployment of various processors and peripheral devices on which the software is implemented. Thus, it can be possible to visualize the layout of these physical nodes and the way these nodes link with each other. It is nothing but visualizing a scenario of nodes of the systems and their internal components at run-time. The deployment diagram is a collection of vertices and arcs that contain nodes and their associated relationships.

Example

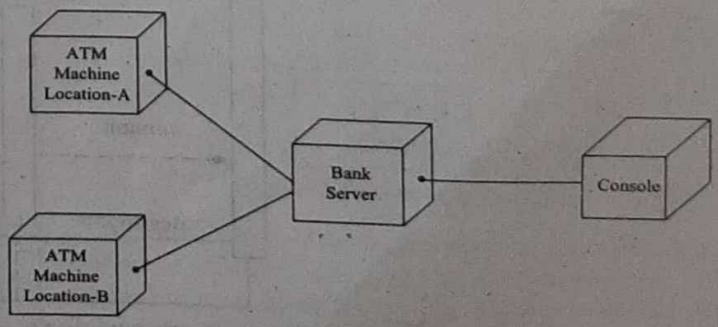


Figure: Deployment Diagram

5. Use Case Diagrams

These diagrams are the behavioral diagrams which are used to model the system's behaviour (i.e., the system's dynamic aspects. It contains actors, use cases and the way both relate to each other). The static use case view of the system is illustrated through these diagrams. It depicts "what" the system performs and is not concerned with "how" the system performs a particular task. The use case within a diagram defines a set of actions that a system can perform through interactions with outside actors or users.

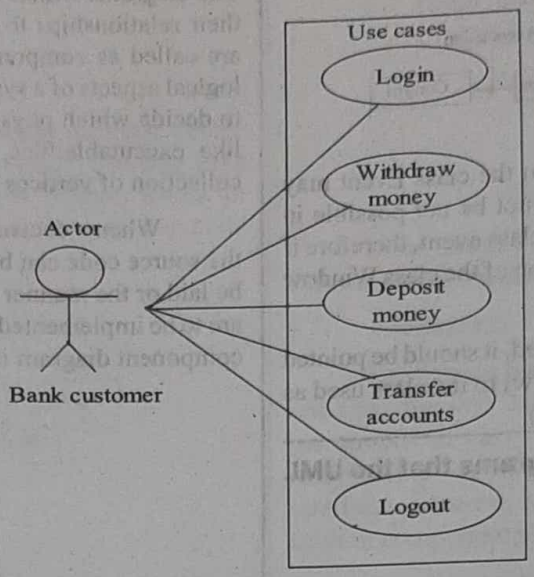


Figure: Use Case Diagram

Usually, use cases diagrams are used to define the system's context and its various requirements.

6. Sequence Diagrams

Sequence diagrams are also used to model the behavioral aspects of the system. It is a type of interaction diagram that shows the interaction between a set of objects, the way they are linked to each other and also the messages exchanged between them. The objects interact among themselves with respect to time ordering of those messages. The sequence diagram depicts a graph with objects placed along the x-axis and message passing along y-axis. The y-axis proceeds downwards with respect to increasing time. Under each object, its life line (dashed line) is drawn along the y-axis, which indicates the existence of the particular object for a particular time period. Objects will be created or destroyed depending on the message stereotype received which can be either "create" or "destroy".

The time duration during which a particular object is active is indicated by a vertical rectangular box called as "focus of control".

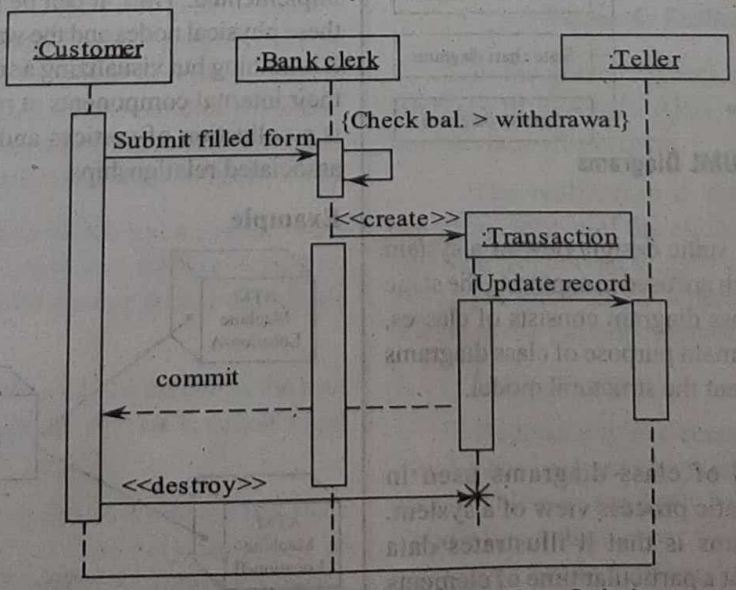


Figure: Modeling of Flow of Control by Time Ordering

7. Collaboration Diagrams

Collaboration diagrams organize various objects participating in an interaction in sequence. These objects can interact with each other using various messages. Thus, in short collaboration diagrams consist of objects, links and messages where the names of the objects are instance of classes. Sometimes, the name of an object also refers to the instance of components, nodes etc. to which they are related.

The most important feature of the collaboration diagram is that, it is used to model dynamic view of the system.

8. State Chart Diagrams

State chart diagrams illustrate the behaviour corresponding to an interface, collaboration or a class hence, in this way it reflects the dynamic aspects of a given system to be modeled. State chart diagrams encompass state machine which specifies various states, transitions, events and activities respectively.

9. Activity Diagrams

These diagrams are considered as effective while modeling functionalites of a given system. Hence these diagrams reflect activity(ies), the type of flows between these activities and finally the response of objects to these activities. In this way, it reflects dynamic aspect of a given system to be modeled while prioritizing the flow of control among objects participating in the system. The above list of diagrams are not sufficient to model a complex software project. Many tools use UML to extact different kinds of diagrams that are sufficient enough to model complex projects.

Q54. What are the rules of UML?

Answer :

A language is said to be a 'well-formed' language, if it satisfies certain predefined rules or specifications. UML or unified modelling language is not an exception for such specification. According to it, a "well-formed model" is equipped with all the essentials and reflects uniformity with all its co-models. Based on this assumption, the developers of UML specified certain rules for the following entities.

1. Names

All the names corresponding to UML's basic entities, its relationships and its supporting models are categorized under this specification.

2. Scope

Scope is the context that assigns certain meaning to the name.

3. Visibility

Visibility in this case refers to the way the names are visioned and utilized by other entities.

4. Integrity

Integrity corresponds to the process of collaborating all the entities into single instance.

5. Execution

Execution refers to the process of triggering the final model.

It has to be noted that every "end model" may not be "well-formed" because, each phase in software development process may unveil new instances. Also, the visualization of a given project may vary with respect to different stakeholders. Hence, with these reasons the models developed may remain inconsistent, incomplete and elided (i.e., hiding few specifications).

Q55. Discuss in detail,

(a) Specifications

(b) Adornments

(c) Common division.

Answer :

(a) Specifications

The "specifications" in UML are used to provide the textual information about the system which is under construction. Using UML, the model of the system is developed which initially consists of only the UML diagrams. Now, the use of 'specification' comes into picture which provides the information about the syntax and semantics of all the building blocks that participate in the system development. This information includes attributes, operations and behaviors of the classes. With use of this approach along with the graphical notation contained in UML, an effective model can be modeled by including the semantics based on the types of diagram used.

(b) Adornments

Every element in UML possesses its own specification and representation. These representations are formatted and designed in such a way that, the element should look simpler. Moreover, every element has a unique style of representation.

For example, consider the representation of class as shown below,

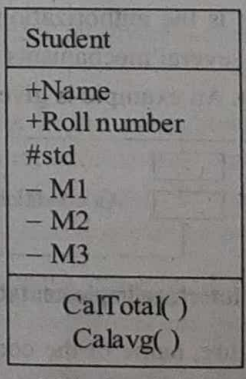


Figure: Adornments in the Class

In the above example, student is the class name. It consists of student's name and roll number as public values, standard as protected and marks M1, M2, M3 as private values. They are represented using predefined format.

(c) Common Divisions

In UML "common divisions" refer to the separation of entities according to their type. Every tool in UML is divided into two categories namely,

(i) Objects/Classes

(ii) Interface/Implementation.

(i) Objects/Classes

A class is said to be a collection of objects where an object represents a real world entity.

Modelling of class and object is given below,

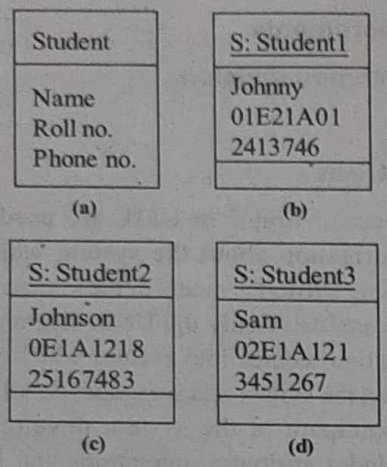


Figure: Representation of Objects and Classes

In the above figure (a), a class is represented by a class name "Student", with Name, Roll no. and Phone no. as its attributes. Figures (b), (c), (d) are the objects of student class. The attributes of the objects are Name, Roll no. and Phone no. of the respective students.

(ii) Interface/Implementation

The second common division is interface/implementation. Interface is a contact to carry out certain services and implementation is the authorization of these services. UML provides several mechanisms to model interface/ implementation. An example is given below,

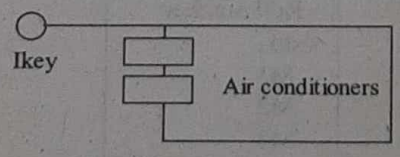


Figure: Interface/Implementation

In the above figure, name of the component is the "air conditioners" and interface is the "Ikey". The component is always represented by a large rectangle with two small rectangles pierced into it. Interface is denoted by a circle which is indicated by including letter "*I*" in the beginning of its name.

Q56. What is meant by tagged values and constraints? Discuss when they can be used and also give suitable examples to show their usage.

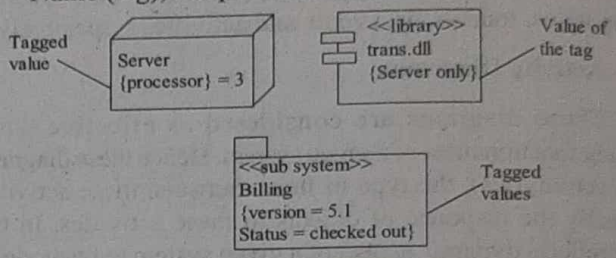
Answer :

Tagged Values

Stereotype can be used to add new things but with tagged values new properties can be added. For example, when a project is released, it is important to keep track of version numbers. Here tagged values can be used to add this information to the models. The tag value applies to the element itself but not to its instances.

Tagged values are used to specify properties related to the coding. It is represented as a string enclosed in brackets and placed below the name of other element. The string syntax is,

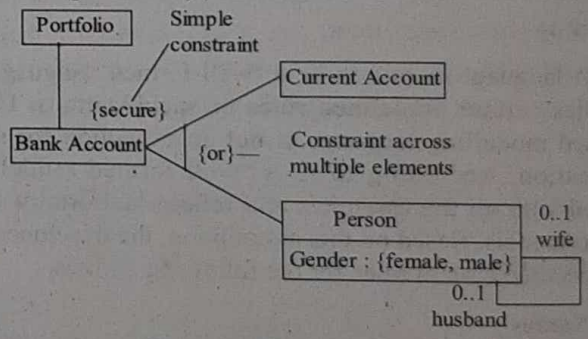
Name (tag), a separator (=), value



Figure

Constraints

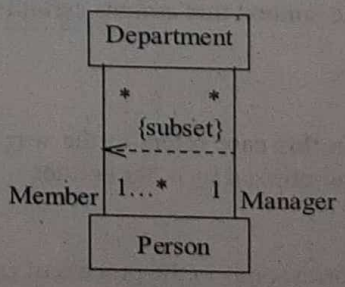
This is an extension of semantics of a UML element which allows the addition of new rules or modification of existing areas. For a system model to be well formed, the conditions specified by the constraints should hold true.



Figure

When more precise specification of semantics are needed, the UML's Object Constraint Language (OCL) is used.

A constraint is represented as a string enclosed in brackets and placed near the association.



Figure

3.3.1 Basic Structural Modeling: Class Diagrams

Q57. What is class diagram? What are the common properties and uses of class diagram?

Answer : Model Paper-III, Q6(b)

Class Diagram

A class diagram is a graphical representation of the static view of the system. It shows a collection of static model elements such as classes, types, their contents and relationships in addition to the interfaces and collaborations. Class diagrams are significant for constructing executable system. Apart from this, class diagrams are used for visualizing, specifying and documenting the structural modeling.

Common Properties of Class Diagram

The major properties of class diagram that distinguishes it from other diagrams are its contents.

Contents of Class Diagram

The class diagrams consist of four things. They are,

(a) **Classes**

Classes are the collection of objects of similar type. They play a major role in class diagrams.

(b) **Interfaces**

Interfaces are collection of operations that specify a service to a class. They focus on the externally visible behavior of an element. This behavior may be complete or only a part of it. Therefore, interface only specifies the operation, but does not implement them.

(c) **Collaborations**

(i) Collaborations defines a unit where different elements work together to provide a cooperative behavior.

(ii) The collaborations are the most essential things to make up the system.

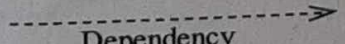
(iii) During implementation, collaborations can have structural as well as behavioral semantics.

(d) **Relationships**

Relationship defines the relation existing between multiple things. Basically, there are three types of relationships commonly used. They are,

(i) Dependency

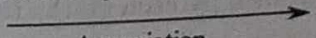
It is a type of relationship between the two entities where, a change caused to one entity may effect the other entity. Dependency is generally represented by a dashed line as shown below,



Dependency

(ii) Association

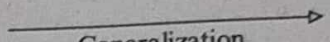
It represents a structural relationship, connecting different objects. This relationship is generally represented by a solid line.



Association

(iii) Generalization

Generalization is termed as a "specialized relationship". Here, object of one entity can be substituted with the objects of another entity. The entity whose objects are substituted is known as a parent entity and the entity which is providing objects for replacement is known as child entity. This relationship is generally represented by a solid line with a arrow head.



Generalization

Uses of Class Diagrams

Following are the uses of class diagram,

1. It plays a significant role in modeling complete details of a system.
2. It models different collaborations. These collaborations define a unit, where different elements work together to provide a cooperative behavior.
3. It also provides several mechanisms to model logical database. Here, database can be referred to as a place where large amount of data can be stored. Modeling such database requires many complex properties. These are provided by the class diagrams.

Q58. State and explain the common modeling techniques of class diagram.

Answer :

The common modeling techniques of class diagrams are,

1. Modeling Simple Collaborations

To understand and design a system, initially identify all the terms associated with it. This helps in identifying all the classes involved within the system. All these classes can later be related or collaborated with each other.

The various steps involved in modeling a class diagram are given below,

(i) Initially, make a clear idea about the mechanism to be modeled. Here, mechanism signifies the behavior or functionality of a module of the system, which is an outcome of interacting classes or interfaces.

(ii) Once done with the first step, identify its relative classes, interfaces, collaborations and their relationships.

(iii) Traverse through the model, keeping in view the scenario being modeled. This helps in locating missing instances and correcting those instances which were incorrect.

(iv) Finally, write the contents for these elements and operations of classes wherever necessary.

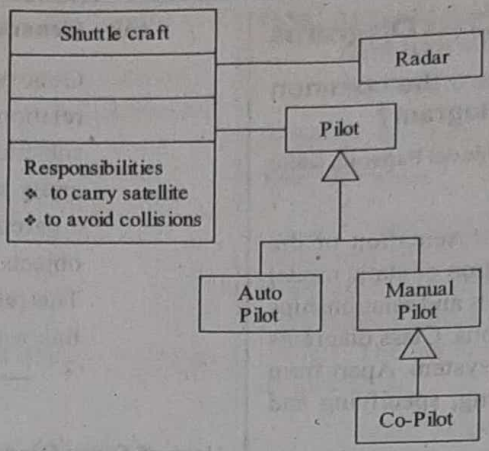


Figure: Modeling Simple Collaborations

In the above figure, the mechanism of a shuttle aircraft is modeled. The mechanism can be first understood by identifying the various things involved in it i.e., shuttle craft, radar, pilot and its types. Later, it can be shown how they collaborate through associations and generalization relationship.

2. Modeling Logical Database Schema

UML is a language, used for modeling different kinds of systems. The main features of UML will be demonstrated especially when used for modeling an Object-Oriented or relational or hybrid database. These systems possess many profound objects, which can be stored/retrieved depending on the requirements.

The following steps are essential in modeling logical database schema,

(i) Select the classes from the existing model. The selection should be made based on a condition that this position should exists till the lifetime of their application.

(ii) Evaluate a class diagram and insert the classes (which was selected in step (i)) in it if required. Also mark these classes using tagged values.

(iii) Make these classes more elucidating by enlarging their representations.

(iv) Reduce redundancy by applying intermediate abstractions. Abstraction is an essential tool for simplifying complexity.

(v) To make these classes look more elucidating, there is a need to stress on their behavior. This can be achieved by elaborating various operations that reflect data integration and data access.

(vi) Based on the requirements, apply tools which revert logical to physical design.

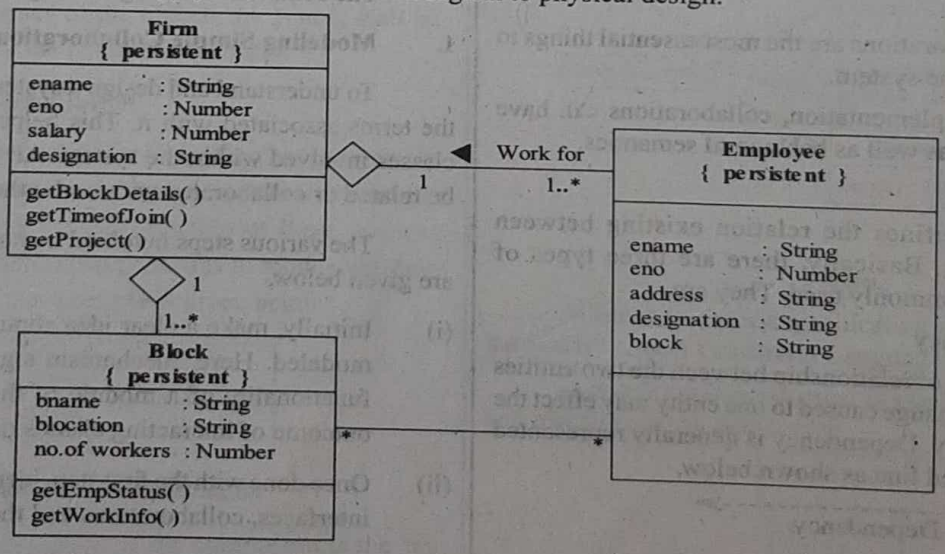


Figure: Example Class Diagram

The above example describes the information that is necessary to be maintained by each firm. The record such as eno, ename, salary and designation contributes the employee's record. The getBlockDetails( ), getTimeofJoin( ), getProject( ) contributes the information contained in the block, the employee's joining and project details handled by each block. The similar type of information is maintained by each class.

Q59. With a suitable example explain how to design a class. Give all possible representation in a class (name, attribute, visibility, method, responsibilities).

Answer :

A class is an abstraction of the common properties from a set containing similar objects. The common properties are attributes, operations, semantics and relationships. It basically captures the terminology or vocabulary of a system. Consider the example of a car, which contains various parts like steering, engine, wheels, body, etc. Each of these parts possess a set of properties. All these things are assembled form the car, therefore these things are related to each other.

A class in UML can be modeled by using the following representations,

(i) Name

Name refers to the name of the class. It must be unique so as to distinguish itself from other classes. The class name is usually a string. A class can be represented by using a class name.

(ii) Attribute

Attributes are data values or range of data values that the objects will hold. The number of attributes in a class may vary at any instance of time. Every attribute will have some value for an object of a class.

An attribute is a named property of a specified type in a class which has values for every instance. Apart from name, an attribute can be specified by showing its visibility, type, initial value etc.

(iii) Visibility

Visibility declares the ability of an element to view or reference another element that is in a different name-space from the referencing element. It can be specified to the classifier attributes and operations. It specifies whether it can be used by other classifiers or not. The various levels of visibility are as follows,

1. **Public**

   This data is generally accessible by other classifiers. It is denoted by a plus (+) symbol.

2. **Private**

   This data is accessible only within the classifier itself. It is denoted by a minus (–) symbol.

3. **Protected**

   This data is accessible only by the classifier and its descendents. It is denoted by a hash (#) symbol.

Example

The various levels of visibility are shown in the figure below,

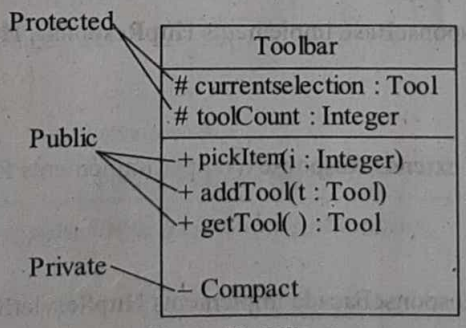


Figure: Visibility

The visibility marker is placed on list elements, such as attributes and operations in a class. It may be suppressed. The absence of visibility marker indicates public visibility.

(iv) Operation/Method

An operation/method is a specification of a service that is called by an object inorder to execute an object class. It also specifies the behavior of an object class. A class may or may not have any operations. The operations can be name, parameters, types, or initial values.

(v) Responsibilities

A responsibility defines the roles performed out by a class using its attributes and operations. It is an obligation or a contract of a class or other element. Responsibility can be a free text written as a phrase, sentence or a short paragraph.

Example

Following example shows a class and all its representations (i.e., name, attribute, visibility, method, responsibility)

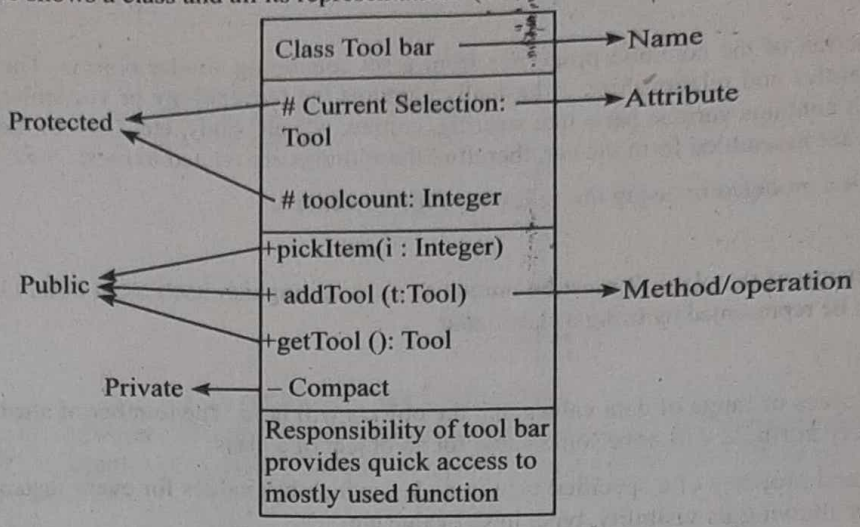


Figure: Class with its Representations

Q60. Enumerate the steps involved in forward engineering and reverse engineering of a class diagram.

Answer :

Forward Engineering

Forward engineering involves converting the models into software code. This process is usually done by using the concept called mapping. However, it is not completely possible to map models into code. This is because, their is a possibility to lose some authentic information during the conversion or mapping. Therefore, it becomes necessary to choose a language that has a higher degree of compatibility so as to avoid loss of information. The steps involved in forward engineering are as follows,

(i) Obtain a clear idea about the rules involved in mapping.

(ii) With respect to the semantics and constraints of a language, choose compatible features of UML.

(iii) Be specific about the language being chosen. This can be done by specifying its name using tagged values.

(iv) Use software tools like Rational Rose through which all the possible models can be created, and if these models are semantically correct, the application auto generates the code in languages like Java and C++.

Example

```
class HttpResponseBase extends ResponseBase implements HttpResponse, HttpServletResponse
{
}
abstract class HttpResponseWrapper extends ResponseWrapper implements HttpResponse
{
}
class HttpResponseFacade extends ResponseFacade implements HttpServletResponse
{
}
abstract class ResponseWrapper implements Response
{
}
abstract interface HttpResponse extends Response
{
}
abstract class ResponseBase implements Response, ServletResponse
```

```
{
}
abstract interface HttpServletResponse
{
}
class ResponseFacade implements ServletResponse
{
}
abstract interface ServletResponse
{
}
abstract interface Response
{
}
```

Reverse Engineering

Reverse engineering involves converting the developed code back to the models. This process makes use of proper mapping procedures, however it results in information loss. Therefore, one has to be very careful during the mapping procedures and should use the following steps for reverse engineering,

1. Clear understanding about the rules is must inorder to perform the mapping.
2. In reverse engineering any mapping tool can be used to convert a code back to its respective model.
3. Class diagrams will not be directly generated. Instead they are created by querying the model. The class diagram can be further elaborated by considering the various relationships among the classes.

Example

The following figure shows is the UML model.

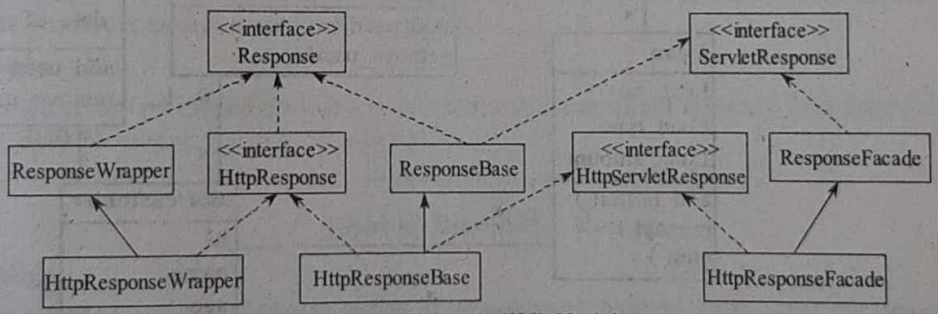


Figure: Example UML Model

Q61. Draw and explain the class diagram for a banking application.

Answer :

Class Diagram

A class diagram depicts the static view of the system. It describes the classes, interfaces, collaborations and relationships between them.

Banking System

A banking system allows its customers to open bank accounts, take loans and deposit/withdraw their money. For this, initially a customer has to register and open their account by providing their personal details. After opening the account the customer can perform the following tasks,

1. Deposit money
2. Withdraw money
3. Take loans
4. Change branch
5. Modify personal details.

Functional Requirement

Following are the functional requirements of the class diagram,

- **Purpose**

To open a new account for the customer.

- **Input**

Personal details like name, age, address and phone number.

- **Output**

Success: Account opened successfully

Failure: Account cannot be opened due to wrong details provided.

Class Diagram for a Banking System

The diagrammatic representation of the class diagram for a banking system is as follows,

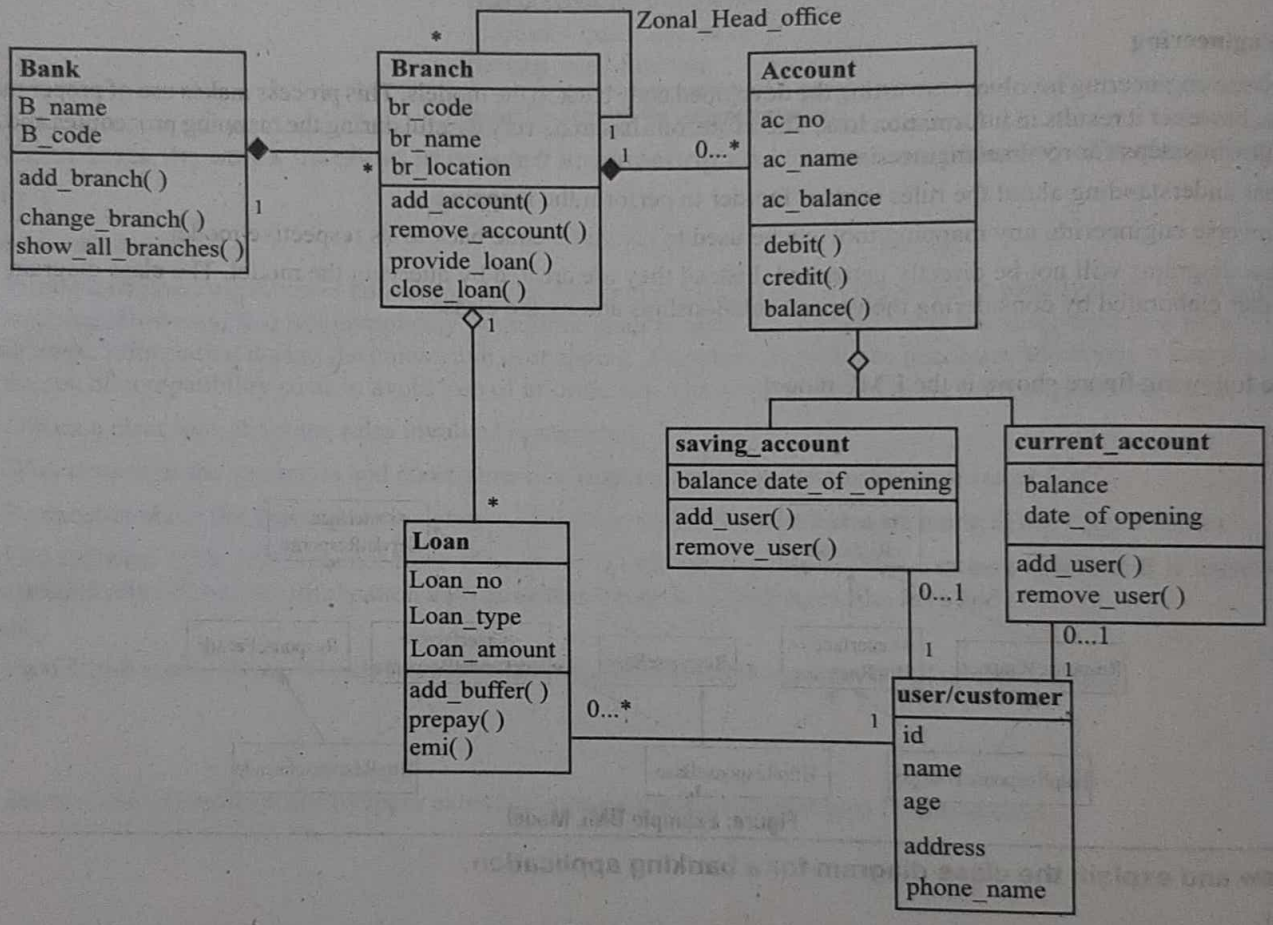


Figure: Class Diagram System

The class diagram for a banking system contains seven classes i.e., Bank, Branch, Account, Loan, Saving-account, current-account and user/customer. A bank may have many branches. Each branch has as zonal head office that manages all the branches present under it. A customer can open either savings account or current account or both. At any instant, the customer is not allowed to have more than one savings or current account. A customer can avail loan from the bank by providing the necessary details the relationships between classes are as follows,

1. A bank has many branches [composition, one-to-many]
2. A branch has "role" zonal_head_office that manages other branches [unary association, one-to- many]
3. A branch has many accounts [aggregation, one-to-many]
4. A customer may have one current/savings account [association, one-to-one]
5. A branch can provide many loans [aggregation, one-to-many]
6. A customer can avail many loans [association, one-to-many].

3.3.2 Sequence Diagrams, Collaboration Diagrams

Q62. Explain with an example Interaction diagram. Model Paper-I, Q7(a)

OR

What do you mean by interaction diagrams? Explain them with a suitable example.

Answer :

Interaction Diagrams

Interaction diagrams shows the interactions between different objects. These interactions consists of transmission of messages between objects and their relationship. Sequence diagrams and collaboration diagrams comes under the category of interaction diagrams.

Properties of Interaction Diagrams

The common properties of interaction diagrams are name and graphical content.

Contents of Interaction Diagrams

The contents of interaction diagrams plays a major role in defining various properties of it (interaction diagram). The basic contents of interaction diagram are as follows,

(i) **Objects**

Objects represents any real world entities which can be distinguished from one another. They are encapsulated in the class. There can be any number of objects in a class communicating with each other.

(ii) **Links**

A link is said to be a semantic connection between two objects. Link plays a major role in sequence diagrams by connecting different objects.

(iii) **Messages**

Messages are said to be the information carriers between the objects. They give the exact information about what type of service is offered by one object to another object, or what services an object render from another objects. Messages are of two types, i.e., synchronous and asynchronous message.

Synchronous Messages

The messages which are processed each at a time making the transmitter object to wait until the process finishes its task are termed as synchronous messages. It can be represented as,

Figure (a): Synchronous Message

Asynchronous Messages

Asynchronous messages are quite opposite to that of synchronous message. In this case, the transmitter object is facilitated to send messages without waiting for the receiving object to complete it's processing. Hence in asynchronous messages, more than one message can be processed at a given instance. It can be represented as,

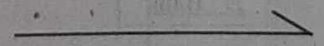


Figure (b): Asynchronous Messages

Uses of Interaction Diagrams

Following are the uses of interaction diagrams,

(i) The main use of interaction diagram is that, it can be successfully applied in building the dynamic aspects of any system. This ability of interaction diagrams causes it to be used in building a model involving interaction of an entity at a given instance.

(ii) Interaction diagrams can be used to model several flow of control with respect to time ordering and organization respectively. For doing so, sequence and collaboration diagrams are used as they are internal entities of interaction diagrams.

Types of Interaction Diagrams

Interaction diagrams are of two types, they are,

1. Sequence diagram
2. Collaboration diagram.

1. Sequence Diagram

Sequence diagram reflects the dynamic view of a system. It is a type of interaction diagram that shows the sequence of transmission of messages. This sequence is time dependent. Sequence diagrams consists of objects and interactions between them. These objects can be instance of classes or other things. A sequence diagram is constructed by placing the participating objects on top of the diagram along the X-axis, and then the messages which are sent and received by objects are placed along Y-axis according to the time order.

2. Collaboration Diagram

Collaboration diagrams organizes various objects participating in sequence interaction. These objects can interact with each other using various messages. Thus, the collaboration diagram consists of objects, links and messages, where the names of the objects are instance of classes. Sometimes, these names of an objects also refers to the instance of components, nodes to which they are related.

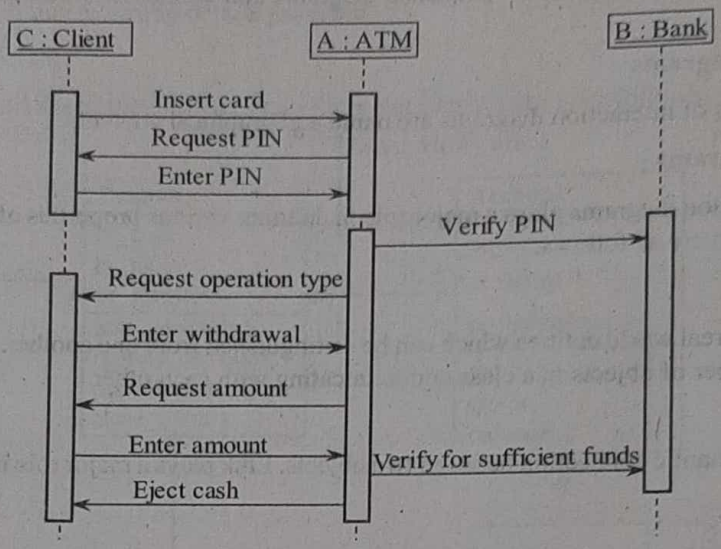


Figure (c): Sequence Diagram for 'Withdraw Amount' Use Case

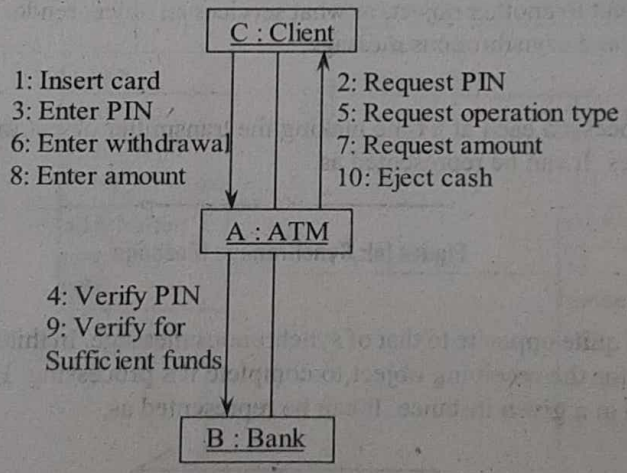


Figure (d): Collaboration Diagram for 'Withdraw Amount' Use Case

Explanation

The above figures shows the interaction diagrams (i.e., sequence and collaboration diagrams) of the use case "withdraw amount". Both sequence and collaboration diagrams has three objects i.e., 'Client', 'ATM' and 'Bank'.

The sequence diagram shows the sequence of actions (with respect to time) that occurs when a 'client' withdraws money from the ATM. It requires a 'client' to insert ATM card into 'ATM' machine, after which the 'ATM' asks 'client' to enter PIN. Once client enters the PIN, ATM sends a request to 'Bank', which verifies the PIN. If PIN verification is successful, ATM requests 'client' to specify what action it wants to perform. The client specifies that it wants to perform withdraw operation. Then the ATM asks client to enter the amount, after that client specifies the amount. Now, ATM again sends a request to 'Bank' that verifies whether sufficient funds are available in client's account or not. Finally, 'ATM' machine ejects out the cash.

The collaboration diagram also shows the same information as that of sequence diagram except that it does not depict the flow or sequence with respect to time, rather it shows various messages each object can send and receive.

Q353. What are the common modeling techniques of interaction diagrams?

Answer :

The common modeling techniques of interaction diagrams are,

1. **Modeling Flows of Control by Time Ordering**

The various steps involved to model flow of control by time ordering are as follows,

(i) Set the perspective or scope through which it is possible to visualize the system or its parts i.e., determine whether the context is a system, subsystem, class or operation.

(ii) Identify the participants of the system (i.e., objects). These objects are then placed in the sequence diagram from left to right at the top of the diagram.

(iii) Draw the lifeline or lifetime of objects. Each object has a time in which it is active, that time period is its life time, which has to be created. The activation and termination of the objects is indicated by the corresponding stereotyped messages.

(iv) Outline the message between the lifelines from top to bottom. As the sequence of messages proceeds vertically down wards, the messages are depicted along with semantic properties.

(v) Whenever message loops are required or the object is continuously active we can use "focus of control" to indicate it.

(vi) Time stamping can be done if it is necessary to show timing constraints, by using timing marks.

(vii) Post-conditions and preconditions of each message can be shown if required.

For example, consider the bank transaction of withdrawing money, since, the scope is already identified. The next step is to consider the objects involved in the transaction i.e., the customer, bank clerk and the teller. The bank clerk creates a transaction using stereotype <<create>>. The respective active state is depicted on the life time through vertical rectangles called focus of control. The figure shows the modelling of flows of control by time ordering.

Withdrawing Money Transaction

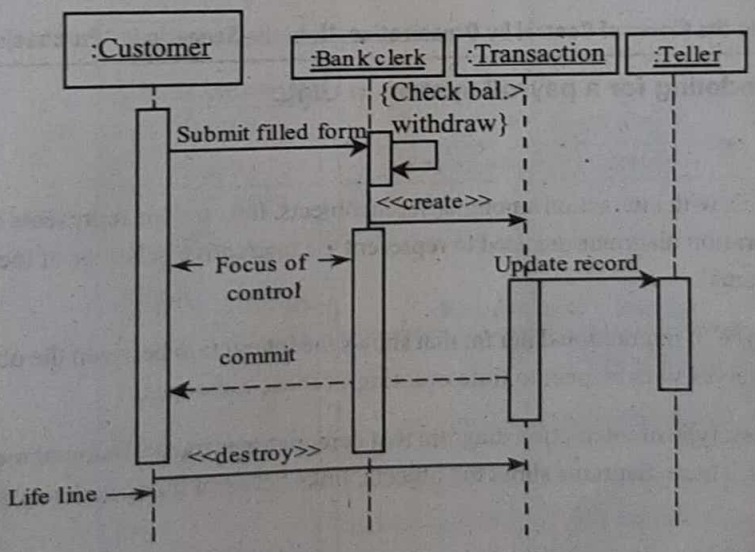


Figure: Modelling of Flows of Control by Time Ordering

2. **Modeling Flow of Control by Organization**

The various steps involved to model flow of control by organization are as follows,

(i) Determine the scenario where interaction takes place (E.g. It can be system on the whole or part of the system etc.)

(ii) Determine the objects participating in the interaction. Place the most essential object at the centre of the diagram surrounded by rest of the objects.

(iii) Assign initial properties to each of these objects. It may happen that the values, stereotypes, states or roles of the objects may change significantly as the interaction proceeds. To narrate this consequence, use virtual objects and assign new properties to it (as per the requirement). Also, assign messages appended with suitable sequence numbers, and stereotypes "copy" and "become" can be used over new messages.

(iv) Assign links to these objects such that the association links are added first followed by other links with respective to the order of stereotypes. Here "global" and "local" stereotypes can be used.

(v) Traverse to the initiating link (where the current interaction get triggered). Add appropriate message over this link along with suitable sequence number. It can happen that there may exists nesting of links. To manage such occasion use "Dewey decimal" numbering methodology.

(vi) If required, add time and scope over these messages, assign them accordingly.

(vii) If required, add pre-conditions and post-conditions to these messages to increase its granularity.

Following figure depicting the modelling flows of control by organization. Here the scenario is "purchasing of bus ticket".

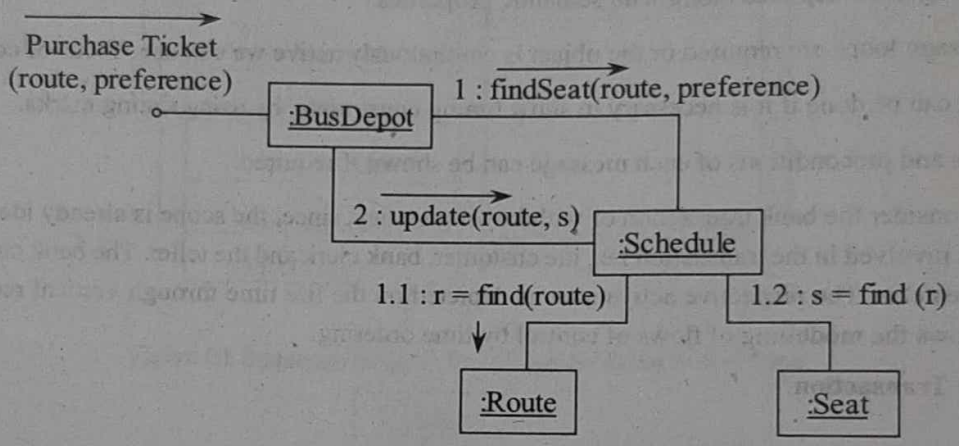


Figure: Modelling the Flows of Control by Organization, Here the Scenario is "Purchasing of Bus Ticket"

Q64. Apply interactive modeling for a payroll system in UML.

Answer :

Interactive modeling deals with interaction among different objects. Interactions represents dynamic behavior of the system. In UML, sequence and collaboration diagrams are used to represent the interactive behavior of the system. Hence, these diagrams are known as interaction diagrams.

Sequence diagram is a type of interaction diagram that shows the interaction between the objects. The objects in a sequence diagram, interact among themselves with respect to time ordering of those messages.

Collaboration diagram is a type of interaction diagram that demonstrates transmission of messages between various objects participating in the interactions. These diagrams show the objects, links between them, and message transmission between them.

Automated Payroll System

Automated payroll system is an electronic activity of maintaining information of the employees of an organization so as to generate the payroll automatically. It is an interface between the employee and the administrator who generates the pay slips.

It allows an employee to check their time record, number of hours worked by them on particular day, leaves taken by them, and also the time and days pent by individual on a single project. Employee can also access and edit their own details. The system runs on every individual's desktop working in an organization. It enables the administrator to generate performance report of the employee, add and delete new employees etc. It aims at providing efficiency in pay slip generation process and eliminates the difficulties which were involved in the manual pay/salary calculation and maintenance of employee details.

Sequence Diagram for Automated Payroll System

Employee and administrator are the two actors involved in the payroll system. They interact with the user interface and database manager. The sequence diagram for an employee and an administrator are as follows,

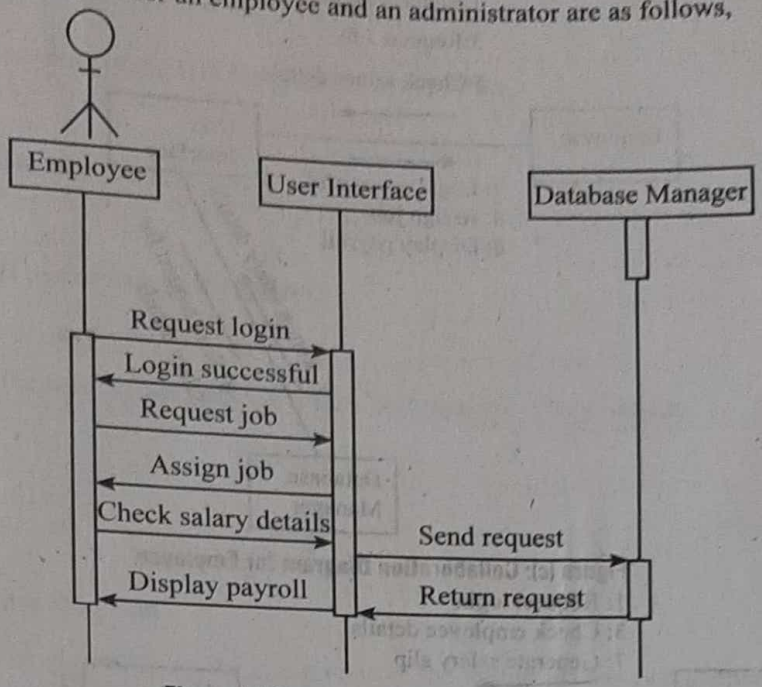


Figure (a): Sequence Diagram for Employee

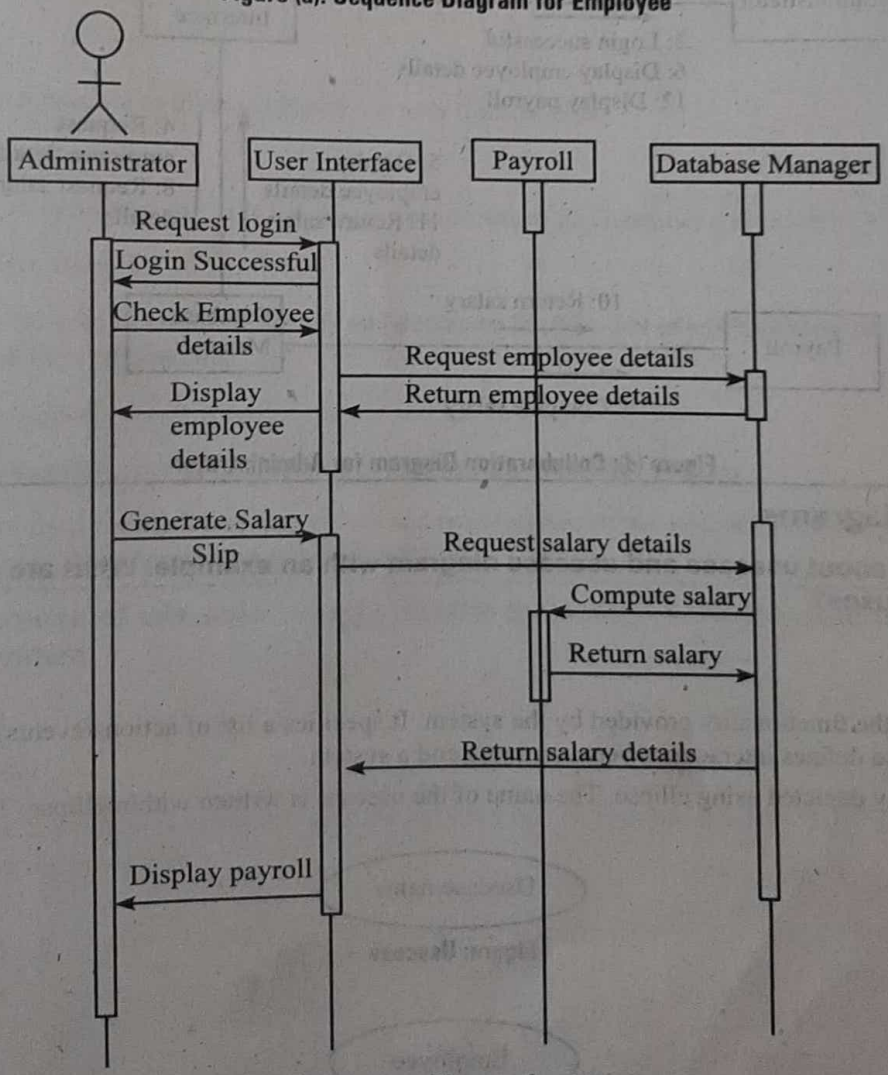


Figure (b): Sequence Diagram for Administrator

Collaboration Diagram

The collaboration diagram for an employee and an administrator are as follows,

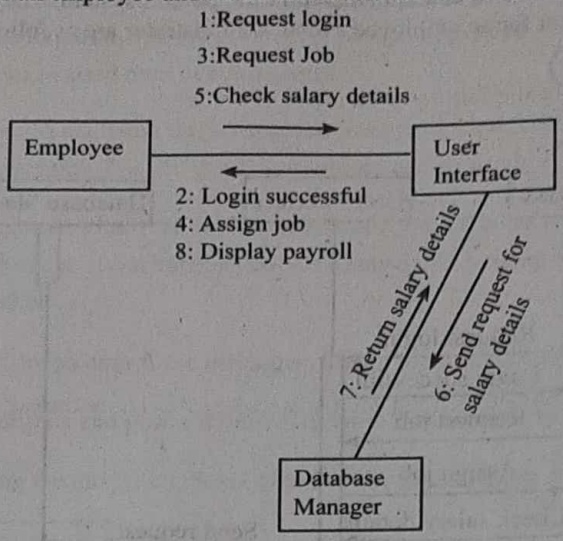


Figure (c): Collaboration Diagram for Employee

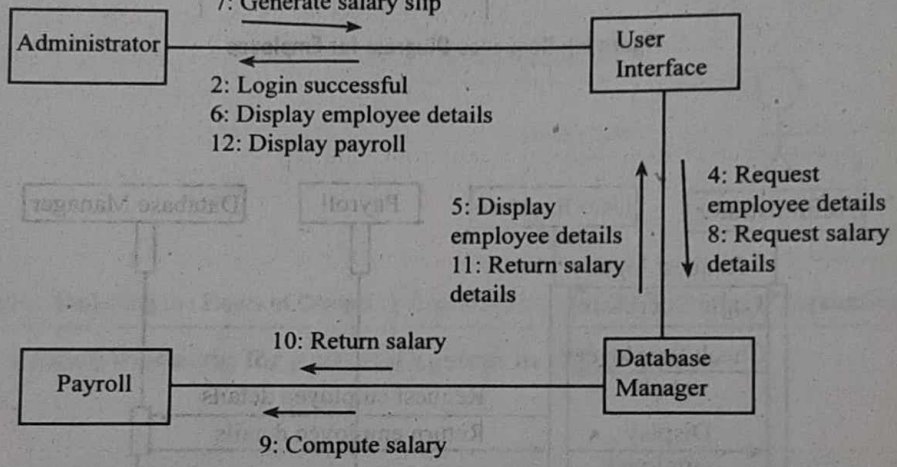


Figure (d): Collaboration Diagram for Administrator

3.3.3 Use Case Diagrams

Q65. Explain in brief about usecase and usecase diagram with an example. What are its contents, common properties and uses?

Answer :

Usecase

A usecase defines the functionality provided by the system. It specifies a list of actions/events performed by the system. Besides this, a usecase also defines interactions between a role and a system.

A usecase is usually depicted using ellipse. The name of the usecase is written within ellipse. This is shown in the figure given below,

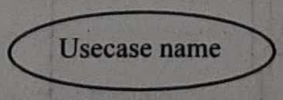
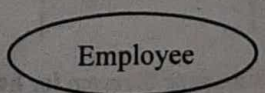


Figure: Usecase

Example



Usecase Diagram

For answer refer Unit-III, Q53, Topic: Use Case Diagrams.

Contents of Usecase Diagram

The use case diagrams consists of the following elements. They are,

(i) **Actors**

These represent the users, organizations, etc.

(ii) **Usecase**

It gives the list of events performed by the system.

(iii) **Relationship**

These helps to specify the relation between the actors, usecases, etc. These include,

(a) Association

(b) Dependency

(c) Generalization.

Common Properties of Usecase Diagram

Usecase diagrams are used to address the static view of the system. It has two properties similar to other diagrams. They are,

(i) **Name**

It is defined within a usecase to distinguish one use case from another.

(ii) **Graphical Content**

It includes actors, relationships etc. It is the unique feature which distinguishes it from other UML diagrams.

Common Uses of Use Case Diagram

Usecase diagrams are used to visualize, specify and document the behavior of an element by presenting the classes, actors, systems realistically in the form of diagrams.

(i) Usecase diagrams supports forward engineering to test the executable systems.

(ii) Usecase diagrams supports reverse engineering to analyze the executable systems.

(iii) These diagrams are used for modeling the context and requirements of the system by allowing actors to interact with the system, but not giving them permission to enter inside the system.

Q66. What is the purpose of use case model? Identify the actors, scenarios and use cases for a library management system.

Answer : Model Paper-II, Q7(a)

Purpose of Usecase Model

The purpose of usecase model is to depict "what the system does". It does not describe "how" the system does a particular task. It includes the following components,

1. Actor
2. Scenario
3. Usecase.

For remaining answer refer Unit-I, Q67, Topics: Actor, Scenario, Usecase.

Library Management system

The usecase diagram for library management system is as follows,

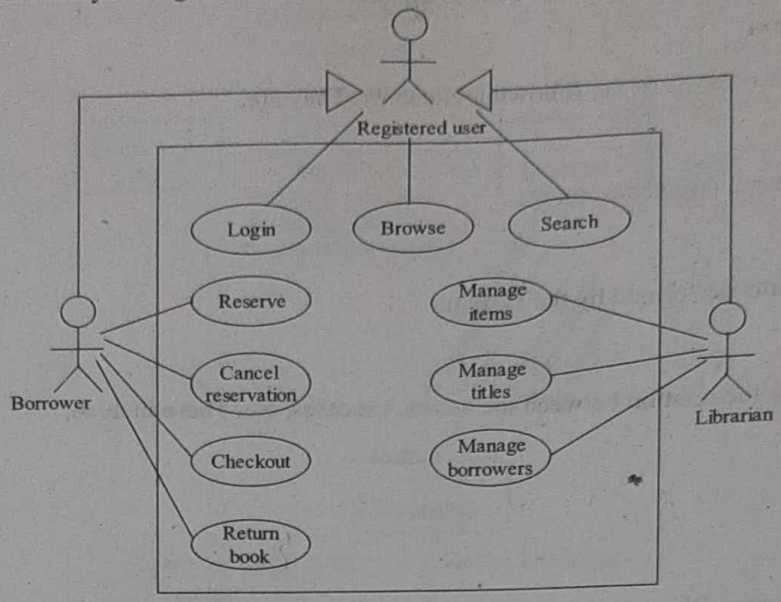


Figure: Usecase Diagram for Library System

Actors in Library Management System

Library management system consists of the following users,

1. Librarian
2. Borrower
3. Registered users.

Scenario in Library Management System

The flow of transactions between the usecases and actors describe a set of sequence known as scenario.

Usecases in Library Management System

The flow of transaction between the usecases and actors describe a set of sequence known as scenario.

	Usecases	Description
1.	Login	This usecase allows the actors to login the system using a unique registered ID.
2.	Browse	This usecase allows the members to access the database so as to know the availability of the required item.
3.	Search	This usecase allows the members to search items based on the title, author name.
4.	Reserve	This usecase allows the borrower to reserve a book if he/she intends to do so incase the required book is unavailable.
5.	Cancel reservation	The reservation made by any of the borrower can be cancelled after borrowing the required book.
6.	Checkout	This usecase allows the borrower to checkout almost four items at a time.
7.	Return	This use case reflects the activity "returning of book" which was previously borrowed by any of the borrower.
8.	Manage title	This usecase allows the librarian to manage the books based on book's title or author's title.
9.	Manage items	This usecase allows the librarian to either add or delete items from database.
10.	Manage borrowers	This usecase allows the librarian to verify the transactions of the borrowers.

Q67. Identify the actor, scenario and usecase with an example.

Answer :

Actor

An actor reflects a person, an organization or an external system which actively participates in the system. These actors interacts with the system by transferring messages.

In usecase diagrams, actors are represented using stick figures, with the role of the actor included at it's bottom, following are examples of actors,

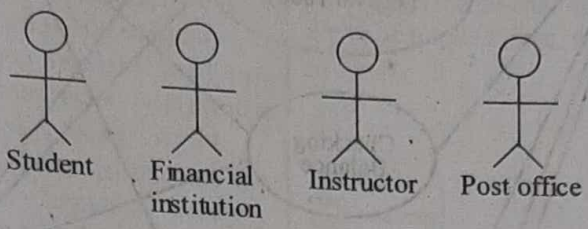


Figure (a): Examples of Actors

Actors can be of two types i.e., primary actors as well as secondary actors. Actors which reflects main system actions are primary actors, and actors performing secondary functions (say database management, communication among systems, etc) are secondary actors.

Apart from above classification, actors can also be termed as active and passive. Active actors trigger a usecase, whereas passive actors remains involved with the usecases.

Scenario

A scenario is a specific sequence of actions which describes the behavior of system. Scenarios are related to usecases in the same way as instances are related to classes. It is an instance of usecase.

For example, consider a withdraw money usecase of an ATM system.

They are many variations such as,

(i) Withdraw money from another account

(ii) Transfer money from one account to another account.

(iii) Deposit money to another account.

The "withdraw money" usecase describes a set of sequences, where each sequence corresponds to the flow via all the above variations and each sequence is known as scenario.

Usecase

A usecase reflects certain sequence of actions that occurs within a given system. These usecases can also provide their functionality to a given system.

Usecases are represented in a horizontal ellipse (in UML notation) with it's respective action embedded in it.

Example

Following are possible usecases where a "patient" interacts with a "nursing home".

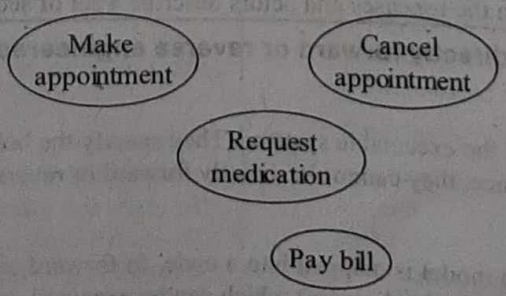


Figure (b): Example Use Cases

Actor, scenario and usecase can be inferred from the following diagram as,

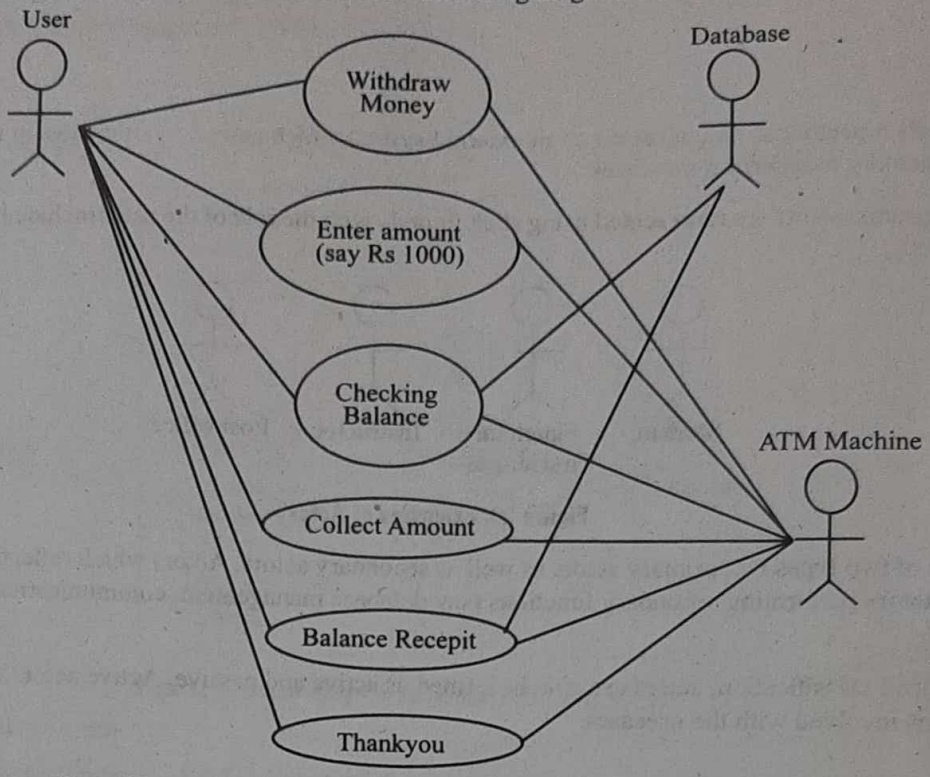


Figure (c): General Flow of Transaction between the User and ATM Machine

Actors identified from figure (c) are,

(i) User

(ii) Database and

(iii) ATM Machine

Usecase

Following are the list of usecases identified from the figure (c),

(i) Withdraw money

(ii) Enter amount

(iii) Checking balance

(iv) Collect amount

(v) Balance receipt

(vi) Thankyou.

Scenario

The flow of transactions between the usecases and actors describe a set of sequence known as scenario.

Q68. Can a usecase diagram be directly forward or reverse engineered? Justify your answer.

Answer :

Usecase diagram of UML reflect the executable systems. They specify the behavior of the element rather than specifying the behavior of the implementation. Hence, they cannot be directly forward or reverse engineered.

Forward Engineering

It is a process in which a desired model is mapped into a code. In forward engineering, the usecase diagram will reveal a test condition (for every usecase present in the diagram) which can be executed, when a new release of element's behavior is made.

Different steps required in forward engineering usecase diagrams are as follows,

- Start the process by initially considering the flow of events and exceptional flow of events associated with every usecase in a given diagram.
- In the next step, generate a test chronicle for each flow involved in the diagram. Consider the preconditions and post conditions of the flow. The preconditions can be taken for testing initial status whereas, the post conditions can be taken as the goal of test chronicle.
- Now, test the actors by generating a testable platform. Replace the original identities of these actors with respect to their roles commonly observed in the real world.
- Finally, using suitable artifacts or tools, execute the testable chronicles when a new element is released. Check to see whether the elements that are released satisfies the purpose of their generation.

Reverse Engineering

It is a process which produces a model by considering the code. Here an implementation language can be utilized. In reverse engineering the use case diagrams will not be fruitful, since there is a maximum probability of loosing certain kind of information. Here implementation is considered rather than it's behavioral aspects.

The essential steps required in reverse engineering the usecase diagrams are as follows,

- Initially, identify the actors present in a system.
- After all the actors has been collected, check for their roles. This includes their interactions, responses and indulgence with the state of the system in which they exists.
- Now, check for the flow of events associated with each actor, such that initial preference is given to primary flow followed by the exceptional flow.
- Accumulate the flows which are related to each other into one section. Assign a suitable use case for each section. While doing this, utilize the relationships such as "extend" and "include" respectively. Attach "include" relationship to those positions where there is a modeling of common flows. Similarly, attach "extend" while modeling the variant flows.
- Finally, consider all the actors and usecases and deploy them into proper usecase diagram. Use relationships to organize them.

3.3.4 Component Diagrams

Q69. Explain the concept of component diagrams in detail.

Answer : Model Paper-I, Q7(b)

Component Diagram

Component diagram describes the way in which component are interconnected with each other to form larger components or software systems. A component is a physically existing part (usually files) in a system. Executable files, .dlls, object files, etc., can be represented as, various components which can exist on a node. They are replaceable and supports openness and adaptability. The diagrams which incorporate these components along with their relationships to model the physical aspects of a system are called as component diagrams. The class diagrams show the logical aspects of a system therefore, component diagrams are used to decide which physical things would be available on a node like executable files, source codes, .dlls, tables etc. It is just a collection of vertices and arcs.

Component diagrams helps in deciding the way in which the source code can be organized (or) the way the database is to be laid (or) the manner in which the executable files of a project are to be implemented. A sample component diagram is shown in figure,

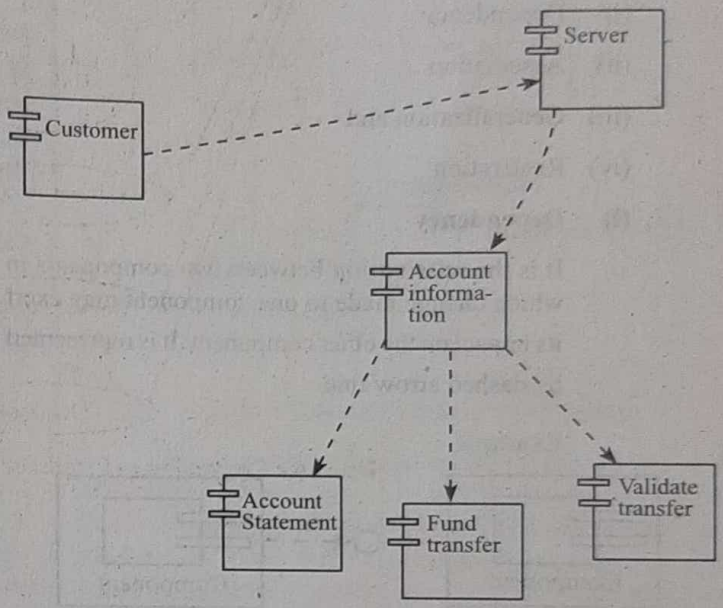


Figure: Component Diagram

Component diagrams are used when there is a need to decide,

(i) The way the source code can be organized.

(ii) The way the data base is to be laid.

(iii) The way the executable files of a project are to be implemented.

Contents of Component Diagram

Component diagrams include contents like,

1. Components
2. Interfaces
3. Relationships.

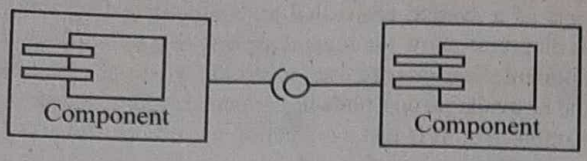
1. Components

Components refer to the part or element of the system to be modeled. This can be executables, libraries, tables, files, etc. Components also binds several classes, interfaces and collaborations by physical representation.

2. Interfaces

Interfaces consist of operations that belongs to classes or components residing in the models. It gives the external behavior of a component to which it is attached. The behavior can be complete or partial. Therefore, interface specifies set of activities, but it (interface) does not account for implementation of these activities.

Example



3. **Relationships**

Component diagrams support four types of relationships namely,

(i) Dependency

(ii) Association

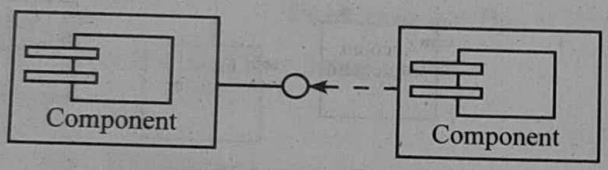
(iii) Generalization and

(iv) Realization

(i) Dependency

It is the relationship between two components in which change made to one component may exert its impact on the other component. It is represented by dashed arrow line.

Example



(ii) Association

It is the relationship between two components which binds them (components) to one particular location.

(iii) Generalization

It is the relationship in which objects of one component can be replaced with the objects of another component.

(iv) Realization

It is the relationship in which a component specifies the services which are carried out by another component.

Component diagrams may also include packages, according to the requirement of the systems. These packages are essential to club all the components to form one large component.

Common Properties of Component Diagram

The most common properties of component diagram (i.e., names and graphical content) are similar to the properties of other UML diagram. However, contents of component diagram is a unique property using which it is easy to differentiate a component diagram from various other UML supportive diagrams.

Common Uses of Component Diagram

Component diagrams are to be used when there is a need to divide a system into components inorder to describe the interrelationships existing through interfaces. These diagrams can also be used for dividing the components into lower-level structures.

Component diagrams are preferred when it is necessary to model the physical aspects of object-oriented systems.

The uses of component diagrams are as follows,

(i) They are extensively used in modeling the adaptable systems i.e., the system which relocates itself during failures. These systems also consists of some mobile agents which shifts its location depending on the situations i.e., shifting occurs whenever there is a need to balance the load or during crash recovery.

(ii) Component diagrams also serves best when used in modeling executable releases.

(iii) Modern way of storing the information on the systems is by using physical databases. These databases stores information using tables or pages. Thus, component diagrams provides good results even in modeling the physical databases.

(iv) Whenever a program is written using any programming language, the source code of the program is stored in the form of files. Component diagram models can be used to model this source code more effectively.

Component-level Design Elements

As the name suggests, component-level design provides all details of a given software component. In order to achieve this, the component design describes the representation of data structure for orderly placement of objects, suitable algorithms to promote processing internal to a given component and finally defines interface. The icon used to represent a component is given below.

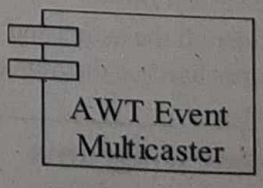


Figure: Component Icon

Name of the component is inscribed in the icon. Component along with interface is shown below :

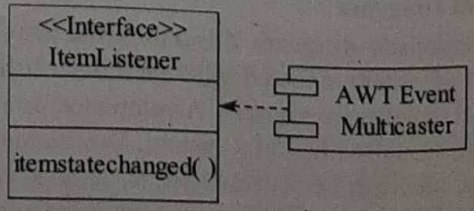


Figure: Component along with an Interface

Q70. Enumerate the steps to model source code. Illustrate with a diagram in UML notation.

Answer :

Source code refers to the code of a program, which is written on a computer. Before executing this code, it has to be saved in the file, which differs depending on the languages. For example, in Java programming language the source code is stored in .java files, in C++ the body of the source code is stored in .cpp files. When the applications of program increases, it becomes difficult to maintain and manage this source code.

Steps to Model Source Code

The steps required for modeling source code are as follows,

1. Initially, determine the set of source code files to be modelled. This is done using either forward or reverse engineering techniques. Once the appropriate files are identified model these files as components apply each of these component with a <<file>> stereotype.
2. If there are bulk files to be modelled, then arrange them in respective packages.
3. Use appropriate tagged values wherever necessary. The data corresponding to these tagged values can be the name of file, author name.
4. Use dependency relationship to model compilation dependencies between these.

Example

Consider an example of three source code files "file.h" is the header file, which is accessed by another file "file.cpp". This file inturn has a compilation dependency on another header file "irq.h".

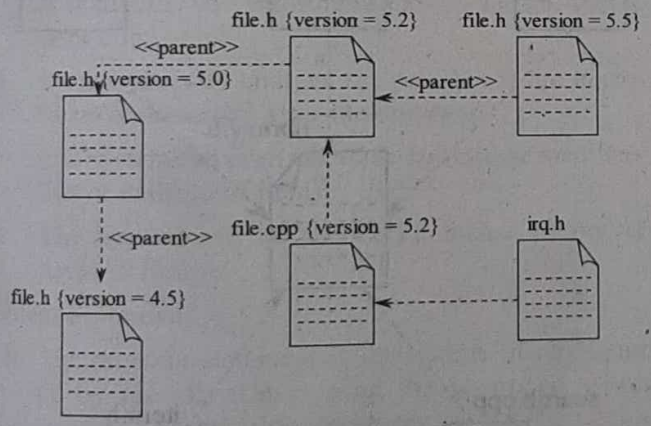


Figure: Modeling Source Code

Q71. Enumerate the steps to model an executable release. Illustrate with a UML diagram.

Answer :

Executable release indicates the storage and maintenance of executable files. These files can be extended by '.exe' or '*.dll*' files etc.

Steps to Model Executable Release

The steps required in modeling these files are as follows,

1. Initially, identify the components which are to be modeled. While doing this, recognize the set of components which are attached to the node or distributed around the node.
2. Apply stereotypes to each of these nodes. UML's extensibility mechanism can also be taken under consideration to visualize the stereotype representation.
3. Finally, relationships provide dependency between a component and its adjacent component.

Example

Consider an example for modeling executable release, representing the withdrawal transaction during ATM processing.

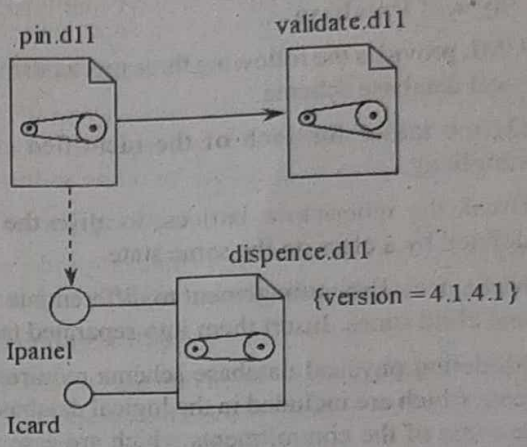


Figure: Representing Executable Release

The above figure represents the various aspects involved while withdrawing money using ATM. As seen from the figure, three components are used namely pin.*dll*, validate.*dll* and dispense.*dll*. There are two interfaces used namely Ipanel and Icard. Thus, these interfaces and components together form the part of transaction process.

Q72. Enumerate the steps to model the following,

(i) Adaptable systems

(ii) Physical database

(iii) Source code.

Answer :

(i) Adaptable Systems

Component diagrams are generally used to model the physical aspects of any object oriented systems. Here, the physical aspects refers to the servers or any storage devices or any databases, etc. Modeling these elements refer to the static view of the system. But the main quality of component diagrams are judged only when they are used to model the complex "Adaptable system", i.e., the systems which dynamically relocates itself to another systems, during the failure of the existing systems.

Steps to Model Adaptable Systems

The various steps provided by the UML to model adaptable systems are as follows,

1. Initially, identify those components which relocates itself between different nodes depending on the situation (modeling events).

2. Specify location of these nodes using location tagged value and implement them in the component diagram.
3. Create an interaction diagram to model the actions, which made the component migrate.
4. As relocation of the system had occurred, same instance with different tagged values can be used to represent the original location and relocated location in the component diagrams.

(ii) Physical Database

UML provides the following three mechanisms to model the physical database schema.

1. Define tables for each of the identified classes for simplicity.
2. Break the inheritance lattices, to align the instances defined by a class on the same state.
3. Make necessary arrangement to differentiate the parent and child states. Insert them into separated tables.

Modeling physical database schema requires mapping of elements which are included in the logical database schema. Here are some of the commitments which are essential while implementing them. They are,

1. Use SQL for implementing the operations like delete, read, update, create.
2. Apply stored procedures to attain best results in mapping, while dealing with complex behaviours.

Basing on the above statements, the following are the steps involved while modeling physical database.

1. Select the classes forming the constituents of logical database schema.
2. Apply different mechanisms in order to map the classes to their respective tables.
3. Create component diagram consisting of components, which are stereotyped as tables so as to visualize, construct, specify and document the mapping.
4. Translate the logical design into a physical design using appropriate tools.

(iii) Source Code

During the development process, certain decisions are made regarding the segmentation process of files. Based on these decisions, the source code files are created which store the information about logical elements like interfaces, classes and collaborations. This information is used in the creation of physical and binary components, which are generated from logical elements by using the development tools.

When a source code file is modelled, graphically then such modeling is used to,

(i) Expose the competition dependencies that exist among a set of source code files.

(ii) Manage the splitting and merging of a set of source files when the development paths are divided and combined using forks and joins, respectively.

Steps to Model a Source Code

The steps involved while modeling the source code are as follows,

1. The development tools impose certain constraints based on which the source code files that store the information about each logical element and its compilation dependency must be modeled.
2. If the modeled files are to be used in the configuration management and version control tools, then the tagged values like author, version and check in/check out information must be defined for every file involved in the management of source code configuration.
3. The relationships that exist among the source code files must be managed by the development tools. They can be represented and documented by using UML.

Example

Consider a model consisting of five header files (i.e., book.h, journal.h, item.h, article.h and library.h) that represented in the form of source code files so as to provide a specification of classes. An implementation file (i.e., search.cpp) is also included, which specifies the implementation mechanism of any one of the five header files.

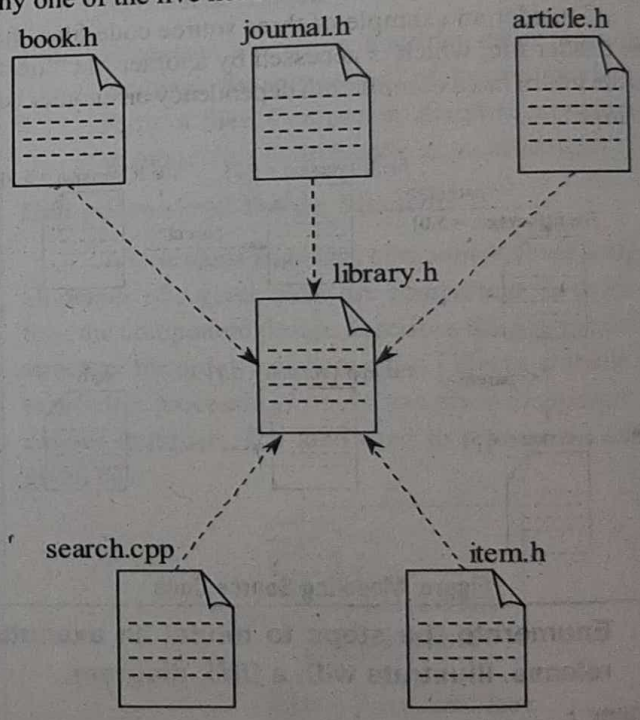


Figure: Source Code Modeling

If large number of source code files exist, then the files with similar concepts and semantics are clustered together in the form of groups. However, these groups are usually placed in different directories by the development tools and are modeled using packages. Typically, trace relationships are used for the visualization of the relationship that exists between a class and its source code file and also the relationship that exists between a source code file and its executable (or library) file.

Q73. Describe the problem of library management system using any two UML diagrams.

Answer : (Model Paper-II, Q7(b) | Dec.-19(R16), Q5(b))

A library is an inventory containing books, periodicals, journals, magazines etc. It is a very beneficial institution for the people who cannot afford to buy books. It lends its items to its registered users. Every registered user is issued a membership card which contains a unique-id. The library continuously purchases items and updates the inventories by adding new items or removing the damaged items from the database.

Existing System

The existing system is the manual library system wherein each work is carried out manually. The librarian maintains the personal details of members and the number of books borrowed by them. All this information is kept in file of record. Once a member returns the book, the information is deleted from the respective members's list.

Proposed System

The proposed system is unified library application that automates the functions of the library developed using Unified Modeling Language (UML). This computerized library system is effective, user-friendly and is capable of performing the following activities,

(i) The system maintains the information about the title item in its inventory along with their respective cost in an efficient manner.

(ii) It provides restricted access to the borrower and complete access to the librarian.

(iii) It maintains the information about all the borrowed item efficiently.

(iv) It automatically charges the member if the borrowed items are returned after the due date.

(v) The system can even calculate and charge members for lost or damage of items.

(vi) The system can be recovered incase of power or network failure.

Problem Statement

The problem statement is the outline of requirements prepared by the client assuming the proposed system's requirements. A system that automates the functionality of a library has to be developed.

A Library management system is a system modeled to manage and monitor the transactions (i.e., borrowing book, returning book) within a library. It consists of two registered users. A registered user is a person who registers into the library system by entering the details like name, age, etc. The person who is registered into the system is provided with a username and password using which he/she can login and perform tasks like searching for a book, borrowing a book, issuing a book, etc. A registered user can be a librarian or a borrower.

(a) **Librarian**

A librarian manages all the transactions of borrowers. He has complete access to the system database.

(b) **Borrower**

A borrower can be a student or a lecturer. He/she can borrow or return the items/books.

The computerized library management system is user friendly and is capable of performing the following activities,

(c) **Login to the System**

The library allows the registered users to login using their username and password.

(d) **Browsing for a Book**

The library system allows the borrower to search for a book by browsing the system using the title name or author name of the book or magazine.

(e) **Borrowing a Book**

The library system allows the book to be borrowed for a specified period of time.

(f) **Reserving a Book**

The library management system allows a registered user to reserve a book of his/her interest. It also allows the user to cancel the reservation if he/she is no longer in need of it.

(g) **Returning a Book**

The library system allows the user to return the book after or before the specified period of time. It issues a fine to the users if the user exceeds the due date of returning the book.

(h) **Maintaining the Information**

The library system allows the librarian to maintain the information about the books borrowed, reserved, returned, etc., in an efficient manner. The librarian also maintains the details of each person who has borrowed the book.

1. Use case Diagram

A use case diagram is the behavioral diagram used to model the behavior of the system. It contains actors, use cases and relationships between them. The use case diagram of a library management system involves two registered users as actors i.e., borrower and librarian. A borrower borrows or returns the books while the librarian performs all the transactions. It has the following use cases,

(a) **Login**

The librarian or borrower can login to the system for performing transactions like book searching, book issuing, etc.

(b) **Search Book/Item**

The borrower searches the books based on the name of book or author.

(c) **Issue Book/Item**

The librarian issues the book requested by the borrower.

(d) **Return Book/Item**

The borrower returns the book borrowed from the library.

(e) **Borrow Book/Item**

The borrower borrows the books/items with his/her interest.

(f) **Checkout**

The borrower checks out after selecting the books for borrowing.

(g) **Logout**

The borrower or librarian can logout from the system after performing transactions.

The following figure shows the usecase diagram for library management system,

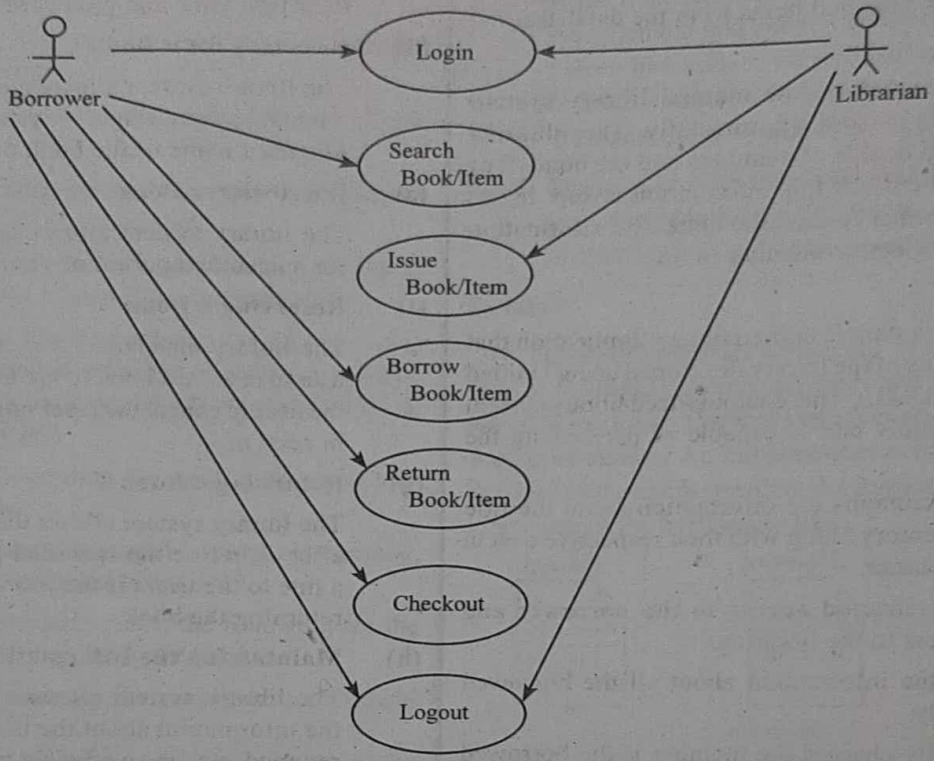


Figure (a): Use case Diagram for Library Management System

2. **Class Diagram**

A class diagram visualizes, specifies and documents the structural modeling. It consists of classes, interfaces and collaborations within the system. The class diagram of a library management system contains four classes — borrower, librarian, checkout and book/Item. The following figure shows the class diagram for library management system,

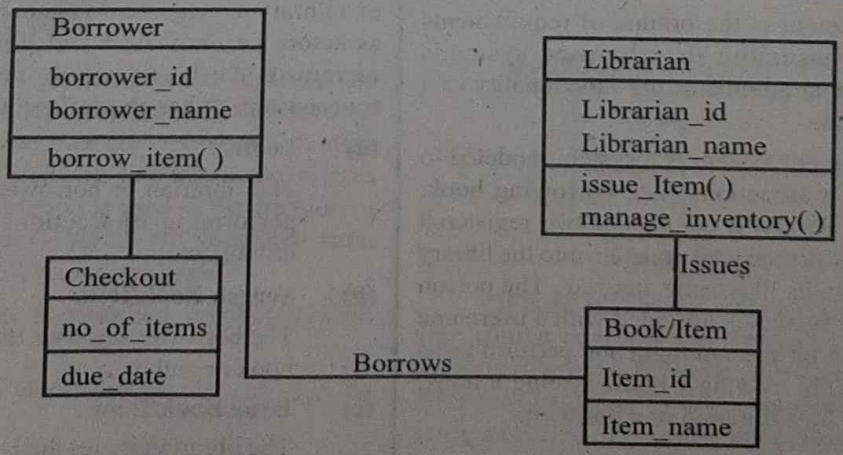


Figure (b): Class Diagram for Library Management System

UNIT 4

Testing Strategies, Product Metrics

Syllabus

Testing Strategies: A Strategic Approach to Software Testing, Test Strategies for Conventional Software, Black-box and White-box Testing, Validation Testing, System Testing, the Art of Debugging.

Product Metrics: Software Quality, Metrics for Analysis Model, Metrics for Design Model, Metrics for Source Code; Metrics for Testing, Metrics for Maintenance.

PART-A SHORT QUESTIONS WITH SOLUTIONS

Q1. What is meant by software testing? And list its characteristics.

Answer : Model Paper-I, Q1(g)

Software Testing

Software Testing is a method of executing a given software is executed with an intention of detecting bugs. It is one of the essential steps in software development life cycle. Hence, it is the responsibility of the programmer to build an error free software.

Characteristics of Software Testing

The following are the characteristics of software testing,

(i) Simplicity

(ii) Understandability

(iii) Stability

(iv) Observability

(v) Decomposability.

Q2. Differentiate between verification and validation. Nov./Dec.-16(R13), Q1(g)

OR

Distinguish between verification and validation.

Answer : March-17(R13), Q1(g)

"Verification" and "validation" are two important aspects of the software testing process. Though these two phrases seems to have similar definitions, but there exists huge difference between them. Verification is the process of ensuring that the given software satisfies almost all the credentials which were laid prior to it's (software's) development. Validation is the process of ensuring that the given software satisfies the customer requirements.

Verification

It checks whether a product is being built in the right way or not.

Validation

It checks whether the right product is being developed or not.

Hence, in order to ensure that the given software is correct in all aspects, then it has to satisfy the verification and validation process successfully.

Q3. Discuss the statement, "Build robust software that is designed to test itself".

Answer :

Build a robust software that is designed to test itself mean a software design should be established in such a way that it withstands in all the worse conditions or in case of failure. A software design should exhibit all the characteristics of high quality. A software should detect all the bugs. And to do this it should use debugging techniques that easily detects the errors and resolves them. The software design should allow regression testing and automated testing.

Q4. List the principles proposed for metrics characterization and validation.

Answer : Model Paper-III, Q1(g)

The following are the principles proposed for metric characterization and validation,

1. A mathematical measurement or a specified range must be assigned to every metric. For instance, if a metric has 0 to 1 range then here '0' indicates an absence of data point, '0.5' indicates a half-way point and '1' indicates the maximum value. In addition, the metric that attempts to be measured on a rational scale should not constitute components measured on an ordinal scale.
2. A software attribute defined by a metric increases when positive characteristics are encountered and decreases when negative characteristics are encountered. Thus, the value of the metric must increase/decrease in a uniform way.
3. Before a metric is used for making decisions or advertised, it should be practically validated in several different domains. Thus, the metric should be applicable to large systems, operate in several different system devices and programming languages and also be able to measure several aspects of interest irrespective of other aspects.

Q5. Define Testing.

Answer : Nov./Dec.-17(R13), Q1(g)

Testing

Testing is the process used to estimate the productivity and quality of software. It is developed by providing the necessary details about the software such as efficiency, portability, usability, capability, etc. Testing not only performs program execution but also detect bugs that halts the execution of a program. Testing is done by comparing the static and dynamic behavior of the software with the specifications described during the process of test design.

Testing Process

Testing refers to the process of determining errors and debugging defects in order to generate error free software. The testing process involves three phases.

1. Test planning
2. Test case design
3. Test case execution.

Q6. Write about drivers and stubs.

Answer : (Model Paper-II, Q1(g) | March-17(R13), Q1(h))

The developer uses two methods for testing the program in the component testing. They are,

1. Drivers

Driver is the module where the inputs of other modules are provided by writing a main program with desired inputs which may be hard-coded or feeded by the user when the provider module is unavailable.

In other words, it can be defined as a software module which is responsible for the following,

- ❖ Invoking the module to be tested.
- ❖ Providing the inputs to the testing module.
- ❖ Controlling and supervising execution.
- ❖ Reporting test outputs.

In addition to this, it also facilitates the unit under testing as it,

(i) Initializes the required environment for testing.

(ii) Provides the desired simulated input format to the testing unit.

2. Stubs

Stubs can be defined as a software module that acts similar to a module which is called by the module under test. The stubs are more simple compared to the actual module. It pretends to be a subordinate unit and service with the minimum desired functionality for that unit.

Q7. Why is a highly coupled module is difficult to unit testing?

Answer :

Unit testing focuses on testing small units or modules of a system. The purpose of testing is to be find as many errors in the system as possible.

"Highly coupled" modules are difficult to unit testing because the level of dependency between them is very high. Dependency among modules exists in several ways.

(i) One module refers to many number of modules.

(ii) A huge number of parameters are passed from one module to another module.

(iii) High level of complexity in the interface among modules.

(iv) One module has great amount of control over other modules. Because of the high level of inter connection between modules it is very difficult to break highly coupled modules into small units for testing to trace errors and to debug. Hence, we can say highly coupled modules are difficult to unit test.

Q8. Define unit testing and top-down integration testing.

Answer :

Unit Testing

The process of executing a single unit/module without effecting other modules and comparing actual outputs with the predefined outputs in the component-level design description document is called unit testing. It concentrates on testing the data structures and internal processing logic of a module. Moreover, various units can be simultaneously tested by parallely performing unit testing for each unit. There are two strong and supporting reasons for unit testing.

Top-down Integration Testing

When the integration starts from the main control module of main program and moves down in the control hierarchy. Then such integration is called "top-down integration". Moreover the down ward movement can be either depth-wise i.e., depth-first or breadth-wise i.e., breadth-first.

Q9. What are the advantages and disadvantages of bottom-up integration?

Answer :

Advantages of Bottom-up Integration

1. Stubs are eliminated as the processing needed for component subordinates is present all the time.
2. Logic modules are thoroughly tested.

Disadvantages of Bottom-up Integration

1. Drivers are must for testing.
2. High-level modules are tested late in the testing process.
3. Early prototype is not possible.

Q10. What is regression testing? Give example.

Answer : Nov./Dec.-16(R13), Q1(h)

A well designed and well coded software when subjected to some changes may not work properly and require retesting to ensure its proper working without any errors. This retesting of software is known as "Regression testing". For example, in case of integration testing, new modules are added or old modules are removed from the software which may change the I/O, control logic and data flow paths. Regression testing is then necessary to ensure that these changes are acceptable and do not result in any errors.

The most important task to be considered in regression testing is to select the tests in the test suite. The tests selected should be optimal and must uncover maximum errors from major parts of the program.

Moreover, the regression test suite must have the tests cases of,

(a) Modified software components,

(b) Software functions that may get affected due to modifications.

(c) All software functions.

Regression testing can be conducted either manually or by capture/playback tools. These tools help the software engineer to determine the tests that were conducted on the old system and also the results obtained. The old results are compared with newly generated results. The tests cases used in testing should be documented properly such that these tests can also be implemented in future in case of changes in software.

Regression testing is not limited to integration testing as it is also carried out while software maintenance process is in progress.

Q11. Define black box testing strategy.

May-18(R15), Q1(h)

OR

Write a short note on black box testing.

Answer : Nov./Dec.-18(R16), Q1(g)

Black box testing is also known as functional testing or behavioral testing. It is an effective and efficient approach used to concentrate on the inputs, outputs and principle function of a software system module. In a black-box testing, the test of a software is based on system specifications rather, than on code. Thus, the system is assumed to be 'black box' whose behavior can only be determined by studying its inputs, outputs and functional requirements of the software. The tests can be observed from the program codes or component specifications.

Q12. How is white box testing performed using statement testing?

Answer : Model Paper-I, Q1(h)

Test values are provided to check whether each statement in the module is executed at least once. Statement testing executes statements when,

(i) Optional arguments are available

(ii) User-provided parameters or procedures are available

(iii) Planned user-actions are available.

(iv) Defined error codes are available.

Q13. Define configuration review and software tools.

Answer :

Configuration Review

Configuration review ensures that software configuration's elements are appropriately developed and recorded. Moreover, these elements should also contain sufficient information to the support phase in the software life cycle. Configuration review is the most important element of software validation. It is also known as "audit".

Software Tools

Software tools for debugging give automated assistance in debugging the problems of software. The purpose of debugging tools is to help the programmer find these problems quickly and efficiently. These tools are approached when manual debugging is difficult to implement and time consuming. Most of the available debugging tools are particular to environment and a programming language.

Q14. What is meant by smoke testing?

Answer : May/June-19(R16), Q1(g)

The software is being developed with a form of testing called "Smoke testing". The main purpose of using smoke testing is to frequently assess time-critical projects. Following are the activities carried out in smoke testing,

1. **Formation of Builds**

   Software components performing a single function are combined into a single component called "build". In addition to the coded modules, a build also includes reusable modules, libraries and data files.

2. **Design and Implementation of Tests**

   A number of tests are designed and implemented for the integrated build. These tests must be designed in such a way that the "show-stopper" errors (the errors that delay product delivery) are not left uncovered.

3. Following either the top-down or bottom-up integration, the newly formed build is combined with the existing builds. The newly formed/integrated build is smoke tested daily till all modules are integrated and tested.

Q15. List the metrics for design model.

Answer : Nov./Dec.-17(R15), Q1(h)

The metrics for design model are used to measure design attributes in such away that they aid the software engineers in evaluating the quality of design. Such metrics comprise of the following,

(i) **Metric of Interface Design**

This metric is basically focused on usability.

(ii) **Metric of Components**

This metric is used to evaluate the complexity of software architectural components as well as those aspects (characteristics) that are dependable on quality.

Metric of Specialized Object-oriented Design

This metric is used to measure classes, their communication aspects as well as the collaboration aspects.

(iv) **Metric of Software Architecture**

This metric is used to define the quality of software architectural design.

Q16. List the metrics for source code.

Answer : (Model Paper-III, Q1(h) | May/June-19(R16), Q1(h))

Metrics for Source Code

The metrics for source code are used to evaluate the source code and its characteristics like testability, maintainability. Such metrics comprise the following,

(i) **Metric of Complexity**

This metric is similar to the metric of component and is used to evaluate the logical complexity of source code.

(ii) **Metric of Software Size**

This metric is used to define the software size.

(iii) **Metric of Halstead**

This method is used to give distinct measures of a computer program.

Q17. Explain function point metrics.

Answer : (Model Paper-II, Q1(h) | Dec.-19(R16), Q1(g))

Function Based Metrics

Function based metrics are applied in measuring functionality performed by a given system. These are also referred to as function point metrics. Values for function point are derived by establishing experimental relationship between those features of software which are easily measurable and the complex nature of the software. The information domain values can be given by the count of following metrics,

1. **External Interface Files (EIFs)**

   The computer stores the data in the form of files. Hence, the files which are residing outside the application, but at the same time being utilized by the application are referred to as external interface files.

2. **Internal Logical Files (ILFs)**

   The files which are residing internal to a given application are referred to as internal logical files. The maintenance of these files are carried out by external inputs.

3. **External Inquiries (EQs)**

   The inquiries that are generated on-line and whose response is also given on-line are called external inquiries.

4. **External Inputs (EIs)**

   These are the inputs given by external sources like user or another application (i.e., external to the application) which supplies the data or control information required by the application.

5. **External Outputs (EOs)**

   These are the outputs developed by the application for the user like screens, reports, etc.

PART-B ESSAY QUESTIONS WITH SOLUTIONS

4.1 TESTING STRATEGIES

4.1.1 A Strategic Approach to Software Testing

Q18. What is software Testing? Explain test Characteristics. Dec.-19(R16) Q8(a)

OR

What is software testing strategy? Explain the desirable characteristics of it in detail.

Answer : (Model Paper-II, Q8(a) | May/June-12, Set-1, Q1)

Software Testing

Software Testing is a method of executing a software with an intention of detecting bugs. It is one of the essential steps in software development life cycle. Hence, it is the responsibility of the programmer to build an error free software.

Software Testing Strategy

Software testing strategy provides the guidelines that outline the following,

- What steps to follow in order to undergo a test?
- When to plan and workout the steps?
- How much time, input and resources must be utilized?

Hence, software testing strategy must include a proper planning for the test, a design for the test case, test execution and resultant data collection and evaluation. It must be able to promote a customized testing approach and also encourage planning and management tracking along with project progress.

Characteristics of Software Testing

The following are the characteristics of software testing,

(i) **Simplicity**

In general, the code written should exert three phases of simplicity i.e., code, structural and functional simplicity. Hence, it is a known fact that "If there is less requirement for testing, then testing takes less time".

(ii) **Understandability**

The easy the software is to understand, the efficient is the testing. Proper documentation about the software can help in clear understanding. However, documents must be accurate, specific, easily accessible and well organized.

(iii) **Stability**

A software is said to be stable if,

- Changes made to it are in control
- It can still be tested using the existing test cases.

(iv) **Observability**

It is the ability of the testing, group/team to make out whether the developed software generates correct output for certain inputs or not.

(v) **Decomposability**

Decomposing i.e., breaking the software into multiple and independent pieces called modules makes testing much easier.

Q19. What is the need of software testing? What are its main objectives and principles?

Answer : Nov.-15(R13), Q8(a)

Need of Software Testing

Software testing is performed inorder to identify and describe errors generated by lack of human skills or by bugs in the system. Apart from this, errors which are not found during reviews can be easily identified at the time of testing.

Moreover the testing is carried out so that the customers after receiving the product should not encounter any faults. Since testing is performed at initial levels, it minimizes the rework time.

Objectives of Software Testing

1. To make sure that the generated solution acquires the business and user requirements.
2. To decide user acceptability and identify errors which can be potential bugs or defects.
3. To make sure that system is ready to use.
4. To obtain the confidence showing that the product will work after testing.
5. To estimate the system capabilities showing that the system performs as desired.
6. To check the necessary documents.

Principles of Software Testing

1. Testing process should display the presence of defects.
2. It shows that complete testing is not possible.
3. It is a difficult task
4. It must be initiated as soon as possible.
5. It should be performed differently in different context.
6. It is risk-based because implementation of a particular feature is at its own risks at project level, development level.
7. It should follow a specific strategy and plan.
8. Testing team should have liberty and freedom.
9. Quality software testing should possess understanding of software product and domain related applications.
10. Testing team should regularly review and revise the test cases inorder to overcome the defects where in software product becomes immune.

Q20. What are the main objectives of software verification and validation? Briefly explain different V and V techniques.

Answer : Nov.-15(R13), Q9(a)

"Verification" and "validation" are two important aspects of the software testing process. Though these two phrases seems to have similar definitions, but there exists huge difference between them. Verification is the process of ensuring that the given software satisfies almost all the credentials which were laid prior to it's (software's) development. Validation is the process of ensuring that the given software satisfies the customer requirements.

Verification

It checks whether a product is being built in the right way or not.

Validation

It checks whether the right product is being developed or not.

Hence, in order to ensure that the given software is correct in all aspects, then it has to satisfy the verification and validation process successfully.

Verification and Validation or V and V activities for developing a software include,

(i) Formal Technical Reviews (FTR)

(ii) Performance monitoring

(iii) Quality and configuration audits

(iv) Feasibility study

(v) Simulation

(vi) Algorithm analysis

(vii) Database review

(viii) Documentation review

(ix) Usability, qualification and installation testing.

The equation depicting the relation between verification and validation with respect to testing is,

$$T = V_e + V_a$$

Where,

T – Testing process

V_e – Verification

V_a – Validation.

However, testing is highly validation oriented.

Both verification and validation use test case design methods and test strategies for software testing. The unit and integration testing strategies are used for verification purposes. Whereas, the validation and system testing strategies are used for validation purposes.

Q21. Who will test the software, either developer or an independent test group? Discuss the advantage and drawback of each one.

Answer :

Software testing is performed either by software developer or Independent Test Group (ITG) of software testers.

Advantages of Software Developer

(i) They have more knowledge regarding the software.

(ii) They are not paid additionally for testing.

Disadvantages of Software Developer

(i) They are trained and motivated for testing.

(ii) They will not show errors in their own software which may be later uncovered by customer.

(iii) They are not much familiar with testing.

The errors which are uncovered by the software developer are handled by ITG. ITG's responsibility is to remove the hidden errors in the software.

Unlike software developer, ITG is not directly involved with the software development process. Hence, they perform the tests casually.

After testing, the software developer must be available to correct errors that are uncovered. To ensure that rigorous tests are being conducted, both developer and ITG work closely till the end of the project.

ITG is part of software development project which means it gets associated at the time of analysis and design and sticks to the project till the end.

Advantages of Independent Test Group

(i) ITG assists in giving an immediate solution for testing since, they are trained and motivated.

(ii) As a result of testing, ITG can comment on software reliability before it is shipped to the user.

(iii) ITGs are more capable of finding the errors regarding system level usability, special cases, interaction among modules and performance problems.

Disadvantages of Independent Test Group

(i) ITG's must be paid for testing.

(ii) Testing performed by ITG may lead to project delays.

(iii) Testing performed by ITG leads to repetition of work. This means that the tests which were performed by software developers are again repeated by the ITG.

(iv) There is a chance of problem if there is no physical collocation between the ITG group and the design group.

Q22. What are the strategic approaches to software testing?

Answer : May-18(R15), Q8(a)

The overall strategy for software testing can be diagrammatically represented using the following spiral model.

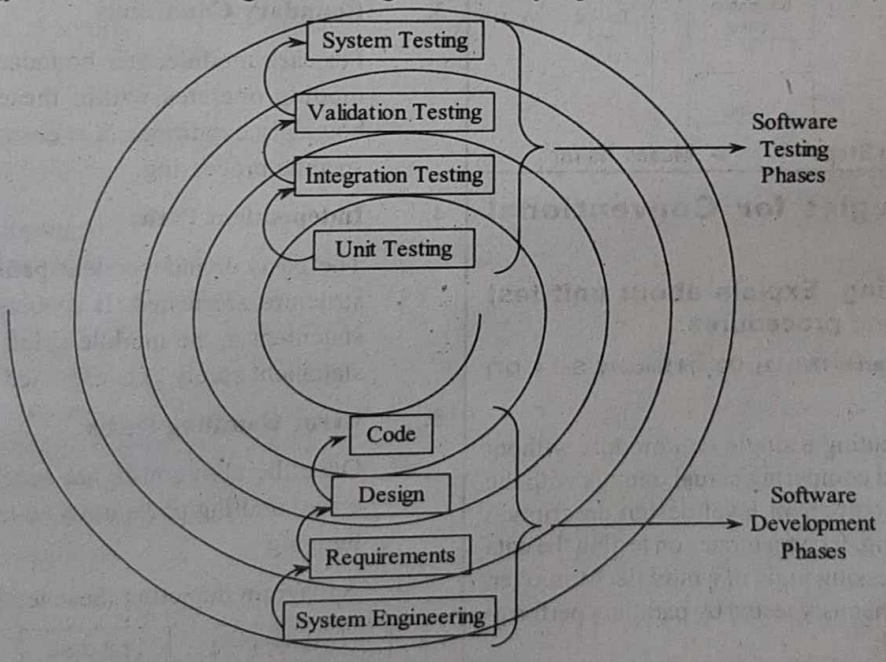


Figure: Testing Strategy Represented Using Spiral Model

An inward movement is made in the spiral in order to develop a software. However, testing makes an outward movement.

Software development begins at system engineering defines the role of the software and ends at coding. Testing begins at coding phase and ends at system engineering phase. At each level of testing, a different testing method is adopted. These methods are,

Unit Testing – At Coding Phase

As soon as the code is developed, each unit or component in it is tested individually. Such testing is called unit testing. Unit testing ensures that maximum errors are detected and entire code is tested. It follows certain paths in the control structure of a component (unit).

Integration Testing – At Design Phase

The units in the developed code are combined or integrated to generate a single package. Integration testing thus tests whether the integration results in any more bugs or not. It concentrates on software architecture's design and construction. Maximum code coverage is ensured by following certain control paths in the program.

Validation Testing – At Requirements Phase

Requirements i.e., functional, behavioural and performance requirements gathered at the requirements phase are validated against the code developed at the coding phase.

System Testing

The system engineer combines the validated software with other system components like databases and hardware and performs system testing. The testing ensures that all system components work together and perform according to the system requirements.

The testing and development steps are depicted in the below figure.

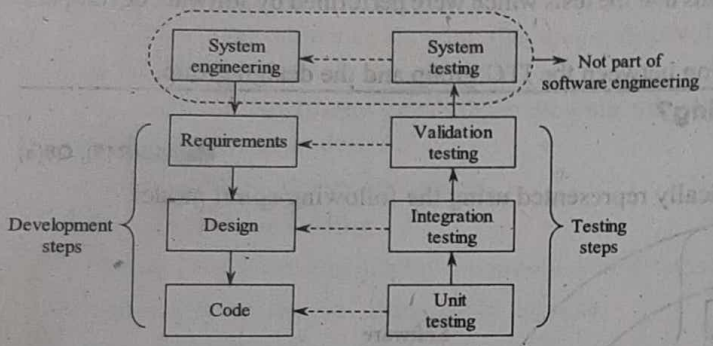


Figure: Software Testing Steps ------➤ Means "is for"

4.1.2 Test Strategies for Conventional Software

Q23. Define unit testing. Explain about unit test considerations and procedures.

Answer : **(March-17(R13), Q9(a) | Dec.-11, Set-4, Q7)**

Unit Testing

The process of executing a single unit/module without effecting other modules and comparing actual outputs with the predefined outputs in the component-level design description document is called unit testing. It concentrates on testing the data structures and internal processing logic of a module. Moreover, various units can be simultaneously tested by parallely performing unit testing for each unit.

There are two strong and supporting reasons for unit testing.

(i) Since, the size of each module is smaller than the entire software, errors can be easily and efficiently located.

(ii) Errors due to complex interactions between modules are avoided. It has few drawbacks. They are,

(a) Modules cannot call each other or pass data among themselves.

(b) Software must be decomposed into independent units/modules.

(c) Only high cohesive modules can be easily tested.

Who and When

The developers are responsible for performing unit testing. The testing is carried at least once during software development and is repeated depending on any changes made to the software.

Tests During Unit Testing

Various stand-alone tests are carried out in the process of unit testing. Each of these tests can be named as "module test" as they test a single module. Generally, five tests are carried out for testing. They are,

1. **Interface**

   Testing the module interface confirms that the information coming into and going out of the unit under test, is correct. This form of testing is called "data-flow testing" which must be conducted prior to conducting any other test.

2. **Local Data Structures**

   Data local to the module are tested in order to confirm their integrity during the execution of an algorithm. Moreover, the effect of local data on any of the global data must also be tested.

3. **Boundary Conditions**

   For each module, few boundaries are set such that the module operates within these limits. By testing the boundary conditions, it is ensured that the module does specific processing.

4. **Independent Paths**

   The basis or independent paths present in the control structure are tested. It ensures that not even a single statement in the module is left unexecuted. Thus, each statement surely gets executed at least once.

5. **Error Handling Paths**

   Once the above tests are successfully carried out, the error handling paths must be tested. This is called anti-bugging.

A diagram depicting these tests is given below.

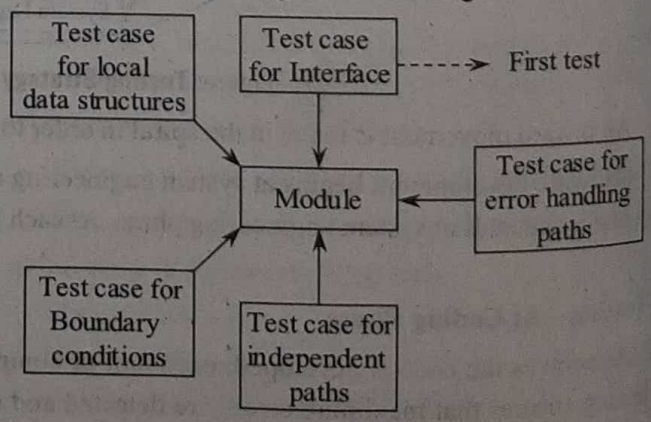


Figure: Tests in Unit Testing

Test Cases and Common Errors

The test cases for the tests in the unit testing must be designed in such a way that all practically possible errors are uncovered. The common errors that are encountered during unit testing are,

(i) Computation Errors

The most common computation errors include incorrect initialization mixed mode operations, precision inaccuracy, incorrect arithmetic precedence, and incorrect representation of expressions.

(ii) Comparison Errors

The frequent comparison errors include improper or nonexistent loop termination, inappropriate modifications of loop variables incorrect comparison of variables, incorrect precedence of logical operators, unable to exit on encountering a divergent iteration, and expectation of equality when equality is made impossible by the precision error.

(iii) Control Flow Errors

Control flow errors are generated as a result of comparison errors, since the result of a comparison changes the flow of control.

Unit Test Procedures

The first drawback of unit testing can be overcome by following unit test procedure,

It involves developing and using driver and stub(s) for the unit under test. A driver is a software that takes test case data, gives this data to the unit under test and displays appropriate outputs. On the other hand a stub acts as the subordinate module of the module under test. In the temporary role played by the stub, the subordinate modules (subordinate that is being replaced by the stub) interface is used, few data manipulations are performed, entry to the module is verified and control is given back to the unit under test. The unit test environment using stubs and driver is depicted below.

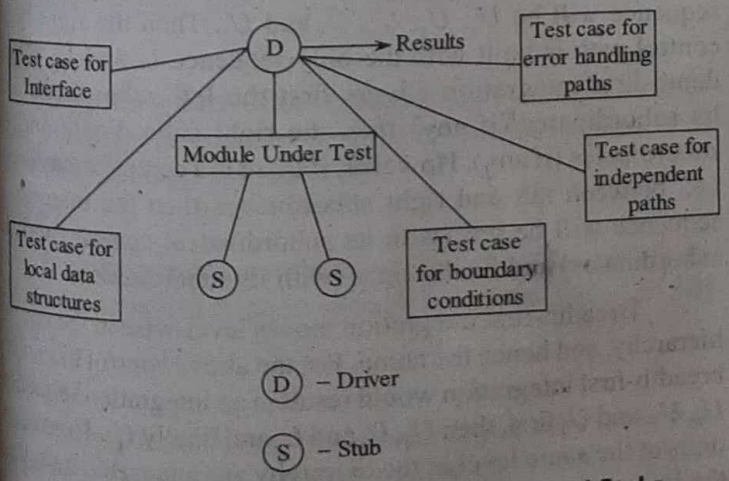


Figure: Unit Test Environment Using Driver and Stubs

Drawbacks

1. Use of driver and/or stubs adds overhead since, these two components are software and must be developed. However, these software are not a part of the end product. Even if they are kept as simple as possible, unit testing for most of the modules cannot be efficiently performed.
2. An alternative is preferably adopted to opt for automatic generation of scaffolding by using a test harness. With the help of a test harness a unit can be isolated from the entire software, while the complete system environment is simulated. The simulation includes supplying correct parameters, interaction, input and output to the unit under test.
3. The inefficient but easy way is to eliminate unit testing and ensure the coverage of a module's structure in integration testing.

Q24. Discuss in detail about integration testing. What are its types? Explain the big bang approach.

Answer : Model Paper-II, Q8(b)

Integration Testing

The components are integrated or combined to form a single architecture after unit testing. Any errors resulting from such integrations are uncovered. This process is called integration testing. Thus, integration testing is not just testing but also building the software architecture.

Need for Integration Testing

The reason for integration testing is to ensure that units that work perfectly in isolation also works perfectly when integrated. Modules when integrated may pose following problems.

(i) Data loss

(ii) Increased imprecision errors

(iii) The purpose of the modules may not be fulfilled

(iv) Modules may badly effect each other

(v) Global variables may worsen the situation etc.

Types of Integration Testing

Integration testing can be broadly categorized into two types, they are,

(i) Incremental and

(ii) Non-incremental.

The non-incremental category has only the "big bang" approach and the incremental category has,

(i) Top-down

(ii) Bottom-up

(iii) Smoke

(iv) Sandwich

The below diagram shows all these types,

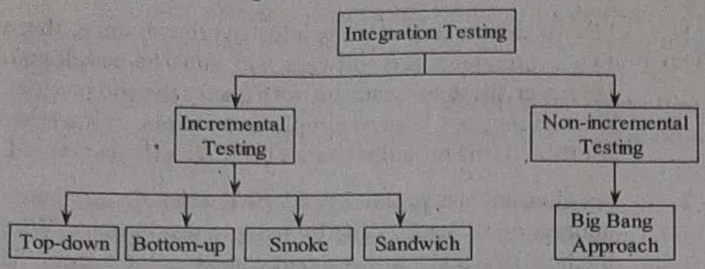


Figure: Types of Integration Testing

Incremental Testing

Incremental testing involves repeated testing of module subsets until the whole program is tested successfully. The module subsets are joined with new modules and then they are tested until all the modules are included to form the final program.

Non-incremental Testing

The units tested in unit testing are first combined to form a single software package and then the package is tested as a whole. Such approach is called non-incremental/big bang approach.

Example

Consider the program main with three units, U_1, U_2 and U_3 as depicted below,

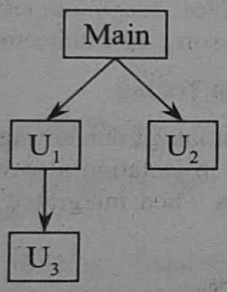


Figure: Example of Program Structure

When the big bang approach is followed, U_1, U_2 and U_3 that are unit tested are combined and then the integrated software is tested, as shown below.

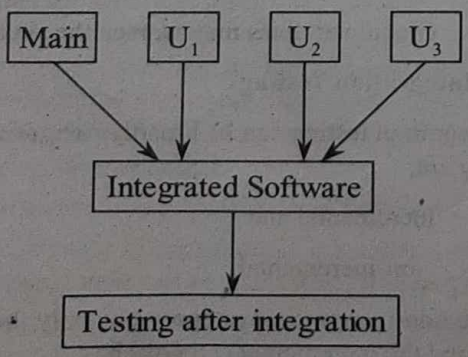


Figure: Example of Non-incremental Testing

Advantage of Big Bang

- The big bang approach can be used only for small systems. For example, the amount of time in integration testing can be relatively less.

Disadvantages of Big Bang

1. A driver and stub(s) are compulsory for each module.
2. Testing can commence only after completion of coding.
3. Modules causing errors cannot be easily located i.e., fault localization is a hectic task.
4. Most of flows that are design-oriented are uncovered at an early stage.
5. Modules critical to the system may not receive extra testing.
6. Testing may not uncover interface errors.
7. Testing may move into an infinite loop as correcting few errors may generate few more errors.
8. Customer/end user cannot view the prototype of the software.

Q25. Explain about top-down integration in detail.

May/June-12, Set-3, Q1(a)

Answer :

Top-down Integration

When the integration starts from the main control module of main program and moves down in the control hierarchy. Then such integration is called "top-down integration". Moreover the down ward movement can be either depth-wise i.e., depth-first or breadth-wise i.e., breadth-first. For example, consider below control hierarchy.

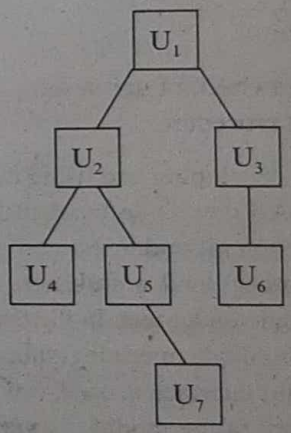


Figure: Example of Control Hierarchy

If depth-first integration is followed then the integration sequence will be U_1, U_2, U_4, U_5 and U_7. Then the right-hand control path is built with the only sequence U_3 and U_6. Thus, depth-first integration covers first the left subordinate and its subordinates (if any) then the right subordinate and its subordinates (if any). However, if there is a central subordinate i.e., between left and right subordinates then the integration sequence will be left (with its subordinates) central (with its subordinates) and finally right (with its subordinates).

Breadth-first integration moves level-wise in the control hierarchy, and hence the name. For the above control hierarchy, breadth-first integration would result in an integration sequence, U_1, U_2, and U_3 first, then U_4, U_5 and U_6 and finally U_7. This means units at the same level in the hierarchy are integrated first, then the integration moves to the next level, and so on.

Testing Steps

Top-down integration testing comprises of following six steps.

Step1

The main control unit is designated as the test driver. Units that are direct subordinates to the main unit are replaced by the stubs.

Step2

The subordinate stubs are replaced (one by one) by actual units following either the depth-first approach or breadth-first approach.

Step3

With integration of each component, the defined test cases are run.

Step4

Once the tests are done, one more stub is replaced by the actual unit.

Step5

Regression testing is performed such that the newly introduced errors (if any) can be uncovered.

Step6

Repeat steps 2 to 5 till the software architecture is completely built.

Advantages

1. Driver is not developed.
2. An early prototype of the software is possible.
3. Various implementation/testing orders are possible.
4. Errors that are design-oriented are uncovered first.

Disadvantages

1. Huge number of stubs are required.
2. Reusable units cannot be efficiently tested.
3. Third party cannot perform top-down testing and thus developers are most likely to be testers.
4. Data cannot flow upward in the control hierarchy.

The last disadvantage can be eliminated by opting any of the three choices.

Choice 1

Postponing most of the tests and perform them as soon as the stubs violates the top-down approach and makes error localization (finding) a difficult task.

Choice 2

Including minimum functionality of actual unit in its stub. This alternative is good enough unless the stubs are not complex.

Choice 3

Moving from bottom of the control hierarchy to the top, and perform integration. This alternative completely violates the top-down integration approach.

Q26. What is meant by bottom-up integration test? Explain how it is implemented.

Answer : Model Paper-I, Q8(a)

Bottom-up Integration

Bottom-up integration is an alternative to the top-down integration process. The technique initially begins by considering the units at the lowest level and proceed upwards till the main unit is reached. The steps in bottom-up integration are,

Step1

All low-level units capable of performing a specific task are combined and referred as builds or clusters.

Step2

A driver is developed for coordinating the input and output of the test case.

Step3

Each cluster or build is separately tested.

Step4

The drivers are removed and combining is started for the clusters by moving upwards.

Consider an example control hierarchy depicted below,

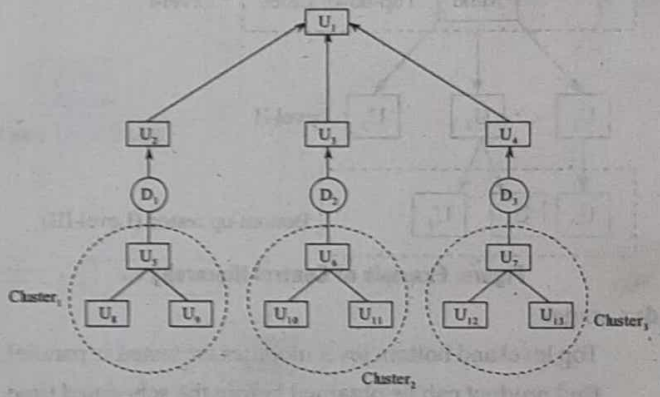


Figure: Bottom-up Approach with Drivers

Unit U_5, U_8 and U_9 are combined in to cluster$_1$, units U_6, U_{10} and U_{11} into cluster$_2$, and units U_7, U_{12} and U_{13} into cluster$_3$. For each cluster, one driver is developed for testing the three clusters. As testing is completed, the drivers are removed and each cluster is combined with its respective parent unit (a unit that is direct subordinate of main unit). The integration proceeds till all units are integrated.

Advantages

1. Stubs are eliminated as the processing needed for component subordinates is present all the time.
2. Logic modules are thoroughly tested.

Disadvantages

1. Drivers are must for testing.
2. High-level modules are tested late in the testing process.
3. Early prototype is not possible.

Q27. Among the top-down and bottom-up strategies. Which is more appropriate? Or, do we have another alternative?

Answer :

In order to select a particular strategy for testing, the following factors must be considered,

1. Number of drivers and stubs required.
2. Availability of hardware and other components.
3. Scheduling concerns.
4. Location of important parts in the complete program.
5. Customer's wish to view an early prototype.

Both top-down and bottom-up approaches have their own advantages and disadvantages. Moreover, bottom-up is preferred over top-down due to the elimination of stubs. And, top-down is preferred over bottom-up as the need for developing a driver is eliminated.

Another form of 'testing known as "Sandwich testing"(also called "hybrid testing") can be used as an alternative. In this testing, the higher levels of hierarchy are tested top-down and lower levels of hierarchy are tested bottom-up.

First, the hierarchy is divided into three layers i.e., top, middle and bottom. Based on the middle layer, the remaining two layers are decided. At the middle layer, the final integration testing takes place.

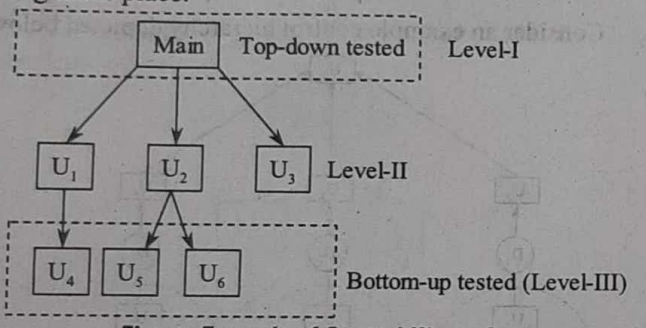


Figure: Example of Control Hierarchy

Advantages

1. Top level and bottom level modules are tested in parallel.
2. End product can be obtained before the scheduled time.

Disadvantages

1. Deciding the middle layer adds overhead.
2. Both drivers and stubs are must for testing.

Q28. Explain about "regression testing" in detail. Explain the importance of it.

Answer : Dec.-11, Set-3, Q7(a)

Regression Testing

A well designed and well coded software when subjected to some changes may not work properly and require retesting to ensure its proper working without any errors. This retesting of software is known as "Regression testing". For example, in case of integration testing, new modules are added or old modules are removed from the software which may change the I/O, control logic and data flow paths. Regression testing is then necessary to ensure that these changes are acceptable and do not result in any errors.

Importance of Regression Testing

The most important task to be considered in regression testing is to select the tests in the test suite. The tests selected should be optimal and must uncover maximum errors from major parts of the program.

Moreover, the regression test suite must have the tests cases of,

(a) Modified software components,

(b) Software functions that may get affected due to modifications.

(c) All software functions.

Regression testing can be conducted either manually or by capture/playback tools. These tools help the software engineer to determine the tests that were conducted on the old system and also the results obtained. The old results are compared with newly generated results. The tests cases used in testing should be documented properly such that these tests can also be implemented in future in case of changes.

Regression testing is not limited to integration testing as it is also carried out while software maintenance process is in progress.

Advantages

1. It maintains both the quality and reliability of software.
2. Error's due to notifications are captured.
3. The working of software according to the requirements is ensured.

Disadvantages

1. Time consuming and expensive.
2. Test suite becomes huge with progress in integration.

Q29. Define smoke testing. List the activities in it and explain how it is useful to complex and time critical software engineering projects.

Answer :

Smoke Testing

The software is being developed with a form of testing called "Smoke testing". The main purpose of using smoke testing is to frequently assess time-critical projects. Following are the activities carried out in smoke testing,

1. **Formation of Builds**
   Software components performing a single function are combined into a single component called "build". In addition to the coded modules, a build also includes reusable modules, libraries and data files.
2. **Design and Implementation of Tests**
   A number of tests are designed and implemented for the integrated build. These tests must be designed in such a way that the "show-stopper" errors (the errors that delay product delivery) are not left uncovered.
3. Following either the top-down or bottom-up integration, the newly formed build is combined with the existing builds. The newly formed/integrated build is smoke tested daily till all modules are integrated and tested.

Consider an example system with eight units, U_1 through U_8. The process of smoke testing is depicted in the figures given below.

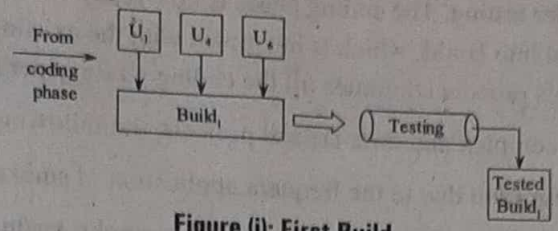


Figure (i): First Build

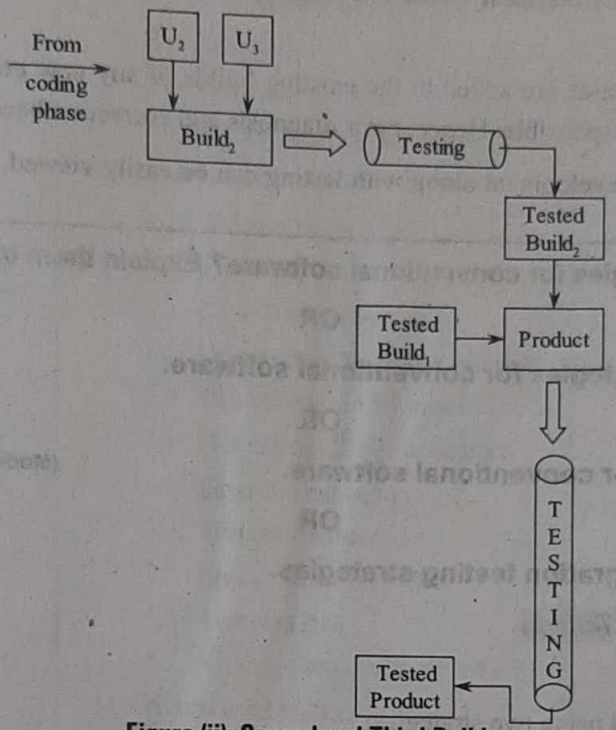


Figure (ii): Second and Third Build

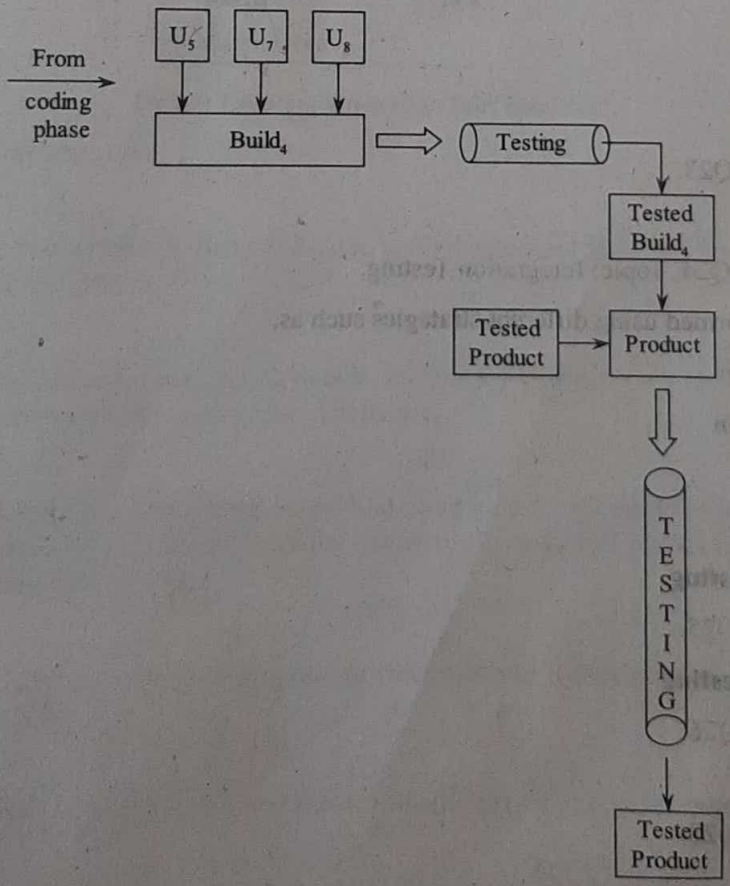


Figure (iii): Fourth and Fifth (Final) Build

Initially, the coding phase generates three units i.e., coded units U_1, U_4 and U_6 performing the same function. The three units are combined into $Build_1$ which goes under testing. The coding phase is now ready with two more functionally-related modules/ units U_2 and U_3. These units are combined into $Build_2$ which is integrated with the existing build i.e., $Build_1$ forming the product (still incomplete) which is also tested. This process continues till the coding phase generates new modules or units.

When smoke testing is applied to complex and time-critical projects, the following benefits are obtained,

(i) Reduced risks associated with integration due to the frequent application of smoke testing.

(ii) Component-level design and architectural errors are uncovered as smoke testing is carried out during the process of construction.

(iii) As testing is performed and builds are added to the existing builds, if any new errors are encountered then the newly integrated build can be made responsible. Hence, error diagnosis and correction becomes easier.

(iv) The progress in the software development along with testing can be easily viewed, boosting up the managers and team members confidence.

Q30. What are the testing strategies for conventional software? Explain them in detail. Nov./Dec.-12(R09), Q6

OR

Explain about the test strategies for conventional software. May/June-19(R16), Q9(b)

OR

Describe test strategies for conventional software. (Model Paper-III, Q8 | Nov./Dec.-17(R15), Q9(b))

OR

Briefly discuss about integration testing strategies.

(*Refer Only Topic: Integration Testing*)

Answer : Nov./Dec.-16(R13), Q9(b)

Conventional software is tested using two strategies,

1. Unit Testing
2. Integration Testing.

1. Unit Testing

For answer refer Unit-IV, Q23.

2. Integration Testing

For answer refer Unit-IV, Q24, Topic: Integration Testing.

Integration testing is performed using different strategies such as,

(i) Top-down integration

(ii) Bottom-up integration

(iii) Regression testing

(iv) Smoke testing.

(i) Top-down Integration Testing

For answer refer Unit-IV, Q25.

(ii) Bottom-up Integration Testing

For answer refer Unit-IV, Q26.

(iii) Regression Testing

For answer refer Unit-IV, Q28.

(iv) Smoke Testing

For answer refer Unit-IV, Q29.

Q31. Explain in detail about integration test documentation.

Answer :

In the process of integration, the major test cases and the entire procedure of integration must be documented. This document is called the test specification. It consists of a test plan, procedure and test results. The integration test document is shown below.

1.0 Scope
2.0 Test plan
  2.1 Test builds and phases
  2.2 Schedule
  2.3 Overhead software
  2.4 Environment and resources
3.0 Test procedure 'P'
  3.1 Order of Integration
    3.1.1 Purpose
    3.1.2 Units to be tested
  3.2 Unit testing of modules in builds
  3.3 Test description for module 'n'
  3.4 Overhead software description
  3.5 Expected test results
  3.6 Test environment
  3.7 Special techniques/tools
  3.8 Test case data
  3.9 Expected test results for build 'b'
4.0 Actual test results
5.0 References
6.0 Appendices

Figure: Template Integration Test Document

The divisions in the test document are,

1. **Scope**

Scope of testing gives the characteristics that are design, performance and functions specific. Moreover, it also gives the terminating criteria for the test phases.

2. **Test Plan**

The test plan includes local requirements, testing details, integration strategy, start and end dates for each test phase and information about testing environment and overhead software.

3. **Test Procedure 'P'**

The test procedure includes order of integration of modules, unit tests for all modules in each build, detailed information about overhead software and test for a module, testing environment, expected results from a unit and a build and special tools/techniques used during testing.

4. **Actual Test Results**

The actual results obtained after executing the program and the concerns in the test report. This division of the test document helps during maintenance phase.

5. **References**

It lists the references or sources used in preparing the document. The most common references are websites and books.

6. **Appendices**

The supplementary material providing information about the test document are placed in the appendices section. It is always given at the end of the document.

Q32. Differentiate between regression and integration testing.

Answer : Dec.-11, Set-3, Q7(b)

Integration testing checks whether the combination of two software units produces any error and checks the compatibility of the test results with the functional requirements . On the other hand, regression testing tests the modified program to detect old or new bugs occurred due to the modification of code.

Integration Testing

In integration testing, two individually tested units are combined to produce a component and testing is done between their interface. A component can include more than one unit. The concept behind integration testing is to combine all the individual units and grow the process in order to test the modules with other groups. If the program is built with more than a single process then those processes should be tested in pairs.

Regression Testing

In regression testing, if the tested program is modified then the modified program is tested again to check whether the changes gave rise to new bugs. The important points to be noted while testing is to minimize the duration of testing and the probability of detection of new bugs in the previous program. Moreover, the bugs must be fixed appropriately so that there will be no side effects in future. Regression tests must be written for the bugs whereever necessary. If more than two tests are similar then the test that is less effective must be removed. Focus of testing must be on functional issues rather than design issues. The tests that are consistently passed must be determined and achieved modification/effects on the program memory must be traced.

4.1.3 Black-box and White-box Testing

Q33. What questions do black-box tests answer?

Answer :

Black Box Testing

Black box testing is also known as functional testing or behavioral testing. It is an effective and efficient approach used to concentrate on the inputs, outputs and principle function of a software system module. In a black-box testing, the test of a software is based on system specifications rather, than on code. Thus, the system is assumed to be 'black box' whose behavior can only be determined by studying its inputs, outputs and functional requirements of the software. The tests can be observed from the program codes or component specifications.

Black box testing is helpful to software engineers in order to understand the set of input conditions, which define overall functional requirements of a program. Black-box testing is useful to identify errors such as,

(i) Inaccurate functions
(ii) Interface errors
(iii) Performance errors
(iv) Data structure errors
(v) Initialization and termination errors.

Black Box testing is performed after studying the white box testing that prints to the control of the system.

The following are the questions that black box testing answers,

1. What will be the capacity and the speed of system data?
2. What are the classes that results in good test cases?
3. How system operation is effected by specific combination of data?
4. How are system performance and behavior tested?
5. How are the Limitations of data class be destroyed?
6. Whether all the input parameters accessible by the system?

Advantages of Black Box Testing

1. This type of testing is found to be more effective compared to white box testing.
2. As no knowledge about the internal structure is required, the testers and programmers work independently of each other.
3. Test cases are performed from the user's perspective.
4. It reveals the problems or anomalies in the specifications.
5. A programming language is enough for the tester to perform accurately.
6. Tests are designed immediately after the completion of specifications.

Disadvantages of Black Box Testing

1. It requires more time for testing each and every input.
2. Unless the specifications are understandable, designing the test cases is difficult.
3. The test cases are repeated and executed until the tester is not informed of the tests that are previously executed.
4. Tests are complicated and cannot be performed directly on the specified code segments.
5. Many of the program paths remain untested.

Q34. Explain graph-based testing method with example.

Answer :

Graph-based Testing

Black box testing works in two steps,

1. The modelling objects and the relationship between them are understood in detail.
2. All the objects that have the relationship with other objects are verified by a chain of tests defined by black-box testing.

A software engineer achieves these steps by designing a graph. A graph is a collection of nodes that specifies the objects. The relationships between these objects are represented by links and link weights specify the characteristics of a link. Each node is associated with the node weight that describes the properties of a node. In graphical representation, nodes are represented as circles. All the nodes are connected with the following types of links,

(i) **Directed Link**

It describes that the relationship between the node is unidirectional.

(ii) **Bidirectional or Symmetric Link**

The relationship between the nodes exists in both the directions.

(iii) **Parallel Links**

It is used to specify the existence of more than one type of relationship between the nodes.

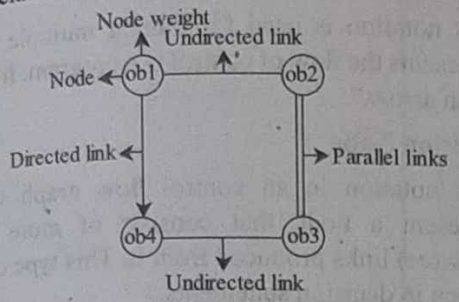


Figure (1)

Example

Consider the graph of the word processing application.

Start dimension : Default or preferences.

Background color : White.

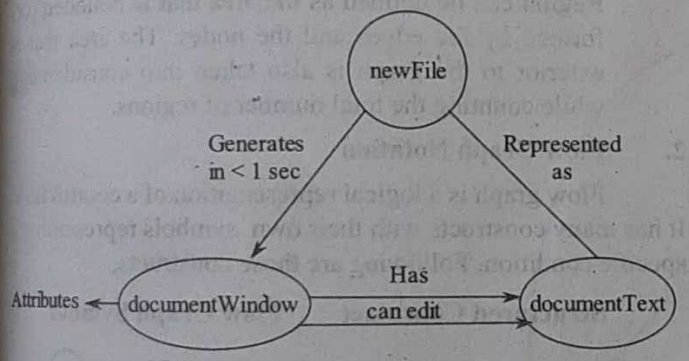


Figure (2): Graph for Word-processing Application

Graph is made by considering the important objects of the word-processing application like newfile, documentWindow and documentText and the relationships among them.

The selection of newfile menu generates a documentWindow in less than one second as shown in figure (2). This time is represented by a link weight. The node weight of the documentWindow defines some attributes such as start dimension (default size or preferences), background color (white) etc. The documentWindow and documentText are related to each other in number of ways. This is represented by parallel links between them. There is a symmetric relationship between newFile and documentText. This is shown by an undirected link between them. A software engineer can use this graph to generate test cases and find out errors in any of the relationships by traversing the graph and covering each of the relationships shown.

Q35. Explain in detail about white box testing.

Answer : Model Paper-II, Q9(a)

White Box Testing

White box testing deals with the internal logic and structure of the program code. It is also known as glass-box, structural or open-box testing. In a glass-box testing, test cases are derived based on the knowledge of software structure and its implementation. The tester can analyze the code and make use of the knowledge about the structure to derive test data. Using the white box testing, the tester can detect which module or unit is not functioning properly. This test is performed depending on the knowledge of "how" the system is implemented. White box tests can analyze the data flow, control flow, information flow, exception and error handling techniques. These techniques are used to test the behavior of the software. White box testing is done to check if the implementation is in accordance with the planned design. It is also used to check the implementation of security functions and to explore the risks.

A tester should have an explicit knowledge about the internal working of the system being tested. Branch testing and path testing are the techniques used in the white box testing. If the implementation details are changed, the tests should also be modified. The different methods used to perform white-box testing are as follows,

1. Statement testing
2. Decision testing
3. Condition testing.

1. Statement Testing

Test values are provided to check whether each statement in the module is executed at least once. Statement testing executes statements when,

(i) Optional arguments are available

(ii) User-provided parameters or procedures are available

(iii) Planned user-actions are available

(iv) Defined error codes are available.

2. Decision Testing

Tests are performed to check whether each branch of a decision is executed at least once. Decisions require two values in standard Boolean decision testing. But in case of compound or nested decisions, the number of Boolean decision values can be greater than two.

3. Condition Testing

Tests are performed to check whether each condition in a decision accepts all the necessary outputs at least once. It also checks whether the entry points to the procedure or flow is invoked at least once. In case of compound and nested loops, condition testing is provided with multiple test values. The other exceptional routines and error handlers are also invoked while testing the entry points.

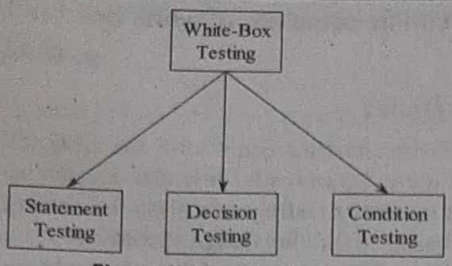


Figure: White Box Testing

Advantages of White Box Testing

1. The application is effective because of the internal knowledge of the program code.
2. The code is optimized.
3. The unnecessary lines of code which results in hidden errors are removed.

Disadvantages of White Box Testing

1. White box testing is very expensive to perform since the tester should have the knowledge about the internal structure.
2. It is not possible to examine every unit for detecting hidden errors which results in application disasters.

Q36. Give an overview of white-box testing techniques with help of flow graph.

Answer : Dec.-19(R16), Q8(b)

White-box Testing

For answer refer Unit-IV, Q35.

The working of white-box testing techniques can be understood by considering basis path testing.

Basis Path Testing

It is one of the structural testing technique which depends on the control structure of the program. This technique uses control flow graph so as to analyze the program structure. This flowgraph is prepared by using all possible paths covered in the structure and also by those that gets executed during testing. This process is referred to as path coverage. A path coverage is considered as a criterion that helps in detecting all possible errors from program structure.

Guidelines

The following are the guidelines for the effectiveness of path testing,

(a) It is a testing technique that depends upon the control structure of a program and using this a control flow graph is designed.

(b) This technique requires the tester to possess knowledge regarding the structure of the program.

(c) This technique can be better executed by the developer of the program as he knows its complete structure and can test any module of the code.

(d) As the size of the code written for a certain software increases (i.e., the structure becomes complex) the effectiveness of the path testing technique decreases. Effectiveness of path testing is inversely proportional to size of the software.

(e) The paths should be selected in such a way that they provide maximum coverage of logic present in the program.

1. Control Flow Graph

A control flow graph graphically defines the control structure of a program. It is a directed graph form (V, E) where V defines set of vertices and E defines set of edges between ordered pair of element V.

Notations Used in a Flow Graph

(a) Node

A node can be any procedural statements that are numbered (or) labelled. It is denoted by a circle

(b) Edges (or) Links

This notation is used for joining multiple nodes . It represents the flow of control in a program. It is denoted by an arrow.

(c) Decision Node

This notation in an control flow graph is used to represent a node that consists of more than one arrow(or) links produced from it. This type of scenario is seen in decision statements.

(d) Junction Node

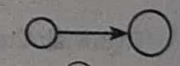
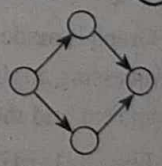
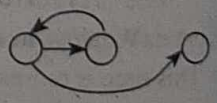
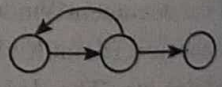
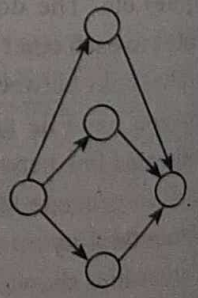
This can be defined as a node that consists of more than one arrow coming into it. They form a junction when multiple arrows meet.

(e) Region

Region can be defined as the area that is bounded (or) formed by the edges and the nodes. The area that is exterior to the graph is also taken into consideration while counting the total number of regions.

2. Flow Graph Notation

Flow graph is a logical representation of a control flow. It has many constructs with their own symbols representing a specific condition. Following are those constructs,

Structured Construct	Flow Graph Symbol
1. Sequence	
2. If	
3. While	
4. Until	
5. Case	

Consider a flowchart to represent the procedural design.

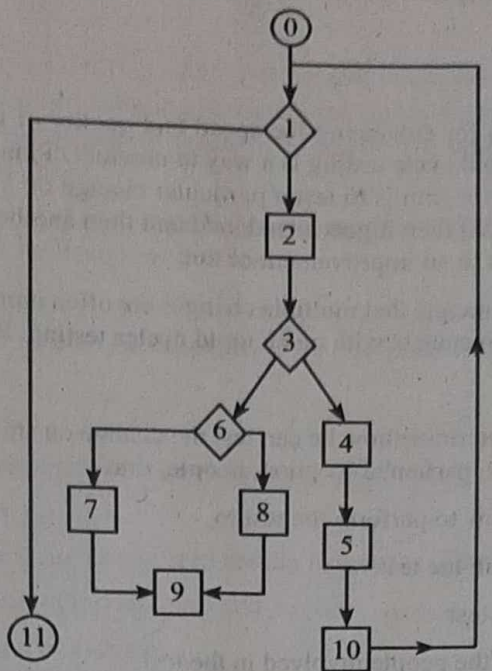


Figure: Flowchart

The flow graph for this flowchart is as follows,

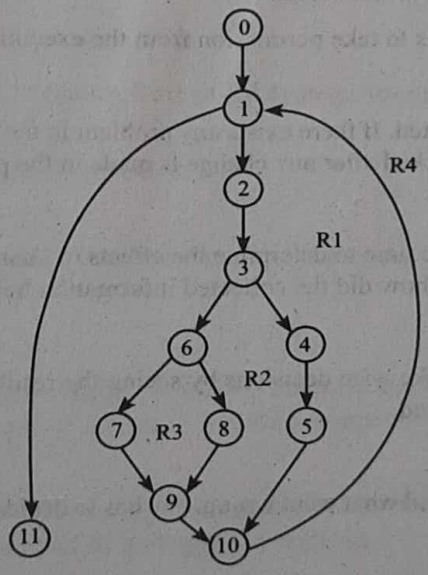


Figure: Flow Graph

A flow graph has various symbols each having a different meaning.

Following are those symbols,

(i) **Circle**

It is called a node and is used to represent statements.

(ii) **Boxes and Diamonds**

They represent data and decisions nodes.

(iii) **Edge**

It is also called as link and is represented by arrows. It indicates flow of control.

(iv) **Region**

It is an area surrounding edges and nodes.

Q37. What is rapid cycle testing? Explain the merits and demerits of it.

Answer : Dec.-11, Set-1, Q3(a)

Rapid Cycle Testing

Rapid cycle testing is an approach for enhancing the speed and quality of tests. Rapid cycle plans are not only makes things faster, but also do things better. Rapid cycle testing is a way to conduct "Plan-Do-Study-Act (PDSA)". This cycle is used in quick, rapid successions. In this cycle, the aim is to test a particular change on a small sample to see if it worked. If the test worked then it is adapted, if it didn't worked then it gets abandoned and then another test is conducted. In this way the software team determines whether the change leads to an improvement or not.

Rapid cycle testing works on the principle that multiple changes are often more effective than a major improvement with a single change. It also leads to larger improvements with quick rapid cycles testing. Rapid cycle testing has the following phases,

1. **Plan the Test**

   Before doing the test, one has to determine how he can test the change on small scale. This means applying the change to only particular things. For example, particular location, people, time.

   Detailed plan has to be made on how to perform the test to,

   (i) Identify the people involved in the test.

   (ii) Plan the date and time of the test.

   (iii) Assign responsibilities to all the people involved in the test.

   (iv) Establish any new procedures or documents that has to be followed.

   (v) Prepare data collection procedures and forms.

   Before implementing the plan, one has to take permission from the executive and alert all the staff about the plan.

2. **Do the Test**

   On has to follow the plan as documented. If there exists any problem in the plan then all the new changes should be documented again. The data should be tracked after any change is made to the plan.

3. **Study the Results**

   After performing the test, now it is the time to determine the effects of change. One has to determine, whether the change was successful and met the aim. And how did the collected information helped in the change.

4. **Act on the New Knowledge**

   After getting the results, one has to take wise decisions by seeing the results. In the next step, one has to decide whether the change can be extended and adapted.

5. **Repeat the Test**

   By considering all the shortcomings and what went wrong, one has to decide what all changes has to be applied then repeat the test.

6. **Verify Success**

   After validating the success, one has to document all the changes and their effects.

Merits

1. Minimizes resistance upon implementation of successful change.
2. Changes should be planned in a way that it leads to less disruption even if they fail.
3. Reduces risks and costs, time and money.
4. Can test ideas for change side by side with existing process.
5. Learn from the effects of success and failures of change.

Demerits

1. In rapid cycle testing, a major improvement with a single change cannot be done.
2. It gets abandoned if the test doesn't work.

Q38. Discuss about software tools for test case design.

Answer :

The test cases are designed in the implementation stage with an aim of creating a small collection of these test cases in order to achieve the goal. A test with a capability of finding undiscovered errors is considered as a good test. The objective of testing is to provide information on the concrete and abstract state of an object. The properties such as encapsulation and multiple inheritance increase the complexity of testing phase.

Software tools for test case design are used to automate the process of designing test cases for white-box and black-box testing. There are two types of software tools for designing test cases.

1. **Static Testing Tools**

Tools that test or analyze the software without executing it on the computer are known as static testing tools. These tools are divided into three types.

(i) **Code Based Testing Tools**

Takes input as source code and generate test cases by analyzing the source code.

(ii) **Specialized Testing Languages**

Another type of static testing tools are specialized testing languages. These tools are used by software engineer to,

(a) Provide complete description of the test case and

(b) Describe how the test case can be executed.

(iii) **Requirements Based Testing Tools**

These tools ensure that testing need not be carried out by using specific requirements of a user. For this, certain test cases are designed.

2. **Dynamic Testing Tools**

Testing tools that analyze the software by executing it are called dynamic testing tools. These tools are used for the following purposes.

(a) To communicate with the program being executed.

(b) To verify the path's coverage.

(c) To either test the specific variable value or make sure that the program is executed properly.

Example Tools

(a) **Panorama**

Panorama is one of the representation tools designed by ISA (International Software Automation) corporation. It assists in development of object oriented software as well as in test case design and planning.

(b) **Test Works**

It helps in automatically designing test cases for only those software that are coded in Java and C/C++. Moreover it also facilitates regression testing. It was developed by software research Inc.

(c) **McCabe Test**

McCabe test uses different techniques of path testing based on the assessment of certain software metrics like cyclomatic complexity. It was developed by McCabe and associates.

(d) **T-vec Test Generation System**

Developed by T-VEC, this tool is used to perform integration testing, unit testing and validation testing. Since it uses an OO requirements specification to help in designing test cases.

Q39. Describe Boundary Value Analysis (BVA) testing for software.

Nov.-15(R13), Q8(b)

Answer :

Boundary Value Analysis (BVA)

Boundary value analysis is an extension of equivalence partitioning that selects its test cases at the class edges instead of focusing only on the input conditions. BVA is a technique introduced to clear the errors that occurs at boundaries rather than at the centre. BVA also derives its test cases from output domain and selects those test cases that executes the bounding value.

Rules For Boundary Value Analysis

(a) A test case should be designed for values x and y along with the values that are present just below and just above them, when a limit is restricted by the values of x and y specified by an input condition.

(b) The maximum and minimum numbers should be executed by generating test cases at the time when a number of values are specified by an input condition. Values that are above and below these maximum and minimum numbers are also tested.

(c) The above 1 and 2 rules are applied on output conditions.

(d) The test cases are to be designed in order to execute the data structure at its boundary for the data structures with assigned boundaries.

These rules makes the boundary testing more complete with a higher degree of error detection.

Example

Consider a function that computes the square root of integers within the range [0, 1000]. For this function the test suite must include the following test cases {0, – 1, 1000, 1001}.

Q40. Explain Black box testing. Give an account of Equivalence partitioning and Boundary value analysis techniques. Dec.-19(R16), Q9(a)

OR

What is equivalence class partitioning? List rules used to define valid and invalid equivalence classes. Explain the technique using examples.

(Refer Only Topic: Equivalence Partitioning)

Answer : March-17(R13), Q9(b)

Black-box Testing

For answer refer Unit-IV, Q33, Topic: Black Box Testing.

Equivalence Partitioning

The derivation of test cases by partitioning the input domain of a program into classes of data is defined as equivalence partitioning. The main objective of Equivalence partitioning is to develop an ideal test case which can reveal most of the errors and minimizes the required test cases to be developed. A set of valid or invalid input conditions are represented by equivalence partitioning. Basically, an input condition can either be a particular numeric value, a set of values, a boolean condition etc.

Rules for Equivalence Partitioning

(a) A total of one correct and two incorrect equivalence classes are defined when a limit is specified by an input condition.

(b) One correct and two incorrect equivalence classes are defined when a particular value is needed by an input condition.

(c) When a member of a set is specified by an input condition, one correct and two incorrect equivalence classes are defined.

(d) For an input condition of boolean type two incorrect and one correct equivalence classes are defined.

These rules are applied while deriving equivalence classes in order to create and execute test cases for every data object in an input domain.

Example

Consider a program that computes the intersection point of two straight lines. The program needs two integer points (Slope1, Const1) and (Slope2, Const2) that form the two straight-lines and form $y = mx + c$. Then, the equivalence classes for the input of the program can be categorized as,

(i) Parallel lines which form when Slope1 = Slope2, Const1 ≠ Const2

(ii) Intersection lines that form if Slope1 ≠ Slope2.

(iii) Coincident lines that forms due to the condition Slope1 = Slope2 and Const1 = Const2.

Then the test suite is formed by taking one representative value from each equivalence class. For example the test suite can be {(4, 4) (4, 6), (10, 5) (5, 10), (2, 2) (2, 2)}.

Boundary Value Analysis Techniques

For answer refer Unit-IV, Q39.

Q41. Explain how black box testing differs from white box testing. Nov./Dec.-18(R16), Q9(b)

OR

Differentiate between black box and white box testing.

Answer : Nov./Dec.-16(R13), Q8(b)

	Structural Testing/White Box Testing		Functional Testing/Black Box Testing
1.	Structural testing is also known as white box, glass-box, open-box or clear-box testing.	1.	Functional testing is also known as black box or closed-box or opaque box testing.
2.	Structural tests are performed based on the knowledge of internal structure of the source code.	2.	Functional tests are performed without the knowledge of internal structure of the software.
3.	Testers and programmers are dependent on each other for test process.	3.	Testers and programmers are independent and performs the tests individually.
4.	Tests are performed either from programmers or designer's perspective.	4.	Tests are performed from user's perspective.
5.	The test cases take finite time but cannot detect all bugs.	5.	The test cases take infinite time and detect all errors.
6.	Inputs are represented by certain predefined paths in software.	6.	Inputs are represented by some peripheral devices or simulated systems.
7.	Structural testing is less effective when compared to functional testing.	7.	Functional testing is more effective than glass-box testing.
8.	Different methods for performing white box testing are as follows, (i) Statement testing (ii) Decision testing (iii) Condition testing.	8.	Different methods for performing black box testing are as follows, (i) Expected inputs method (ii) Boundary values method (iii) Illegal values method.
9.	It helps in removing the unnecessary code that may result in some bugs or errors.	9.	It helps in disclosing the problems and inconsistencies that may arise in functional specifications.

4.1.4 Validation Testing

Q42. Describe the following terms with respect to validation testing,

(a) Validation test criteria

(b) Configuration review

(c) Alpha and beta testing.

Answer :

The product is usually delivered to the customer at the end of software development. The customer adapts the product and checks that the software is developed to his expectation. This scenario is referred as validation testing i.e., to the extent, upto which the customer is satisfied with the developed software.

(a) Validation Test Criteria

In general terms a validation test to be complete only after analyzing that the developed software satisfies all the requirements suggested by the customer. The testing is carried out using two things – test plan and test procedure. The classes of tests are included in the test plan and the test cases in a test procedure. The test plan and procedure are designed keeping the performance requirements, correct documentation, behavioural characteristics, usability, compatibility, maintainability, error recovery, transportability and functional requirements in focus.

Once the validation testing is done, either of the two outcomes are possible,

(i) Functional or performance characteristics are as per specification, resulting in their acceptance.

(ii) A variation from the specification is identified, resulting in the creation of a deficiency list.

If the outcome falls in the second possibility, then the customer must be contacted and a way to resolve the deficiencies must be obtained since, the product cannot be corrected at the stage when the scheduled delivery date is almost reached.

(b) Configuration Review

Configuration review ensures that software configuration's elements are appropriately developed and recorded. Moreover, these elements should also contain sufficient information to the support phase in the software life cycle. Configuration review is the most important element of software validation. It is also known as "audit".

(c) Alpha and Beta Testing

When the software is developed completely, tested suitably and reviewed thoroughly, it has to be tested by the users. This is because, the user may not be able to understand the instructions written or may not be comfortable with the outputs even though these outputs satisfy the developers. This form of testing is called acceptance testing. The duration of acceptance testing can vary from weeks to months. Such a prolonged form of testing eliminates the degradation in the system that may evolve with time.

Acceptance testing proves to be an efficient form of testing when the product is developed for a single user. However, if there are more than one user, then other types of testing – alpha and beta testing must be used.

(i) Alpha Testing

During the alpha testing, several users are called to the developer's site and are exposed to the product. Under the guidance of the software engineer, the user uses the product. In the mean while, the software engineer notes all the possibilities like, the problems faced by the users during normal operations, the instructions that are difficult to understand, the operations generating bugs, etc.

(ii) Beta Testing

During the beta testing, the product is delivered to the users. Here, in the absence of the software engineers, the product is tested and a report is prepared by the user. The report contains details of the difficulties faced by the users during the product operation. After analyzing the report, the software engineer makes suitable modifications to the software and then supplies it back to the users.

Q43. What is the need of beta testing?

Answer : (Model Paper-I, Q8(b) | May/June-19(R16), Q8(b))

Beta Testing

Beta testing is needed for revealing problems related to usability and performance without involving development team. Beta testing can be defined as the process of testing where the test is conducted for the alpha tested product so as to enhance the assurance about the readiness of the product for shipping. The product in this testing should be complete and usable in the production environment. Moreover, it involves only few customers for the assurance purpose. It generally employs black box techniques.

Entry Criteria to Beta

The beta test will be conducted if the following criteria is met.

- Alpha sites result is positive
- Alpha testing has been performed with respect to customer bugs
- Regression testing has been conducted in accordance to bug fires
- The system should not contain fatal errors
- Secondary platform compatibility testing has been finished
- Beta sites are in the state of installation.

Guidelines for Beta Testing

The following are the guidelines that helps in conducting Beta test.

1. Beta test should not be conducted for less than four releases
2. Do not add even a small feature during the beta process as it takes back tester to the starting of 8 weeks and also requires 3-4 releases
3. Give atleast 15 days time to the beta testers for releasing new builds.

Q44. Show using a small example, why it is practically impossible to exhaustively test a program?

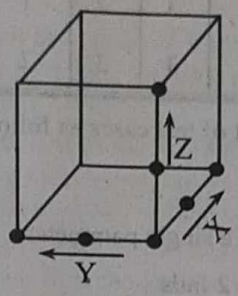
Answer :

Nov./Dec.-18(R16), Q8

Orthogonal array testing is done when the input parameters and values are small in number but too large to perform exhaustive testing. This testing strategy can be used to reduced the number of test combinations and provide maximum coverage with a minimum number of test cases. Whereas in case of exhaustive testing, it involves considering all the possible combinations and variations of test cases. Testing all the combinations is possible when the number of input domain is restricted, but becomes impossible when number of input values and discrete values increases.

Orthogonal array testing is particularly effective in finding errors associated with faulty logic within the computer software component.

Now, compare orthogonal array testing and traditional "one input at a time technique" for example consider three parameters A, B and C, suppose each of them have positive values 1, 2, 3, in case of exhaustive testing, all combinations of the three parameters would involve executing a total of 27 test cases. Let us look at the geometric view of the all possible test cases in which one input value goes in sequence along with each input axis x, y, z.



Figure

On the other hand, orthogonal array testing selects combination of parameters effectively and uniformly. It uses $\angle 9$ orthogonal array of test cases. It chooses a set of nine combination of parameters which effectively finds the faulty logic by uniformly spreading the testcases throughout the test domain.

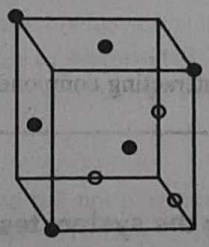


Figure: $\angle 9$ Orthogonal Array

Now, consider an example of $\angle 9$ orthogonal array. Consider a fax application's send function which contains four parameters with discrete value as,

$P1 = 1$, send it now

$P1 = 2$, send it after an hour

$P1 = 3$, send it after midnight.

Other parameters like $P2$, $P3$, $P4$ also take values of 1, 2, 3 corresponding to other send functions.

In case of traditional one input at a time technique the sequence goes in the following way,

(1, 1, 1, 1), (2, 1, 1, 1), (3, 1, 1, 1), (1, 2, 1, 1), (1, 3, 1, 1), (1, 1, 2, 1), (1, 1, 3, 1), (1, 1, 1, 2) and (1, 1, 1, 3).

Traditional technique finds faulty logic of single parameter value which is called as single mode fault. In case of exhaustive testing, $81^{(3+)}$ possible test cases are required to find the bug and it requires high efforts. Whereas in case of orthogonal array testing only 9 test cases are required as shown below,

	Test Parameters			
Test Case	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

The given $\angle 9$ orthogonal array assess the result of test cases as follows,

(i) **Single Mode Faults**

In single mode fault it can detect logic faults of a single parameter.

For example, all test cases with parameter $P1 = 2$ fails.

Hence, the condition "send it after an hour" is not working properly in the send function.

(ii) **Double Mode Faults**

Double mode fault is caused when two specific parameter values are used together. It indicates incompatibility between the pairs.

(iii) **Multimode Faults**

In multimode fault occur when more than two interacting components produces inconsistent output orthogonal arrays can also detect multimode faults.

4.1.5 System Testing

Q45. What is software testing? Explain clearly the system testing. May-18(R15), Q9(a)

OR

Explain the methods of system testing. Nov./Dec.-17(R15), Q8(a)

OR

Explain about stress testing, performance testing.

(Refer Only Topics: Stress Testing, Performance Testing)

Answer :

Software Testing May-13(R09), Q6(a), (b)

For answer refer Unit-IV, Q18, Topic: Software Testing.

System Testing

In a computer-based system, there is more than software i.e., hardware, operating system etc. Thus, when the software is developed, it must be integrated with other elements and tested. This type of testing is called system testing, which is not a part of the development life cycle. The software engineer is not solely responsible for conducting system testing as the system engineer is the one that actually performs system testing. The only responsibility of the software engineer is to sit with the system engineer and check if there are any errors due to software development. Even this can be avoided if the software engineer takes extreme care while designing, planning and testing the software.

Types of System Testing

There are four types of system testing for software-based systems. They are,

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing.

1. Recovery Testing

The most important and desirable feature of a computer-based system is "fault tolerance". Meaning, the system if effected by some faults or failures should resume processing within a stipulated period of time. Moreover, the system should be strong enough to overcome the failures and function normally. To know whether a system is fault-tolerant, try to fail the system in all aspects and measure its recovering/resuming capabilities. This type of system testing is called recovery testing.

The system under test may either use automatic recovery or human-intervened recovery approach. In the former case, the system itself performs recovery, but does require that the correctness of checkpoint mechanisms, reinitialization, restart and data recovery is checked. In the latter, manual recovery is done. MTTR (Mean Time to Repair) is the only factor used to check whether manual recovery is within the specified limits or not.

2. Security Testing

Computer-based systems are the systems that are prone to security attacks. Penetration into these systems can be from hackers, employees of the organization in which the system is installed and many other outsiders. With security testing, it can be ensured that such penetrations do not effect the system security.

In order to carry out security testing, the tester acts as the penetrator and tries all possibilities of breaking the security. For example, knowing passwords by paying the clerks or employees, browsing through malicious data, etc. The tester realizes that if proper amount of resources and enough time is given then any system with any level of security can be penetrated.

Thus, the responsibility of ensuring security of a system also lies on the system designer. The design must be so tricky that the profit obtained by penetrating should be made less than the amount or effort invested in penetrating.

3. Stress Testing

When the developed software is targeted to abnormal situations like generating twenty interrupts/second whereas the average rate is three interrupts/second, increasing data rates, requesting maximum memory, intentionally causing memory management issues, huge requirements for searching data on disk, etc., then the system is being stress tested.

In stress testing, the aim of the tester is to design such test cases that can overwhelm the program. Any real-time system must undergo stress testing to know the maximum functionality a system can provide before it fails.

However, a system may fail when it is requested to perform beyond its capability, and also when everything is within the limits. The data may also be valid but may degrade the performance of the system. Testing a system for valid data inputs resulting in erroneous processing is called "sensitivity testing".

4. Performance Testing

Any system that meets functional requirements but not performance requirements cannot be delivered to the customer. Performance testing ensures that the software meets the performance requirements. It is performed during the entire process of testing from the lowest level of testing i.e., unit testing till the highest level i.e., system testing. With this, the performance of the system is ensured. Performance testing is frequently used along with stress testing.

Q46. Distinguish between error and failure. Which of the two is detected by testing? Justify.

Answer : Nov./Dec.-18(R16), Q9(a)

Error

The term "error" can be defined in two different ways,

1. It is difference (discrepancy) between a computed, observed or measured value and the true, specified or theoretically correct value i.e., the difference that exist between the output of a software and the theoretically correct value.
2. In general, the human actions due to which there can be a defect is software in referred to as an error.

Failure

It is defined as the inability of a system or component to behave as per the system specifications. The occurrence of failures may be due to the functional or performance factors. A failure can be recognized by the behaviour of the software that is different from the specified behaviour. Some times it happens that the failure occur but are not detected.

The terms error, fault and failures are interrelated to each other. Since the presence of error indicates that some failure has occurred, which ensures that some fault is present within the system.

Among these two terms (i.e., error and failure), error is detected by testing. Since, the basic role of the testing is to identify the errors that are present in the program.

4.1.6 The Art of Debugging

Q47. Discuss about art of debugging in detail. (Model Paper-III, Q9(a) | May-18(R15), Q8(b))

OR

Discuss the process of debugging. May/June-19(R16), Q8(a)

Answer :

The process of removing bugs is called debugging. It is performed after uncovering all the bugs in testing. The process of debugging is depicted below.

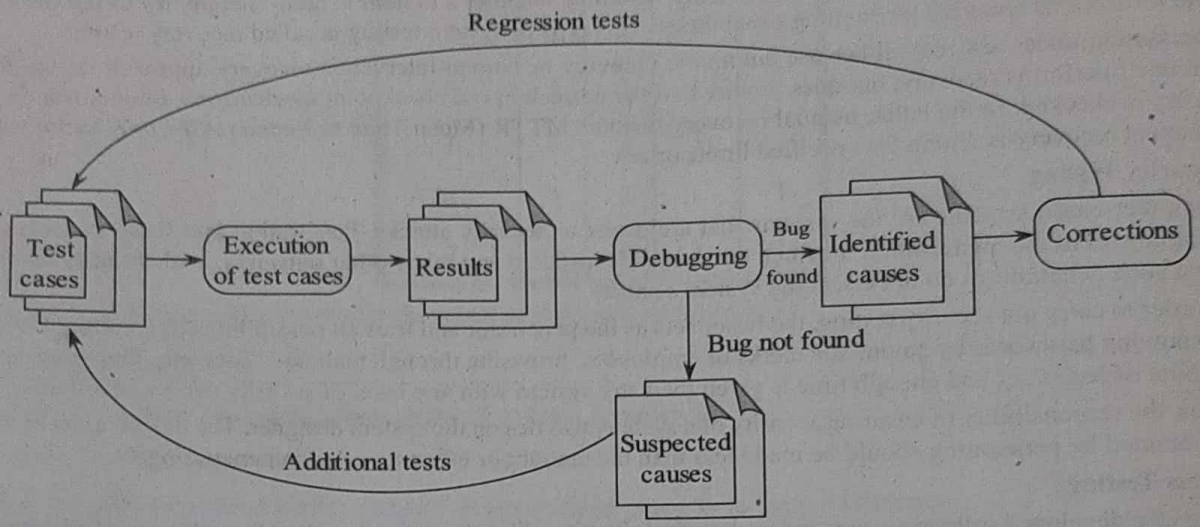


Figure: Debugging Process

Firstly, the test cases are executed and the results are gathered. These results are compared with the expected results. Any variation between the two results a bug. The cause of the bug may be either identified or it still remains hidden. In the second case, the professional performing debugging may suspect the cause, design test cases for it and repeatedly execute these test cases, till the bug is corrected. In the above figure, the two outcomes from debugging represents the above two cases. On successfully identifying the causes, corrections are made and regression testing is carried out, ensuring that everything works well after modifications.

The process of debugging continues till all errors are fixed. There is no specific format for debugging, rather it is an art and largely depends upon the psychology of the professional performing debugging. The professional may not be willing to accept the errors as debugging may increase difficulties, ask for frustrating problem solving and brain teasers, etc.

Q48. State and explain various debugging tactics.

Answer :

The main aim behind the debugging process is to find the cause and accordingly eliminate the error. To do this, several systematic approaches can be adopted or by suspected locations are analyzed. (i.e., the error may be present at certain location). Sometimes it also depends on the luck or the combination of all these three strategies. However, there is no direct method which can take us to the cause of error but there are three basic methods derived for the sake of software engineers,

(i) Brute force approach

(ii) Backtracking mechanism and

(iii) Cause elimination process.

(i) Brute Force Approach

Brute Force approach is the most common approach of finding the cause of an error. However, it often takes more of human approach but rarely satisfies him. The main principle of brute force approach is to involve the computer in finding the causes of an error. Hence, keeping this strategy to be of prime focus, the entire program code residing in computer memory is considered and its run time operation is closely analyzed (while it is under execution). Also, its outputs are mapped with respect to execution of each line of code. Hence, in this bulk process sometimes the sources of errors get easily traced. Though this process is applied frequently, but majority of times it is unsuccessful.

(ii) **Backtracking Mechanism**

Backtracking proves effective only when applied to smaller programs. In this mechanism developers initially look for the symptoms of errors and then they start moving backwards manually until the cause of error is identified. However, as number of source lines increases, managing backward paths becomes difficult.

(iii) **Cause Elimination Process**

This process begins by organizing those parts of code which include traces of errors. As a next step, a cause hypothesis is written. Now, this hypothesis is mapped to the organized lines of code. Hence, in this way validity of the given hypothesis can be determined. If the hypothesis becomes valid, then real causes of errors are tested and the code is refined so as to remove the error.

Automated Debugging

This process suggests the usage of certain debugging tools to be used along with the debugging strategies mentioned above. Usage of these tools are given vital importance, since they improve the performance of debugging. Few of these tools are,

(a) Cross-reference mapping tools

(b) Dynamic debugging aids

(c) Debugging compilers

(d) Automatic test case generators etc.

Q49. Discuss about software tools for debugging.

Answer :

Software Tools

Software tools for debugging give automated assistance in debugging the problems of software. The purpose of debugging tools is to help the programmer find these problems quickly and efficiently. These tools are approached when manual debugging is difficult to implement and time consuming. Most of the available debugging tools are particular to environment and a programming language.

The following is the list of few software debugging tools.

(i) **C++ Test**

A unit testing tool developed by Parasoft, supports the entire range of tests that can be carried out on the code written in C and C++.

(ii) **Tprobe ThreadAnalyzer**

This tool is developed by Sitraka which assists in the analysis of various thread problems such as stalls, race conditions and dead locks. The analysis is crucial as these problems largely affects the performance of Java applications.

(iii) **CodeMedic**

This tool is developed by New-Planet Software. For the standard UNIX debugger '*gdb*', CodeMedic provides a graphical interface along with an implementation of important features in gdb. Most programming languages such as C/C++, Java, FORTRAN, module-2, assembly language are supported by '*gdb*'. Moreover, various embedded systems and PalmOS are also supported.

(iv) **GNATS**

GNATS is a freeware application. It is a collection of tools which assists in detecting bugs.

(v) **BugCollector Pro**

This tool was developed by NesbittSoftware corporation. It implements a multiuser database. This database not only helps the software team in monitoring reported bugs but also maintains other requests and manages the workflow of debugging.

4.2 PRODUCT METRICS

4.2.1 Software Quality

Q50. What is meant by software quality? Discuss clearly the McCall's software quality factors.

Model Paper-I, Q9(a)

OR

Explain the factors that affect software quality.

Answer : Nov.-15(R13), Q11(a)

Software Quality

When a software's documented development standards, functional and performance requirements and implicit characteristics are achieved then its quality is said to be achieved.

McCall's Quality Factors

Following figure depicts the McCall's software quality factors.

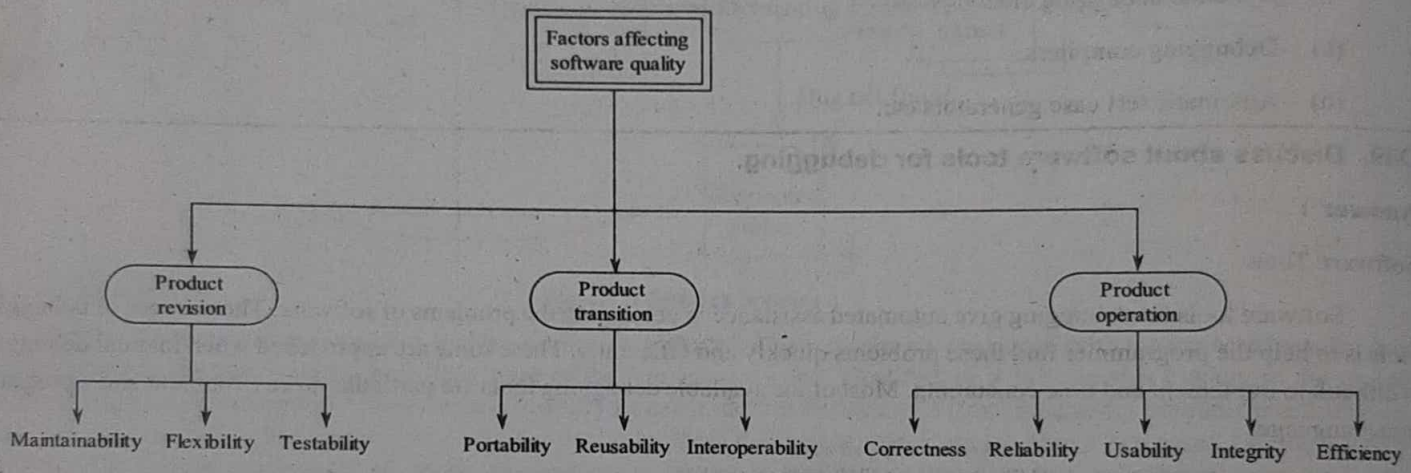


Figure: McCall's Quality Factors Tree

As shown in the above figure, McCall has grouped the software quality factors (i.e., the entity which may have their impact on the software quality) into three categories. They are,

1. **Product Operation**

In this category, the factors which may exert their impact while the product is under operation are grouped. These factors are,

(i) Efficiency

The quantity of resources and code utilized by a given program to deliver its services.

(ii) Integrity

The internal ability of the software in avoiding unauthorized access to data or software.

(iii) Usability

The amount of effort needed in performing all activities – learning, operating, preparing input and interpreting output, of a given software.

(iv) Reliability

The ability of a program to perform its function with adequate amount of accuracy.

(v) Correctness

The ability of the given software satisfying its purpose of developing the software at the same time satisfying the customer requirements.

2. **Product Transition**

In this category, the factors which may exert their impact while the product is being moved from one environment to another are grouped. These factors are,

(i) **Interoperability**

It refers to the amount of effort applied to logically join one system to another system.

(ii) **Reusability**

It refers to the ability of the program or modules of the program to be reused in developing other applications.

(iii) **Portability**

It is the ability of the program being adaptable to any kind of hardware/software.

3. **Product Revision**

In this category, the factors which may exert their impact while the product undergoes changes are grouped. These factors are,

(i) **Testability**

The amount of effort applied while testing a product's functionality.

(ii) **Flexibility**

The amount of effort applied to make suitable changes to the existing product (software).

(iii) **Maintainability**

It refers to the amount of effort applied in repairing a given software.

Q51. What is software quality? Explain the determinants of software quality in detail.

May/June-12, Set-2, Q4

OR

Discuss about determinants of software quality.

(Refer Only Topic: Determinants of Software Quality)

Answer : May-13(R09), Q7(a)

Software Quality

For answer refer Unit-IV, Q50, Topic: Software Quality.

Determinants of Software Quality

A process is one of the controllable factors that improves the software quality. There are three determinants that extremely influence the software quality and organizational performance. They are,

1. People
2. Product and
3. Technology.

1. **People**

The factor that mostly influences quality and performance is the skill and motivation of people.

2. **Product**

The complexity of the product affects the team performance and software quality significantly.

3. **Technology**

The software engineering methods and tools are the technologies that help to populate the process, also have an impact on the quality and performance.

The below figure depicts three determinants of the software quality.

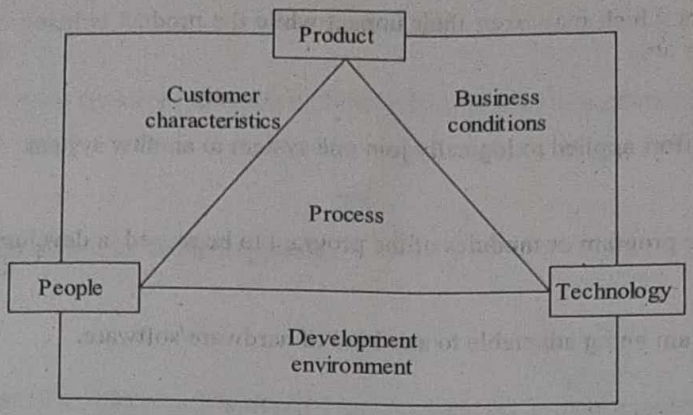


Figure: Determinants of Software Quality

In the above figure, the process connecting three determinants is surrounded by three other environmental conditions. They are, the development environment such as integrated software tools, business conditions deadlines, business rules and customer characteristics and communication and collaboration.

Q52. Explain ISO 9126 quality model with a neat sketch.

Answer :

(Model Paper-III, Q9(b) | Nov.-15(R13), Q10(a))

ISO 9126 Quality Model

The ISO 9126 is an international standard for software product quality measurement. The standard comprises four parts which are quality model, external metrics, internal metric and quality. However, the emphasis is given only on quality model. In simple terms, the model structures the software quality into six quality factors covering functionality, reliability, usability, efficiency, maintainability and portability. These factors can be used as a basis for indirect measures. This standard presents the structure for quality requirements and quality evaluation. Moreover, the quality of a system can be assessed by using factors in the form of a checklist. The factors are,

(i) Portability

It refers to the ability of a given software adaptable to any kind of hardware/software. Hence, installability, adaptability, replaceability and conformance can be termed as the sub-attributes of portability.

(ii) Efficiency

If refers to the ability of a software to use system resources in an optimal way. Time behaviour and resource behaviour are the sub-attributes of efficiency.

(iii) Maintainability

If refers to the ease with which a software can be repaired. Hence, stability, changeability and analyzability can be termed as the sub-attributes of maintainability.

(iv) Functionality

It refers to the extent to which a given software satisfies its purpose of development. Hence, security, accuracy, suitability, compliance and interoperability can be termed as the sub-attributes of functionality.

(v) Usability

It refers to the ease of using a software. Hence, operability, understandability and learnability can be termed as the sub-attributes of usability.

(vi) Reliability

It refers to the time span during which the given software can be used. Hence, recoverability, maturity and fault tolerance can be termed as the sub-attributes of reliability.

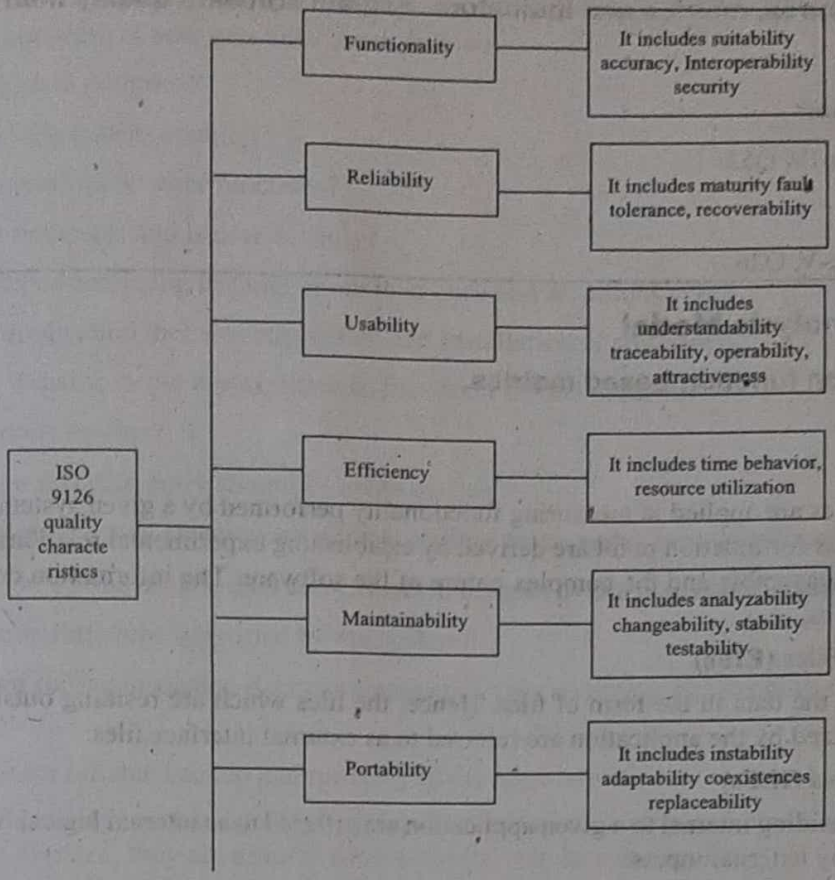


Figure: ISO 9126 Characteristics

Q53. Describe the framework for software product metrics.

Answer : Nov./Dec.-16(R13), Q8(a)

The framework for product metrics are as follows,

(i) Measures

(ii) Metrics

(iii) Indicators.

(i) Measures

A measure is defined as a quantitative indication that gives description about the size, amount, extent or capacity of a certain attribute. It is initiated whenever a data point is collected.

Here, data point may refer to errors (of a software component) that has not been recovered. The measure is usually determined by a unit called 'measurement'. Measurement can be evaluated by collecting at least one data point i.e., by collecting the number of errors in each component when reviewing and unit testing is being performed.

(ii) Metrics

A metric is defined as a quantitative measure of having a certain attribute in a system/process/component. This means the degree upto which the process may constitute a specified attribute. Typically, every software is dependent on every measure such as the average number of errors detected are dependent on the unit test/review of the system component.

A software metric can do the following,

1. Aid the design such that an efficient testing can be done.
2. Indicate that the source code and design are associated with the complexity measures.
3. Aid to derive the measures and analysis of design models.

(iii) Indicators

An indicator is defined as a metric or a collection of metrics using which detailed information about the software project/ process/product can be obtained. It also aid the software engineers or the project managers in such a way that the product, project, process are adjusted so as to make the components in a better and efficient way. Thus, indicators are evaluated by collecting measures and developing metrics by a software engineer.

Q54. Differentiate measures, metrics and indicators. Explain software quality metrics.

Dec.-19(R16), Q9(b)

Answer :

Measures, Metrics, Indicators

For answer refer Unit-IV, Q53.

Software Quality Metrics

For answer refer Unit-V, Q26.

4.2.2 Metrics for Analysis Model

Q55. Discuss in detail on function-based metrics.

Model Paper-I, Q9(b)

Answer :

Function Based Metrics

Function based metrics are applied in measuring functionality performed by a given system. These are also referred to as function point metrics. Values for function point are derived by establishing experimental relationship between those features of software which are easily measurable and the complex nature of the software. The information domain values can be given by the count of following metrics,

1. **External Interface Files (EIFs)**

   The computer stores the data in the form of files. Hence, the files which are residing outside the application, but at the same time being utilized by the application are referred to as external interface files.

2. **Internal Logical Files (ILFs)**

   The files which are residing internal to a given application are referred to as internal logical files. The maintenance of these files are carried out by external inputs.

3. **External Inquiries (EQs)**

   The inquiries that are generated on-line and whose response is also given on-line are called external inquiries.

4. **External Inputs (EIs)**

   These are the inputs given by external sources like user or another application (i.e., external to the application) which supplies the data or control information required by the application.

5. **External Outputs (EOs)**

   These are the outputs developed by the application for the user like screens, reports, etc.

Information Domain Value	No. of	Weighting factor			Total
		Simple	Average	Complex	
EIFs	_____ ×	5	7	10	= _____
ILFs	_____ ×	7	10	15	+ = _____
EQs	_____ ×	3	4	6	+ = _____
EOs	_____ ×	4	5	7	+ = _____
EIs	_____ ×	3	4	6	+ = _____

The formula for calculating function point is,

$$FP = \text{Total} \times \left[(0.65) + 0.01 \times \sum_{i=1}^{14} f_i \right]$$

Total value is obtained from above table and f_i is the Value Adjustment Factor (VAF) which depends on the answers to be provided for a set of fourteen questions. These questions are as follows,

1. Is performance of the system complex?
2. Is internal processing of the system complex?
3. Is the code reusable in developing other functions?
4. Does the system support changes and is user friendly?
5. Is the application developed for, being installed more than once and at various sites?
6. Does the design of the application includes conversion and installation of software?
7. Is their any complexity existing in the inputs, outputs, files and external inquiries?
8. Are updations to ILFs done on-line?
9. Does the system requires that data entry should be made on-line?
10. Is the system capable of working in an environment that already exists and is heavily utilized?
11. Does the input transaction for the on-line data entry needs to be built over a number of operations or screens?
12. Are distributed processing functions supported by the system?
13. Is there any requirement for sophisticated data communications for receiving and sending information from and to the application respectively.
14. Is their any requirement for reliable backup and recovery of the application?

Answers to these questions assign certain weights by using a numerical scale starting from '0' and extending to a maximum of '5'. Once these values are computed, they are directly substituted into the formula for FP.

Uses of Function Points

Certain uses of function point metrics are,

(i) It is essential in predicting the number of units as well as number of lines of code, the software may accommodate which is under development.

(ii) Estimation of errors which are likely to be uncovered during testing.

(iii) It can also be used in predicting the total cost and effort to be applied in developing the given application.

Q56. What are the metrics for specification quality?

Answer :

Metrics for Specification Quality

One of the metrics for the analysis model are metrics for specification quality. These metrics are applied on the features that focus on assessment of analysis model's quality and the related requirements specification.

For this, Davis uses specificity, reusability, traceability, verifiability, understandability, achievability, modifiability, correctness, completeness, concision, precision, internal consistency and external consistency.

To represent these characteristics using the specification quality metrics, let the number of requirements in a specification be denoted by N_R.

It is obtained by adding number of functional requirements (N_F) and number of nonfunctional requirements (N_{nF}), i.e.,

$$N_R = N_F + N_{nF}$$

In addition to measuring screen cohesiveness, this metric is also used to measure,

(a) The test size and density on the screen.

(b) The number of content (data) objects displayed on the screen.

(c) The amount of time needed to perform a particular task.

(d) The amount of time needed to recover from any error condition.

Metrics for specificity is computed by calculating a metric that is designed on the basis of how consistency between the requirement specification is interpreted by the reviewers. This type of metric is obtained by dividing the number of reviewers interpretation of every requirement specification (N_{UI}) with the number of requirements in a specification (N_R).

Therefore,

$$M_S = N_{UI} / N_R$$

Where M_S = Metric for specificity.

If the value of $M_S = 1$, then specificity will be more.

Metrics for completeness is calculated by dividing the number of unique function requirements (N_{UF}) with the cross product of number of specification inputs (N_I) and number of states specified (N_S). It can written as,

$$M_C = N_{UF} / (N_S \times N_I)$$

Where,

M_C = Metric for completeness of function requirements.

The problem of using the above formula is that it does not consist of nonfunctional requirements. To overcome this, the above formula is modified by using number of requirements validated as correct (N_V) and number of non-validated requirements (N_{nV}).

Therefore, the modified metric for completeness is,

$$M'_C = N_V / N_V + N_{nV}$$

Q57. Explain the metrics for Analysis Model.

Nov./Dec.-17(R15), Q8(b)

Answer :

For answer refer Unit-IV, Q55, Q56.

4.2.3 Metrics for Design Model

Q58. What are the metrics for design model? Discuss in detail the architectural design metrics.

Answer :

Metrics for Design Model

The metrics for design model are used to measure design attributes in such away that they aid the software engineers in evaluating the quality of design. Such metrics comprise of the following,

(i) Metric of Interface Design

This metric is basically focused on usability.

(ii) Metric of Components

This metric is used to evaluate the complexity of software architectural components as well as those aspects (characteristics) that are dependable on quality.

(iii) Metric of Specialized Object-oriented Design

This metric is used to measure classes, their communication aspects as well as the collaboration aspects.

Metric of Software Architecture

This metric is used to define the quality of software architectural design.

rics for Architectural Design

Architectural design metrics focus on program architectural characteristics and does not require any information about the internal working of a module. These metrics are measured using the following three software design complexity measures defined by Card and Glass.

Data Complexity

The complexity existing in the internal interface of a software unit is called data complexity and is given by,

$$D_{com}(n) = \frac{\text{var}(n)}{[f_{out}(n)+1]}$$

Where, D_{com} refers to data complexity,

'n' refers to a given module.

var(n) refers to number of variables that are supplied to and from module 'n'.

$f_{out}(n)$ refers to Fan-out values of software module.

Structural Complexity

Structural complexity is expressed as,

$$S_{com}(n) = f_{out}^2(n)$$

Where, $S_{com}(n)$ refers to structural complexity of a software unit 'n' and $f_{out}(n)$ refers to the Fan-out value of module 'n'. It is used for hierarchical architectures.

System Complexity

System complexity Syscom is the sum of structural and data complexity, i.e.,

$$Sys_{com}(n) = S_{com}(n) + D_{com}(n)$$

An increase in these complexities also increases the efforts required while integrating and testing various units of software and architectural complexity.

Moreover, the following metrics can be used to compare various program architectures. These were suggested by Fenton and are called morphology metrics.

1. **Size**

   It is the sum of nodes and arcs or edges in architecture tree given by,

   Size = Number of nodes + Number of arcs.

2. **Width**

   The highest count of nodes present in any level of the architecture tree.

3. **Depth**

   It refers to count of nodes accommodating on the longest vertical path starting from the root node till the leaf node.

4. **Arc-to-Node Ratio**

   The arc-to-node ratio is given by,

$$R = \frac{\text{Number of edges in a given architecture}}{\text{Number of nodes in the same architecture}}$$

The above values are useful in calculating the architectural connectivity density and coupling values. Now consider an example architecture,

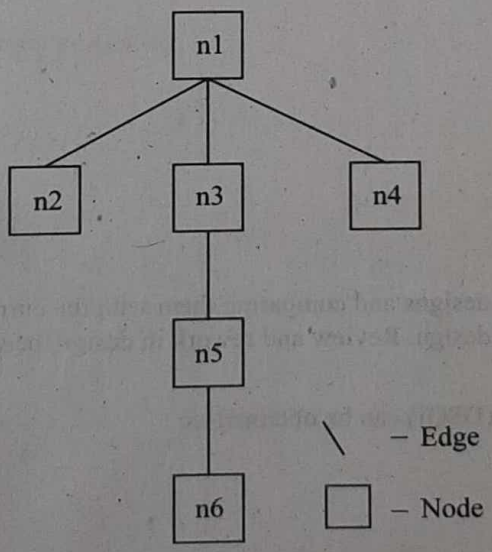


Figure: An Example Architecture

Size = 5 + 6 = 11 (Number of arcs + Number of nodes)

Width = 3 (Longest horizontal path) i.e., $n_1 - n_3 - n_5 - n_6$

Depth = 4 (Longest path from root node to leaf)

$(n_1 - n_3 - n_5 - n_6)$

Arc-to-node ratio = $\frac{5}{6} = 0.83$

US air force systems command developed the following values for measuring the Design Structure Quality Index (DSQI),

(i) Count of units in the architecture = X_1

(ii) Count of units which relies on source of data input for their functioning = X_2

(iii) Count of units which relies on prior processing for their functioning = X_3

(iv) Count of data objects and their attributes = X_4

(v) Count of unique items of database = X_5

(vi) Count of segments present in the database = X_6

(vii) Count of units having only one entry and one exit = X_7

After determining the above values, they can be used to derive the following intermediate values:

Program Structure (Y_1)

$$Y_1 = \begin{cases} 1, & \text{When the architectural design is obtained using a data-flow method or object-oriented method.} \\ 0, & \text{Otherwise} \end{cases}$$

Unit Independence (Y_2)

$$Y_2 = 1 - \frac{X_2}{X_1}$$

Unit Independent of Prior Processing (Y_3)

$$Y_3 = 1 - \frac{X_3}{X_1}$$

Size of Database (Y_4)

$$Y_4 = 1 - \frac{X_5}{X_4}$$

Database Compartmentalization (Y_5)

$$Y_5 = 1 - \frac{X_6}{X_4}$$

Unit Entry and Exit (Y_6)

$$Y_6 = 1 - \frac{X_7}{X_1}$$

Using the DSQI values for the prior designs and comparing them with the current one, gives an indication of how much review and rework is needed in the current design. Review and rework in design, becomes compulsory when DSQI for current design is less than average.

The Design Structure Quality Index (DSQI) can be obtained as,

$$\text{DSQI} = \sum_{i=1}^{6} w_i y_1$$

It can range between 0 and 1 and w_i is the relative weighting of the intermediate values. If the y_i's are not equally weighted then $\sum w_i = 1$ and if they are then, $w_i = 0.167$.

Need of Design Metrics

The main purpose of using metrics in design model is to measure the software quality and also the design complexity during the design phase. These metrics help the designer to use the mechanisms of object oriented paradigm (i.e., inheritance, encapsulation, polymorphism) in a convenient and understandable manner. This enhances the software quality and development procedure. It would be easier for the designer to select the best design model from multiple design alternatives based on the value computed using the design metrics.

Q59. A major information system has 1140 modules. There are 96 modules that perform control and coordination functions and 490 modules whose function depends on prior processing. The system processes approximately 220 data objects that each has an average of three attributes. There are 140 unique database items and 90 different database segments. Finally, 600 modules have single entry and exit points. Compute the DSQI of this system.

Answer :

Given that,

(i) Total number of modules in information system $(X_1) = 1140$

(ii) Number of modules that perform control and coordination functions $(X_2) = 96$

(iii) Number of modules that depend on prior processing $(X_3) = 490$

(iv) Number of database items $(X_4) = (220 \times 3) = 660$

(v) Number of unique database items $(X_5) = 140$

(vi) Number of different database segments $(X_6) = 90$

(vii) Number of modules having single entry and exit points $(X_7) = 600$

The values Y_1 through Y_6 can be computed, if the values X_1 to X_7 are known.

(i) Program structure $(Y_1) = 1$ [$\because$ The architectural design is for the information system dataflow]

(ii) Module independence,

$$(Y_2) = 1 - \frac{X_2}{X_1} = 1 - \frac{96}{1140}$$

$$= \frac{1044}{1140}$$

$$\boxed{Y_2 = 0.9158}$$

(iii) Modules which does not depend on prior processing,

$$(Y_3) = 1 - \frac{X_3}{X_1} = 1 - \frac{490}{1140}$$

$$= \frac{650}{1140}$$

$$= 0.570$$

(iv) Size of database,

$$(Y_4) = 1 - \frac{X_5}{X_4}$$

$$= 1 - \frac{140}{660} = \frac{26}{33}$$

$$= 0.788$$

(v) Database compartmentalization,

$$(Y_5) = 1 - \frac{X_6}{X_4} = 1 - \frac{90}{660}$$

$$= \frac{19}{22}$$

$$= 0.864$$

(vi) Module entrance or exit characteristic,

$$D_6 = 1 - \frac{X_7}{X_1}$$

$$= 1 - \frac{600}{1140}$$

$$= 1 - \frac{10}{19} = \frac{9}{19}$$

$$= 0.474$$

DSQI (Discrete structure Quality Index) is calculated as follows,

$$\text{DSQI} = \sum_{i=1}^{6} w_i D_i$$

Where W_i is the relative weighting and $\sum_{i=1}^{6} w_i = 1$ [$\because$ All D_i's are weighted unequally]

$\therefore$ DSQI = (1 + 0.9158 + 0.570 + 0.788 + 0.864 + 0.474) × 1

= 4.612

DSQI = 4.612

Q60. Discuss the nine distinct and measurable characteristics of Object Oriented design.

Answer :

Nine distinct measurable characteristics of an object oriented design are used. These are,

1. **Cohesion**

Cohesive nature of a piece of software is defined as the dependence among operations (in that software) to achieve a definite task. It is usually derived by closely analyzing the extent to which each property is possessed by a class which becomes the part of a problem domain.

2. **Volatility**

Volatility refers to the existence of possibility of alterations to be made to the existing software.

3. **Similarity**

If two or more classes have same purpose, function, behaviour or structure, then they are said to be similar and is given by the characteristic "similarity".

4. **Completeness**

When various points of view are considered for comparing the abstraction or design component, completeness for the abstraction is implemented. Completeness gives the degree of reusability of a design component.

5. **Complexity**

Complexity in terms of object oriented strategy, usually focuses on structural aspects of the classes i.e., the way each class is related to the other classes.

6. **Size**

To measure the size following elements are used,

(a) **Length**

The depth of an inheritance tree.

(b) **Volume**

Dynamic count of OO entities like operations, classes, etc.

(c) **Population**

Static count of OO entities.

(d) **Functionality**

It is usually a value, obtained indirectly by delivering a product in the hands of customers.

7. **Coupling**
The level of interaction between methods in classes is represented by coupling.

8. **Sufficiency**
The necessary requirements of a design component indicates its sufficiency.

9. **Primitiveness**
The amount/level of simplicity an operation has, gives its primitiveness.

Q61. Discuss the class-oriented metrics, the CK metrics suite.

Answer :

Class-oriented metrics have been proposed by Chidamber and Kemerer. Hence, these metrics are also referred to as CK-Metric suite. While deriving these metrics, the following issues are of prime focus,

(i) Class consisting of its attributes and operations

(ii) Class being parent or child

(iii) Each class having its collaboration with other external classes respectively.

Based on these issues, metrics for object oriented system (or CK metrics suite) can be given as follows.

Lack of Cohesion in Methods (LCOM)

In object oriented programming, it is a known fact that a class is a collection of objects. Usually each of these classes possesses certain instance variables, which are referred as as attributes. The number of methods of a given class accessing same attributes or instance variables is referred as LCOM. Assume that there is a class AA consisting of 3 objects. If out of these three objects, two accesses the same attributes, then LCOM value will be '2'. Similarly, in this class if no methods accesses same attributes, then LCOM value in this case remains zero. It has to be remembered that an increase in LCOM value increases the complexity of a class design. Hence, it is recommended to keep the LCOM value low to maintain high cohesion.

Number of Children (NOC)

Child class is a class, which derived its properties from any other class and the class whose properties are being derived is referred as parent or "root class". This concept is referred as inheritance. It has to be noted that, inheritance is implemented so as to attain code reusability. But as the NOC increases, testing increases. Hence, the "root class" may lose its abstraction property if the child classes are not managed accordingly.

Response for a Class (RFC)

For a class, a response set gives a collection of methods that must be executed as soon as an object of the same class receives a message. The count of methods in a response set gives the RFC. It is often valuable to keep the RFC value low, since, it decreases the testing efforts, test sequences and design complexity.

Depth of Inheritance Tree (DIT)

A tree showing the classes implementing inheritance and the classes that are inherited is called an inheritance tree. Number of levels of classes observed while moving from root class to lowest child class is called the Depth of Inheritance Tree (DIT). The advantage of large value of DIT is that large volume of code is reused.

The methods in the leaf classes will be potentially strong since they possess properties of their own as well as properties of all its parent classes. But increasing DIT also has disadvantages i.e., complexity of design increases, inability to easily summarize the behavioral aspects of a class etc.

Weighted Methods per Class (WMC)

It is a known fact that a class is a collection of object classes also containing methods. Consider a class with 'n' methods and complexities COM, .COM, COM_3.....COM_n respectively. In this regard, the complexity metrics should be chosen such that, it should possess a value 1.0 to be the minimum complexity value. The weighted methods for a class is the summation of all COs i.e., $\sum_{i=1}^{n} CO_i$. There, are certain reasons, which favour lowering of WMC to greater extent.

- As the number of methods and their complexities increases, the effort required during testing and implementing a class also increases.
- Increase of methods in a given class causes complex inheritance structures. This in turn, decreases the software reusability feature.

Coupling between Object Classes (CBO)

The number of collaborations present on the CRC index card of a class gives CBO for that class. This considers that completeness and consistency are properly assessed, while manually developing the CRC index card. Following are certain consequences that result on increasing the CBO value,

(i) It becomes difficult to make alterations for a given class, which in turn rises complexity during testing.

(ii) The reusability property of a given class gets lowered.

Hence, decreasing or lowering the value of CBO is a good idea.

Q62. Discuss the class oriented metrics-the MOOD metrics suite.

Answer :

Following are few MOOD Metrics.

Method Inheritance Factor (MIF)

The extent of utilization of inheritance by a given class architecture for the sake of it's methods and attributes is referred as method inheritance factor. Mathematically, MIF is given by,

$$\text{MIF} = \frac{\sum_{i=1}^{n} M_i(c_i)}{\sum_{i=1}^{n} M_a(c_i)}$$

Where,

$M_a(c_i)$ – Count of methods in c_i that can be invoked with regards to c_i

$M_d(c_i)$ – Count of methods declared in class c_i

$M_i(c_i)$ – Count of methods inherited but not overridden in c_i

n – Total number of classes in the architecture

c_i – i^{th} class in the architecture.

Also the value of $M_a(c_i)$ is equal to sum of number of methods in the class c_i and number of methods the class c_i inherits from other classes. Hence, the equation for $M_a(c_i)$ becomes,

$$\boxed{M_a(c_i) = M_d(c_i) + M_i(c_i)}$$

Coupling Factor (CF)

In context of MOOD metrics suite, coupling CF is defined as,

$$\text{CF} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} rel\_bet\_clnt\_Srvr\_class(c_i - c_j)}{(n^2 - n)}$$

Where,

$$\text{rel\_bet\_clnt\_Srvr\_class} = \begin{cases} 1, & \text{if the client class } c_c \text{ and server class } c_c \text{ are related and } c_c \neq c_s \\ 0, & \text{Otherwise} \end{cases}$$

The higher the value of CF, the higher is the software complexity and lower is the software reusability, maintainability and understandability.

Q63. Write short notes on,

(a) Component level design metrics

(b) Operation Oriented metrics.

Answer :

(a) Metrics Associated with Component Level Design

Component level design metrics are used as soon as a procedural design is available. These are,

1. **Cohesion Metrics**

These metrics define the extent of cohesive nature exhibited by a single module. They can be defined using the following five measures,

(i) **Data Tokens**

For a module m, data tokens are the variables defined in this module.

(ii) **Data Slice**

It refers to a piece of software in a module which relies on certain data values capable of changing the state of the module. A module and in turn a program can have many data slices.

(iii) **Glue Tokens**

It refers to set of data tokens being present in one or more data slices.

(iv) **Superglue Tokens**

It refers to set of data tokens, possessed by all data slices, present in a module.

(v) **Stickiness**

For a glue token g, stickiness is defined as the number of data slices bounded by g. Thus, the more data slices it binds the more is its stickiness.

2. **Coupling Metrics**

According to Dhama, coupling metrics can be defined using three types of metrics for global coupling, environmental coupling, control and dataflow coupling. These metrics are as follows,

(i) **Metrics for Global Coupling**

G_d = Count of global variables being used for data

G_c = Count of global variables being used for control.

(ii) **Metrics for Environmental Coupling**

W = Count of modules invoked by the module being considered.

R = Count of modules invoking the module being considered.

(iii) **Metrics for Data and Control Flow Coupling**

D_{in} = Count of input data arguments

D_{out} = Count of output data arguments

C_{in} = Count of input control arguments

C_{out} = Count of output control arguments.

Unit coupling indicator for a given unit of the software can be expressed as,

$$U_{cop} = \frac{P}{D_{in} + (x \times C_{in}) + D_{out} + (y \times C_{out}) + G_d + (z \times C) + W + R}$$

Where P is a proportionality constant and the values for x, y and z must be experimentally obtained. With an increase in U_{cop}, there is a decrease in the overall module coupling. To have the proportional effect, use the revised metric,

$$C = 1 - U_{cop}$$

3. **Complexity Metrics**

Most of the complexity metrics use flow-graphs as their basis. Another metric that adds to the complexity metrics is the cyclomatic complexity. With the help of complexity metrics, valuable information about maintainability and reliability of a software system can be easily predicted, the software design activity can be controlled due to the feedback provided by these metrics, etc.

(b) **Metrics associated with Operations or Operation Oriented Metrics**

Lorenz and Kidd proposed the following metrics in favour of operation oriented metrics. These metrics consider the average characteristics of methods or operations.

1. **Average Number of Parameters per Operation**

The average number of parameters required by each operation or method is given by the metric NP_{avg}. Its value must be kept as low as possible to reduce collaboration complexity between objects.

2. **Average Operation Size**

Operation size is usually measured using LOC, but this probably fails due to several shortcomings. Alternatively, operation size can be measured in terms of number of messages sent by a given operation. If this value increases, it indicates that the operations were not defined accurately. The average number of messages sent by an operation is given by OS_{avg}.

3. **Operation Complexity**

Operation complexity can be measured with any of the complexity metrics developed for a given software. It is denoted by OC. To limit an operation for performing a specific task, OC must be kept low.

Q64. Write notes on user interface design metrics.

Answer :

User interface design metrics are used as metrics in design model. There are two types of user interface design metrics,

1. (Layout Appropriateness) LA metric.
2. Cohesion Metric for UI screens.

1. **(Layout Appropriateness) LA Metric**

It helps in designing human/computer interfaces. It focuses on enabling the user to work in a GUI environment where different layout entities for example, menu, text, graphic icons, windows are available. Using these entities, user can perform different tasks like,

(i) Relocating from one layout entity to the other.

(ii) Determining a layout entity's absolute and relative positions.

(iii) Finding the frequency of usage of each entity.

(iv) Computing the cost required to relocate from one layout entity to the next.

Thus, achieving appropriateness of the interface using the LA metric.

2. **Cohesion Metric UI (User Interface)**

For a user interface, screen cohesion is a measure of amount of connectivity between various contents present on the screen.

Cohesion metric shows how to determine the cohesiveness of the screen in two ways,

(i) UI cohesion for a screen is low when multiple data objects are related with data (content) presented on the screen.

(ii) UI cohesion is high when only one data object is associated with content displayed on the screen.

4.2.4 Metrics for Source Code

Q65. Write short notes on metrics for source code.

Answer :

Metrics for Source Code

The metrics for source code are used to evaluate the source code and its characteristics like testability, maintainability. Such metrics comprise the following,

(i) Metric of Complexity

This metric is similar to the metric of component and is used to evaluate the logical complexity of source code.

(ii) Metric of Software Size

This metric is used to define the software size.

(iii) Metric of Halstead

This method is used to give distinct measures of a computer program.

Metrics for source code are liable only after the completion of coding or designing of software. Hence, in this regard the following are few important measurable factors which are taken into consideration.

n_1 = Number of different operators present in the code

n_2 = Number of different operands present in the code

T_1 = Number of times an operator appears in the code

T_2 = Number of times an operand appears in the code.

The above mentioned values are extremely useful in determining,

(i) The estimated number of faults existing in the hardware of a given system.

(ii) In calculation of total project development time.

(iii) In estimation of development effort.

Also these values are extensively used in generating expressions required for,

(i) Overall program length.

(ii) Actual and minimum volume (number of bits needed for specifying a program) of an algorithm.

(iii) Language level.

(iv) Program level.

The program volume 'V' is given by,

$$V = L \log_2(n_1 + n_2)$$

Where, L is the length of the program given by,

$$L = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

The value of 'V' in above equation may change with respect to the programming language. Finally, while dealing with metrics for source code, a new ratio 'V_r' referred as volume ratio is used which can be defined as,

$$V_r = \frac{\text{Number of bits accommodating a given program when it is in compact form}}{\text{Number of bits accommodating the same program when it is in normal form}}$$

And is represented using the following expression,

$$V_r = \frac{2}{n_1} \times \frac{n_2}{T_2}$$

4.2.5 Metrics for Testing

Q66. Discuss the metrics for testing.

Answer :

Dec.-11, Set-2, Q2(b)

Metrics for testing are derived by considering the implementation of process of testing. Hence, in this case, the testers usually resort to metrics laid for analysis, design and code phases. These remain fruitful for them in deriving test cases, so that they can be implemented during testing. Also the testers take into account the function-based and architectural design metrics. Here, function-based metrics are useful in many aspects like,

(i) Predicting the estimated values of the project by taking into consideration the values of previously completed project and

(ii) Estimating the efforts applied during entire phase of testing.

In the same way architectural design metrics are useful in,

(i) Determining the consequences which can be encountered during implementation of integration testing and

(ii) Determining the requirement of sophisticated testing software etc.

Halstead Metrics Applied to Testing

Halstead Program Level PL is given by,

$$PL = \frac{1}{\frac{n_1}{2} \times \frac{T_2}{n_2}}$$

Using the above equation, the equation for Halstead effort (e) can be obtained as,

$$e = \frac{V}{PL}$$

Where,

n_1 = Number of different operators present in the program

T_2 = Number of times an operand occurs in the program

n_2 = Number of different operands present in the program

V = The volume of information required to specify a program. It is measured in bits.

e = Halstead effort.

For a module 'm', the percentage of overall testing effort is given by,

$$P_{te(m)} = \frac{e(m)}{\sum_{j=1}^{n} e(j)}$$

Where,

n = Number of modules in the system

$P_{te(m)}$ = Percentage of testing effort required for module 'm'.

Metrics Associated with Object Oriented Testing

These metrics help in understanding the design quality and the testing effort that is necessary to test an OO system.

Fan IN

Fin in terms of object-oriented system is the measure of multiple inheritance. When Fin>1, it means that the class inherits from more than one root class, which should be avoided as much as possible.

Lack of Cohesion in Methods (LCOM)

LCOM should be kept as small as possible in order to minimize the number of states that need to be tested (checking the side-effects generated by the methods).

Number of Children (NOC) and Depth of Inheritance Tree (DIT)

For answer refer Unit-VI, Q61, Topics: Number of Children (NOC), Depth of Inheritance Tree (DIT).

Percent Public and Protected (PAP)

PAP is the percentage of class attributes that are declared either protected or public. High PAP results in high coupling between classes, which should be avoided. However, when PAP = 0 i.e., all attributes are declared private, the level of coupling reduces. Special tests must be designed to uncover side-effects like high coupling.

Public Access to Data Members (PAD)

PAD gives the number of methods or classes accessing attributes of other classes. The potential for side-effects between classes can increase with an increase in PAD. Thus it should be kept as low as possible.

4.2.6 Metrics of Maintenance

Q67. What is meant by software quality? Explain the metrics for maintenance. May-18(R15), Q9(b)

OR

What is the maintenance metric suggested by IEEE?

(Refer Only Topic: Metrics for Maintenance)

Model Paper-II, Q9(b)

Answer :

Software Quality

For answer refer Unit-IV, Q50, Topic: Software Quality.

Metrics for Maintenance

IEEE suggested a separate metric referred as Software Maturity Index (SMI) for maintenance. It measures a product's stability given by,

$$SMI = \frac{(X - (y_a + y_c + y_d))}{X}$$

Where,

X = Number of units in the current release of the software

y_a = Number of units added to the current release of software

y_c = Number of units changed in the current release of software

y_d = Number of units deleted in the current release of software.

When SMI ≈ 1.0, the product moves towards stability.

Advantages

1. SMI can give an estimated or mean time for producing a software.
2. SMI helps in developing the empirical models for maintenance effort.

UNIT 5

Metrics for Process and Products, Risk and Quality Managements

Syllabus

Metrics for Process and Products: Software Measurement, Metrics for Software Quality.

Risk Management: Reactive Vs Proactive Risk Strategies, Software Risks, Risk Identification, Risk Projection, Risk Refinement, RMMM, RMMM Plan.

Quality Management: Quality Concepts, Software Quality Assurance, Software Reviews, Formal Technical Reviews, Statistical Software Quality Assurance, Software Reliability, The ISO 9000 Quality Standards.

PART-A SHORT QUESTIONS WITH SOLUTIONS

Q1. Define metrics.

Answer :

Model Paper-III, Q1(i)

A metric is defined as a quantitative measure of having a certain attribute in a system/process/component i.e., the degree upto which the process may constitute a specified attribute. Typically, every software is dependent on every measure such as the average number of errors detected are dependent on the unit test/review of the system component.

A software metric can do the following,

1. Aid the design such that an efficient testing can be done.
2. Indicate that the source code and design are associated with the complexity measures.
3. Aid to derive the measures and analysis of design models.

Q2. What are software quality metrics?

Answer :

Dec.-19(R16), Q1(h)

A software metric refers to any type of measurement to improve the process of developing software as well as to understand all aspects of the management of that software. Software metrics are used in the overall development of life cycle from initiation of product to monitoring the reliability of the end product. It provides the techniques which are helpful for monitoring and controlling the progress of the software development. It must be simple, consistent and persuasive. It must be independent of the programming language and must give positive feedback to the software developers. Software metrics are given out by several measures. Therefore, the metrics vary from one software to another.

Q3. Explain the merits and demerits of use-case-oriented metrics.

Answer :

Merits of Use-case-oriented Metrics

- Usecase can be used as a normalization measure similar to LOC or FP.
- Usecase provides information about the functions in the software.
- Usecase represent features of the applications.
- The count of usecases is proportional to,
  (a) The size of the software in LOC and
  (b) The count of test cases to be developed which is capable of testing the entire software.

Demerits of Use-case-oriented Metrics

- Since usecases are created at different levels of abstractions, they do not have any standard size.
- Without any standard measure the application of the usecase as a normalization measure is often suspected.

Q4. What guidelines should be applied when we collect software metrics?

Answer : Model Paper-I, Q1(i)

Guidelines for collecting software metrics,

1. Interpret metrics data carefully.
2. Do not ignore the metrics data that show a problem area because they are the way to improve the process.
3. The teams/individuals who collect measures and metrics should be given regular feedback.
4. Do not threaten teams/individuals using metrics.
5. Fix appropriate goals and metrics. This can be done by working with teams and practitioners.
6. Metrics should not be used to judge individuals.
7. Do not focus only on one metric isolating all other important metrics.

Q5. What are the metrics used for software maintenance?

Answer : Nov.-15(R13), Q1(h)

The following are the metrics used for software maintenance,

(a) Direct Metrics

Direct metrics are the software metrics that can be measured directly without using any functionality, complexity.

(b) Indirect Metrics

Indirect metrics are the metrics that are measured indirectly. When compared to direct metrics, indirect metrics are more difficult to collect.

(c) Public Metrics

Public metrics are software process metrics that usually consists of information, which were private to software project teams and to individual member of project team.

(d) Private Metrics

Private metrics are the software process metrics (data) that consists of information, which is private to individual software engineers.

Q6. What are software risks? Explain various types of software risks. Dec.-19(R16), Q1(i)

OR

Give the different categories of risks.

Answer : May/June-19(R16), Q1(i)

Software Risks

In general terms, software risks refer to the probable threats expected to be faced by a given software.

Types of Software Risks

Different types of software risks are,

1. Business Risks
2. Project Risks
3. Technical Risks
4. Predictable Risks
5. Unpredictable Risks
6. Known Risks.

Q7. Define risk management.

Answer : Model Paper-III, Q1(j)

Risk management is a process of identifying and managing the potential risks that might affect the project schedule or the quality of a software. The software tools for risk management are used to assist the project team in order to identify, access and minimize the risks from the software project.

Q8. Differentiate between reactive risk and proactive risk strategies. Nov./Dec.-18(R16), Q1(i)

OR

Distinguish between reactive and proactive risk management.

Answer : May-18(R15), Q1(i)

Reactive Vs Proactive Risk Strategies

Reactive Vs proactive risk strategies usually refers to two different approaches of risk management. Following are the details of describing each of these approaches.

1. Reactive Risk Strategy

It is one of the older approaches in risk management. In this case, usually the resources which can be utilized during the occurrence of risks are maintained separately. When the risks are encountered, these resources are used to nullify them. If the risks are certainly more strong such that they are intolerable even while applying these resources, then this project is moved to *crisis management team*. Here, the projects are strived rigorously to overcome the risks.

2. Proactive Risk Strategy

Consider the following thought "prevention is better than cure". Proactive risk strategy is one of the risk management strategies, which consider the above thought to be the fundamental hypothesis. In this case, a document is prepared by stating all the probable risks, which are going to be encountered throughout the project, even before the project begins.

Q9. Explain about product size risks.

Answer :

Product size risks are the risks associated with the overall size of the software that is to be developed or modified. The probability for the occurrence of product size risk is 30% and its impact on the organization is castastrophic that results in significant degradation of technical performance.

The following are certain risk item issues considered with respect to product size risks,

1. What is the estimated product size as well as the level of confidence with the estimated size?
2. What is the size of database allocated to product for both creating and using it?
3. What is the total count of the users using the product?
4. What are the modifications to be performed on the requirements of a product?

Q10. Distinguish between generic risks and product specific risks.

Answer :

	Generic Risks		Product Specific Risks
1.	Generic risks are essential threat to every software project.	1.	Product specific risks are essential threat that is specific to software project.
2.	Generic risks can be identified easily.	2.	Product specific risks can be identified by examining of project plan and software scope.
3.	Generic risks occurs due to uncertainties involved in accessing or estimating various inputs to the software process.	3.	Product specific risks occurs due to conditions and constraints about resources, customers and lack of organization support.
4.	Generic risks are simple to understand.	4.	Product specific risks are difficult to understand.

Q11. Define risk refinement.

Answer :

(Model Paper-I, Q1(j) | Nov./Dec.-17(R15), Q1(i))

Risk refinement is a process of refining the already stated risk definition and representing it in a more detailed form. The process of refinement is initiated when project planning process is thoroughly analyzed. This refinement has been proved useful to mitigate, monitor and manage these risks in a more convenient manner. The process can be carried out by following a standard notation of risk representation referred to as Condition Transition Consequence (CTC). This notation states that, "If a condition is specified, then there exist a concern that may possibly result in a consequence".

Q12. Write short note on RMMM.

Answer :

May-18(R15), Q1(j)

Any risk analysis activity should include the following steps,

(i) Risk avoidance

(ii) Risk monitoring and

(iii) Risk management as well as contingency planning respectively.

It is one of the famous assumptions that "prevention is better than cure". Hence, the software team, always take an initiative before the risk approaches can resort to risk avoidance strategy. For effectively implementing the risk avoidance strategy, one has to depend on suitable plans referred as "*risk mitigation plans*" (the word "mitigation" refers to the process through which the occurrence of risks can be eliminated).

Consider an example, wherein one can analyze the process of risk mitigation, monitoring and management. Here, we consider the risk as "high staff turnover".

Q13. Define maintenance. What are the types of software maintenance?

Answer :

Maintenance

March-17(R13), Q1(j)

Software maintenance refers to the changes which are performed once the software is delivered to users. Maintenance activity is compulsory for each type of product and cannot be avoided. Generally, products requires maintenance activity due to its heavy usage. But, software need maintenance in order to improve features, correct errors. It is an essential activity adopted by large number of organization because of the misuse of hardware.

Types of Software Maintenance

There are three types of software maintenance.

(i) Corrective Maintenance

It refers to the maintenance type which corrects the errors when the system is in use.

(ii) Adaptive Maintenance

It refers to the maintenance type which need to adapt new platforms, new operating system, new hardware and new software as required by the customer or on demand.

(iii) Perfective

It refers to the maintenance type which supports distinct features, modify functionality of the system on customer demand to improve the systems performance.

Q14. When will be the formal technical reviews are conducted? And what are its objectives?

Answer :

FTR is conducted when software engineers wants to review a product at any phase of the development process. It involves software engineer and a review team.

The main objectives of FTR are as follows,

(a) To check whether software meets the requirements.

(b) To check the uniformity of the software process.

(c) To verify that software representation is as per the predefined standards.

(d) To discover errors of the software with respect to its operation, logic and implementation.

(e) To make the projects in a controllable way.

Q15. What is meant by software review?

Answer : (Model Paper-II, Q1(i) | May/June-19(R16), Q1(j))

Software review is the process of inspecting the software at different points with an intention of pointing out defects and errors normally occurring in the software, hence, improving its quality. Basically, there are many types of software reviews each having its own value or importance.

Q16. What is the importance of software reviews?

Answer : Nov./Dec.-16(R13), Q1(j)

The importance of software review are as follows,

(i) The correction of errors that are found initially in the process is cost-effective.

(ii) Errors are amplified along with progression of the process. Hence, the minor errors that are not corrected initially can be amplified into a big set of errors later in the project.

(iii) Reviews reduce the amount of rework that is required late in the project. By this, lot of time is saved.

Q17. What is software reliability and how this parameter helps in managing software quality?

Nov./Dec.-18(R16), Q1(j)

OR

Define software reliability. (Nov./Dec.-17(R15), Q1(j) | Nov./Dec.-16(R13), Q1(i))

OR

What is software reliability? Define.

Answer : Nov.-15(R13), Q1(i)

Software Reliability

Software reliability refers to the probability of software running with out any failure. The software failures mainly occur due to the design pattern followed or due to implementation problems but the software doesn't wear out like the hardware components. Software availability refers to probability that a program or software is available.

The Role of Software Reliability in Managing Software Quality

There are six quality characteristics that are defined by ISO 9126. One among them is software reliability which plays a major role when compared to other characteristics. A software reliability program requires the development of a balanced set of user quality objectives and recognizes the intermediate quality objectives that helps in obtaining the user quality objectives. As a consequence, it can be concluded that the software reliability is depended on the high quality software.

Q18. Explain the measures of software reliability and availability.

Dec.-19(R16), Q1(j)

Answer :

Measures of software reliability and availability are listed in the following table,

Metric		Meaning	Example Systems
(i)	POFOD (Probability of Failure on Demand)	This is a measure of the likelihood that the system will fail when a service request is made. For example, a POFOD of 0.001 means that 1 out of 1000 service requests may result in failure.	Safety-critical and non-stop systems, such as hardware control systems.
(ii)	ROCOF (Rate of Failure Occurrence)	This is a measure of the frequency of occurrence with which unexpected behaviour is likely to occur. For example, ROCOF of 2.100 means that 2 failures are likely to occur in each 100 operational time units. This metric is sometimes called the failure intensity.	Operating systems, transaction processing systems.
(iii)	MTTF (Mean Time to Failure)	This is a measure of the time between observed system failures. For example, an MTTF of 500 means that 1 failure can be expected for every 500 time units. If the system is not being changed, it is the reciprocal of the ROCOF. Mean time failure is the measure of software reliability which is expressed as MTBF = MTTF + MTTR where MTTF is Mean-Time-to-Failure MTTR is Mean-Time-to-Repair	Systems with long transactions such as CAD systems. MTTF must be greater than the transaction time.
(iv)	AVAIL (Availability)	This is a measure of how likely the system is to be available for use. For example, an availability of 0.998 means that in every 1000 time units, the system is likely to be available for 998 time units. Availability of software is measured with the following expression. $\text{Availability} = \left(\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}\right) \times 100\%$	Continuously running systems such as telephone switching systems.

Table: Software Reliability and Availability Measures

Q19. Can a program be correct and still not exhibit good quality? Explain.

Answer : (Model Paper-II, Q1(j) | Nov.-15(R13), Q1(j))

Yes, it is possible that in spite of correct program, it may not exhibit good quality.

A program may support the quality of conformance i.e., confirming to all functional and performance requirements writhing the scheduled time. Degree of quality of conformance is high when the resulting system meet the requirements and performance criteria.

If a program is designed accurately ad generates correct output, still there is possibility of hidden errors which causes the failure of a system. A system fails due to the following hidden errors,

(i) Poor quality control

(ii) Inadequate architectural design

(iii) Performance specifications like expected volume of the data, traffic congestion, transactions are insufficient in nature.

PART-B ESSAY QUESTIONS WITH SOLUTIONS

5.1 METRICS FOR PROCESS AND PRODUCTS

5.1.1 Software Measurement

Q20. What is meant by software process and project metrics?

Answer : Model Paper-III, Q10(a)

Software Measurement

Measuring of software process and project metrics allows the software engineers to analyze and develop the software process, which improves the system efficiency. The projects that are developed by the software process are known as framework. During the process, the quality and productivity of data is analyzed, compared and assessed to determine the improvements of quality and productivity. Metrics are used to improve the development of software process.

Software Process Metrics

A software process is a set of activities which together produces a software product. These activities are specification, development, validation and evaluation carried out by the software engineers.

The process maturity of a software can be increased by considering software process metrics which has the following conditions,

(i) Use the data which can be easily understandable.

(ii) The output produced by the software metric must be displayed without any failure.

(iii) Software metrics may not be initialized by the individual.

(iv) Metric must be carried out by the individual.

(v) Improper data called as negative data helps in improving the process.

(vi) Software metric handles multiple data.

Project Metrics

Project metrics are used for middle level or tactical purposes. A project manager can handle various responsibilities by using project metrics. The main principle of using project metrics is in the estimation of various activities of the software project. Project metrics are used by a project manager to control the project cost, time and effort. These activities can be carried out with the help of management of skills, technology and development, customer relations and software solution design. Project metrics help project manager, to plan and execute the development of life cycle activities, which is common for all projects.

Generally, project metrics are obtained from the previous projects that are used to estimate the time and effort of current software work. As project proceeds, the measures of effort and time are compared to the actual estimations. Therefore, a project manager applies these data in order to monitor and control the progress of software process.

The principle objectives of project metrics are,

- To know the status of current projects.
- To detect potential risks.
- To compute the project team's capability for controlling the quality of software products.
- To manage workflow or achieving the goals.
- To minimize the development of project schedule by solving the potential risk issues and avoiding the delays.
- To improve the quality of software product on current basis.

Q21. Discuss about software tools for project and process metrics.

Answer :

Software Tools for Project and Process Metrics

The software tools designed for project and process metrics helps in data collection, estimation and documentation. Each and every tool for project and process metrics differ in its own application but uses the same mechanism. The software tools are discussed below.

1. **Function Point Workbench**

It was developed by Charismatek.

Features

(i) It is one of the leading software tools based on the function point analysis strategy. It is used to measure and estimate the software in easier manner. Besides this, it also provides software definition, outsourcing benchmarking, project control and cost related transactions etc.

(ii) It is modeled specifically not only to use within the software development process, but also for corporate management and reporting.

(iii) Function point workbench acts as multiserver for the necessities of business people, but the necessities of function point counters are directly met by using this tool. Function point counters assist in determining the functions and measures as per the international standards, provide concurrent data sharing, activate the counter in order to reuse all or specific components and assist in building the templates for further use etc.

2. **PSM Insight Tool**

It was developed to support *software and systems measurement*.

Features

(i) The main objective of this tool is to create and analyze the project's database.

(ii) It is a windows-based application tool, that permits the user a high flexibility level in managing the data which includes modification, browsing, complexed graphing etc., in order to produce an effective issue driven measurement program.

(iii) This tool is designed in accordance with the industry standards developed for Open DataBase Connectivity (ODBC), thereby permitting the tool for creating and retrieving different databases with different formats.

3. **Metric Center**

Metric center was developed by '*distributive software*'.

Features

(i) This tool supports each and every aspect of an organizational measurement process like automated data collection, analysis, report generation, formatting of charts etc., for an effective decision making.

(ii) Distributive data drill management software provides a platform for project managers during the process of developing a systematic plan and also assists them throughout the project life cycle.

4. **SLIM Tool Set (Software Life Cycle Management)**

It was developed by 'QSM software'.

Features

(i) It is a dual set of metrics and estimation tools.

(ii) It is designed for better decision making carried out at each stage of software life cycle. It can be estimating, tracking and control, analysis of software metrics for the completed projects.

5. **Tychometrics**

It is based on predicate logic's patented measurement modeling technology.

Features

(i) This tool is designed for automated data collection across the world through internet, managing of metrics collection, report generation etc.

(ii) It resolves the problems in an effective manner that are associated with the measurement.

(iii) It ensures high degree of consistency, accuracy and detailed description over the entire organization. It also offers assistance of measurement modeling technology called "*universal data measurement analysis and control system*".

Q22. Explain the size-oriented metrics with an example.

Answer :

Software Size-oriented Metrics

In software size-oriented metrics, size of the software (i.e.,) number of lines of code is of primary focus. This can be analyzed deeply by considering certain facts given below.

The following table shows the maintenance of information by any software development organization.

Project name or title	Total number of lines of code	Amount in Rs.	Total efforts applied (person-month/year)	No.of people in development team	Bugs reported	Defects	Documentation
Web auction	2000	8000	15	4	75	6	200
Web portal	15000	12000	24	8	104	16	1000
Airway reservation	28000	56000	66	12	200	8	1200

Table: Information Maintained by a Given Software Development Organization

The above tabular information is based on the software which was already developed by a given organization. The table initially begins with the project title or its name, followed by the total number of lines of code, amount of effort applied, number of people involved in the development team, bugs reported, defects and documentation respectively. Details on each project is also quoted in the fields.

Now, the size-oriented metrics strive to make the number of lines of code to attain the normalized value. Based on this analogy, various other metrics can be derived as follows,

- Defects per thousand lines of code.
- Amount charged per thousand lines of code.
- Bugs encountered per thousand lines of code.
- Number of pages of documentation per thousand lines of code.
- Efforts applied per thousand lines of code.
- Bugs encountered per person per month etc.

Though the above mentioned artifacts looked fruitful in considering the size-oriented metrics to be the best source for measuring the software process, it is not a widely accepted analogy. The organizations favoring number of lines of code as key in measuring the software process, supports this analogy by referring it to be the major artifact. This artifact is used to derive almost all the essential measures. There are many software estimation models in which the number of lines of code forms the basic aspect. Also there exists many already written documents favoring it. But the organizations which do not favour this analogy, condemns the fact by saying that there are many languages available, which are capable of providing large outputs in only few lines of code. Hence, no proper conclusion is resulted.

Q23. Differentiate between function-oriented and size-oriented metrics.

Dec.-11, Set-2, Q4(a)

Answer :

	Size-oriented Metrics		Function-oriented Metrics
1.	Size-oriented metrics are direct measure of software.	1.	Function-oriented metrics are indirect measure of software.
2.	Size-oriented metrics are the attempt to measure the size of the software.	2.	Function-oriented metrics are the attempt to measure the functionality of software.
3.	Size-oriented metrics use lines of code as a normalization value.	3.	Function-oriented metrics use measure of functionality as the normalization value.
4.	Size-oriented metrics include effort (time), money spent, KLOC(1000s lines of code), pages of documentation created, errors, people on the project etc.	4.	Function-oriented metrics include number of external Inputs [EIs], number of External Outputs (EOs), Number of External Inquires (EQs), Number of Internal Logical Files (ILFs) and Number of External Interface Files (EIFs).
5.	It focuses on the lines of code.	5.	It focuses on the function points.
6.	It is programming language dependent.	6.	It is independent of programming language used.

Q24. Write short notes on,

(i) Object-oriented metrics

(ii) Usecase-oriented metrics.

Answer :

(i) Object-oriented Metrics

Following are the metrics favoring object-oriented projects.

(a) Count of Subsystems

Collection of certain set of classes favoring a definite function is referred as a *subsystem*. In a given project, there can be any number of subsystems. Hence, as the subsystems are identified, the software development team can easily schedule their work and can easily distribute these subsystems among themselves.

(b) Count of Key Classes

Key classes are the most essential metrics that are identified at the initial stages of the project development. Their identification plays a vital role during the calculation of the total efforts to be applied during the entire project development. It is also used in the estimation of the amount of the software reuse. They form the major entities of problem domain.

(c) Count of Support Classes

Support classes are also the major requirements of the project, but they do not form the major entities of problem domain. Their presence is only to support the key classes. The purpose of their identification is same as that of the key classes.

(d) Count of Scenario Scripts

As the name suggests, scenario scripts usually consist of steps that a given end user performs while interacting with the application being developed. During script analysis, software size can be easily determined. It is mostly used in developing testing strategies.

(e) Average Count of Support Classes Per Key Class

The key classes are identified in the initial stages of the project development process. While, the supporting classes are developed the project proceeds. In this case, if average count of support classes per key class is known, then it becomes very easy to estimate the total value of support classes per key class. In this regard, it has been estimated that for the projects involving GUI components, there are about two to three times greater number of supporting classes for each key class. The value decreases in case of projects with non-GUI supported components.

(iii) **Usecase-oriented Metrics**

(a) Usecase can be used as a normalization measure similar to LOC or FP.

(b) Usecases provide information of the functions as well as the features of the application to be developed.

(c) They are independent of the programming languages.

(d) The count of usecases is proportional to,

- The size of the application in LOC and
- The count of test cases to be developed which is capable of testing the entire application.

(e) There is no standard size for the usecases as they are created at different levels of abstraction. Due to this reason, they are considered as the normalization measures.

Q25. The software used to control a photocopier requires 32,000 lines of C and 4200 lines of small talk. Estimate the number of function points for the software inside the copier.

Answer :

The languages which are used in programming gives the difference between the function points and number of lines of codes (LOC). These languages will define the functionalities provided by the data of the system.

The table represents predefined number of lines of codes along with its function points which varies from one language to another language.

Languages	Function Points			
	Low	Mediam	High	Average
C	33	109	704	62
C++	29	53	178	66
Small talk	10	19	55	26

The table indicates that one line of code of C++ will be equal to the 2.4 times the function points of line of code of C. Moreover, one LOC of small talk will be equal to the 4 times the function points of one LOC of other programming languages such as C, Ada etc.

For example, if there is a software to control the photocopier, which needs 32,000 lines of code of C and 4200 lines of code of small talk then the number of function points used inside the copier is given as follows,

We know that, for one LOC of C, it will be 4 times the function points of LOC of small talk. As a result,

$$4200 \text{ (LOC)} = 4 \times 32{,}000 \text{ (f)}$$

$$4200 \text{ (LOC)} = 128{,}000 \text{ (f)}$$

$$\text{LOC} = \frac{128{,}000}{4200} \text{ (f)}$$

$$= 30.47 \text{ (f)}$$

Therefore, for one LOC of small talk, 30 times the function points of 'C' are required.

$$\therefore \text{ Function points} = 4200 \times 30 \text{ (f)} + 30 \text{ (f)}$$

$$= 126000 \text{ (f)} + 30 \text{ (f)}$$

$$= 126030 \text{ (f)}$$

Therefore, the total number of function points required for the photocopier is 126030 (f).

5.1.2 Metrics for Software Quality

Q26. Explain the metrics for Software Quality.

(May/June-19(R16), Q9(a) | Nov./Dec.-17(R15), Q9(a))

OR

What is software metrics? Explain the importance of it in detail.

Answer : (Model Paper-III, Q10(b) | Dec.-11, Set-3, Q8(a))

Software Metrics

A software metric refers to any type of measurement to improve the process of developing software as well as to understand all aspects of the management of that software. Software metrics are used in the overall development of life cycle from initiation of product to monitoring the reliability of the end product. It provides the techniques which are helpful for monitoring and controlling the progress of the software development. It must be simple, consistent and persuasive. It must be independent of the programming language and must give positive feedback to the software developers. Software metrics are given out by several measures. Therefore, the metrics vary from one software to another.

Importance of Software Metrics

The importance of software metrics are listed below,

1. It is possible to measure various kinds of software in terms of its quantitative analysis. The data metrics that were collected during the software development gave rise to the standards for planning and estimating the software resources, size, cost, efforts and time for the software development.
2. Software metrics are used in the overall development of life cycle from initiation of product to monitoring the reliability of the end product.

   They provide the techniques which are helpful for monitoring and controlling the progress of the software development.
3. Metrics can be used to indicate the basic attributes.
4. The development and validation of the software process models are based on the quantitative analysis. It can be used for determination of software quality and productivity.
5. The characteristics of a software product or process are quantified by simplifying the understandability of various system portions.
6. Metrics are useful to analyse and take decisions for a method and a product in order to accept, refuse or develop a software.
7. Metrics are useful in setting goals for new processes, methods or products. It also helps to identify the requirements in the development process.

Q27. What are the metrics used for software maintenance? Discuss.

Nov./Dec.-16(R13), Q9(a)

Answer :

The following are the metrics used for software maintenance,

(i) Direct metrics

(ii) Indirect metrics

(iii) Public metrics

(iv) Private metrics.

(i) Direct Metrics

Direct metrics are the software metrics that can be measured directly without using any functional complexity. Line of code, execution speed, defect rate reported over a certain period of time interval are some of the direct metrics.

(ii) Indirect Metrics

Indirect metrics are the metrics that are measured indirectly. When compared to direct metrics, indirect metrics are more difficult to collect. These metrics are usually associated with an object feature that is to be measured. Function-oriented metrics can be considered as indirect software metrics.

(iii) Public Metrics

Public metrics are software process metrics that usually consist of information, which were private to software project teams and to individual member of project team. These metrics help in improving the organisation level of process maturity. The information is collected and evaluated so as to identify the indicators that are capable of enhancing the overall performance of the organisation process. Some of the public metrics used in an organisation are,

(i) Number of defects made at project level

(ii) Amount of time and effort consumed.

(iv) Private Metrics

Private metrics are the software process metrics that consists of information, which is private to individual software engineers. These metrics are collected on an individual basis and are used as an indicator for individual only. If individual software engineers desire to improve their performance level, then in such situation, private metrics can be used as essential drivers for achieving the required level of performance. Some of the private metrics are,

(i) Number of defects made by individual

(ii) Number of defects made by software component

(iii) Number of errors found during the development process.

In addition to these process metrics, there are some process metrics that are private to software project team but are public to individual members of the project team. These metrics are evaluated by respective software team so as to identify the indicators that can be used for improving the performance level of entire project team. Some of these metrics are,

(i) Number of defects reported while implementing software functions

(ii) Number of LOC and FP available for each component or function.

Q28. Elaborate on Web App project metrics in detail.

Dec.-11, Set-4, Q4

Answer :

Web App Project Metrics

Web apps project metrics is designed to deliver suitability, accuracy, interoperability compliance and security content to the end users.

Measures and metrics which were used for older software projects have become difficult to be changed to web apps. But their database can be created to allow access to quality measures and internal productivity.

Some of the measures that can be collected are,

(i) **Static Web Pages**

Static web pages display the same information for all the users. Users have no control of the web page. This feature has less complexity and requires less efforts to develop static web page.

(ii) **Dynamic Web Pages**

Dynamic web page content changes with times. For example, news content, financial applications and search engines. It provides overall size of the application and it has higher complexity and requires more effort to develop dynamic web pages.

(iii) **Internal Page Links**

An internal link allows to link to another section on the some other web page in the web app. As the rate of page links increases, efforts on navigational design and construction also increases.

(iv) **Persistent Data Object**

When more number of persistent data objects are accessed in the web pages, the complexity increases. As the rate of data objects increases, effort to implement it also increases.

(v) **Interfacing with External Systems**

Web apps usually interface with business applications. As the requirement for interfacing with external system increases, effort and complexity also increases.

(vi) **Static Content Object**

Static content is published to regular files on the server. Static web pages contain multiple objects which will be displayed on a single web page. Static content object consist of graphical, video and animation which are built within the web app.

(vii) **Dynamic Content Object**

Dynamic content objects are made by the users of the page. The page can be viewed by the individual user only at the moment and will not be displayed to any other user. Many dynamic content objects will be displayed in a single web page.

(viii) **Executable Functions**

Executable functions provide some computational services like applet script. The effort to develop and model increases as executable functions increase.

Q29. Discuss four useful indicators for software quality.

Answer :

Useful Indicators for Software Quality

The four useful indicators for software quality are,

1. Usability
2. Integrity
3. Maintainability and
4. Correctness.

1. Usability

It is always a proposed fact that, a given application perishes for certain duration only when it facilitates user-friendly nature during its operation. The negative aspect of this saying is that, the given application is prone to error, if it is complex during its usage. Hence, high quality project requires that its usability standards should be given high preference.

2. Integrity

As the technology is improving, maintenance of the system's integrity is becoming a tedious task. Integrity is usually the feature of a given software which focuses on resisting itself from several attacks made by hackers and other persons (intentionally or unintentionally) with a desire of exploiting its resources (i.e., programs, data and document). Hence, maintaining integrity of a given software always forms the major issue while determining the quality of a given software. The two most important aspects of the software i.e., threat and security must be considered here. *Threat* is a term used to describe the probability of occurrence of an attack on the given software, in a specific period of time. On the other hand, the *security* defines inherent nature of software in resisting the given attack.

Now, integrity of a given software can be expressed using the following expression (i.e.,)

$$I = \Sigma[1 - \{T \times (1 - S)\}]$$

Where,

I refers to integrity

T refers to threat

S refers to security.

In the above expression, if the values of T and S are substituted and the resultant 'I' value obtained is 0.98 or 0.99. So, it is said that the given software is inherently strong in resisting the attacks.

3. Maintainability

When compared to any of the software quality factors, maintainability is the most complex issue. Maintainability of a given software can be defined as the feature of software,

- Which is capable of adjusting with any given environment
- Which is capable of upgrading on demand.
- Which is capable of correcting itself on the occurrences of errors etc.

Hence, while considering the above facts, one can realize that there are no direct metrics to measure the maintainability of a given software. Hence, an indirect measure is generally adopted, which is nothing but the "mean-time-to-change" (MTTC). It is simply the duration in which the request for a change is analyzed, designed, deployed, tested and referred to the users of a system.

4. Correctness

Correctness is a feature of given software, that describes the extent to which software functions as per the expectations of the user. The measure of correctness is the defects per thousand lines of code. Here, defects are nothing but the instance of software functions, that fail to satisfy the user's requirements. Also, the defects of software are listed only when it is delivered to the user. Always, the software industries maintain a list of defects encountered per year.

Q30. Explain the role of software metrics in maintaining quality.

Answer : May/June-12, Set-4, Q4(a)

Role of Metrics in Software Quality

A high-quality system, product or an application is the main goal of software engineering. It can be achieved by employing the effective techniques and tools with a matured software process. The goodness of the quality of a system depends on the goodness of the following,

1. The requirements that showcase the problem.
2. The design model that provides the solution to the showcased problems.
3. The code that brings out an executable program.
4. Test cases that uncover the errors.

Thus, the quality of the requirements, design model, source code and test cases should be measured. Software engineers must measure the software to know whether high quality is achieved or not. Product metrics have to be applied for the evaluation of software quality as the project improves.

In addition to the collection of quality measures, importance must be given to the errors and problematic areas occuring during the project. The metrics which are the outcome of these measures indicate the improvement in software quality assurance and control activities.

Although many measures for software quality have come up, software engineering has provided four useful indicators for it.

Measures for Software Quality

For answer refer Unit-V, Q29.

Q31. What is meant by Defect Removal Efficiency (DRE)? How it can be assessed?

Answer :

Defect Removal Efficiency (DRE)

DRE is a metric which is useful when considering the project as a whole and also while performing the simple processes of the entire project.

It can be assessed in two levels,

1. DRE at the project level and
2. DRE at process level.

1. DRE at the Project Level

In this case, DRE is given by the following ratio,

$$DRE = \frac{Error}{Error + Defects}$$

Here, "*error*" refers to the bugs encountered during the software testing process whereas '*defects*' are the casualties reported by a given end user, once the software is delivered. It has to be noted that, the ideal value of DRE indicates that the software is completely error free. But, generally this is not the case. The value of DRE can be greater than '0' and less than '1'. Moreover, the value of DRE can reach closer to '1'. This happens when the value of 'error' increases significantly in the given ratio. This enables the software engineers to choose an easy error correcting process rather than the cumbersome defect removal methods.

2. DRE at Process Level

DRE when used along with a single entity of an entire project, improves team skills in locating errors. This makes the team to directly reach to next module of the same project. Hence, in this case, the DRE is given as follows.

$$DRE = \frac{Error_i}{Error_i + Error_{i+1}}$$

Here, '$error_i$' are the errors encountered in module 'i' whereas '$error_{i+1}$' refers to the errors in module $i+1$. For example, if $i = 3$ the '$error_3$' refers to the errors encountered in the module 3 and '$error_{3+1}$' (i.e.,) error4 are errors encountered in software module 4.

5.2 RISK MANAGEMENT

Q32. Define risk management. Discuss about software tools for risk management.

Answer : Model Paper-II, Q10(a)

Risk Management

Risk management is a process of identifying and managing the potential risks that might affect the project schedule or the quality of a software. The software tools for risk management are used to assist the project team in order to identify, access and minimize the risks from the software project.

Software Tools for Risk Management

There are several software tools for risk management. They are as follows,

(i) **Risk Radar Tool**

This software tool is developed by SPMN with the purpose of assisting the project managers for identifying and managing the potential project risks.

(ii) **Risk Tool**

This software tool is developed by C/S solutions with the cooperation of Microsoft project for identifying and managing cost and schedule problems.

(iii) **Risk Track Tool**

This software tool is developed by RST with the purpose of tracking the potential risks. It is used to identify, analyze, report and manage the risks throughout the software project.

(iv) **Riskman Tool**

This software tool is developed at Arizona state university in the expert system for identifying the potential risks that are related to the software project.

(v) **X-Primer Tool**

This software tool is developed by Graft technologies as a generic web-based tool that provides a graphic view to identify the problems existed in the software project. It also helps to identify the reasons for the existence of potential risks.

Q33. Explain various steps in risk management.

Dec.-19(R16), Q10(a)

OR

List and explain the phases in risk management.

Answer:

Risk management involves various phases that are as follows,

1. Risk identification
2. Risk analysis
3. Risk planning
4. Risk monitoring.

The risk management process is considered as an iterative process. It is involved throughout the project. Initially, a risk management plan was proposed. In that, the arising risks should be detected and more information will be available. So, risks need to be reanalyzed whether the risk priority has changed and if there is a change then risk avoidance and contingency plans should be changed.

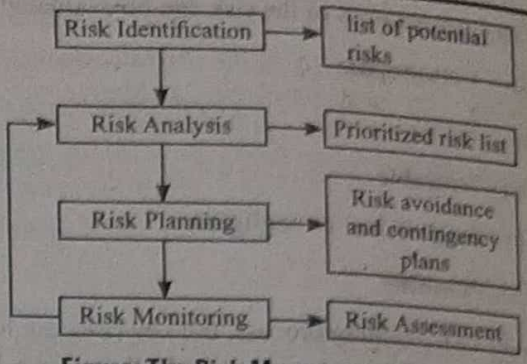


Figure: The Risk Management Process

1. Risk Identification

It is the first stage of the risk management process. This phase involves detection of risks that cause a major threat to the software engineering process. A group of members work in this phase to identify the possible risks. At the same time, the project manager plays major role and may use the experience to identify the most probable or critical risks. Some of the different types of risks are as follows,

(i) Technology Risks

The risks related to software or hardware technologies.

(ii) People Risks

The risks that are associated with persons, involved in development team.

(iii) Organizational Risks

The risks that come from the organizational environment where the software or product is being developed.

(iv) Tool Risks

The risks that come from the software tools and form other support software that is used to develop the system.

(v) Requirement Risks

These risks derived from changes made in the customer requirements and the process of managing the requirements changes.

(vi) Estimation Risks

Risks these are derived from in accuracies in estimating the resources and the time required to build the system in a proper way.

2. Risk Analysis

In the process of risk analysis, one should consider each identified risk and estimate the probability and seriousness of that risk. The estimation should be based on the previous projects. It is impossible to make precise, numeric estimation of the probability and seriousness of each risk. So, the risk should be assigned to one of a number of bands.

(a) If the probability of risk is less than 10% then it is very low. If it is 10 – 25%, then it is low. If it is 25 – 50% then it is moderate. If it is 50 – 75% or greater than 75% then it is considered to be high or very high.

(b) The consequences of the risk can be evaluated as

(i) Catastrophic

(ii) Serious

(iii) Tolerable

(iv) Insignificant or negligible.

3. Risk Planning

In the risk planning process, the key risks are identified and then strategies are developed to handle these risks. For each and every risk, respective action should be taken to reduce the problem occurred due to risk and information should be gathered while project monitoring so that the problems can be predicted. Contingency planning completely depends upon the decision and experience of project manager. Some of the strategies are as follows,

(i) Avoidance Strategies

By using these strategies, the possibility of risk occurrence can be reduced.

(ii) Minimization Strategy

By using these strategies, the risk impact can be decreased.

(iii) Contingency Plans

Contingency plan can be defined as plan that provides strategy to handle if any certain issues or risks occur.

4. Risk Monitoring

After the identification, analysis and planning, it is also required to monitor the progress of the product. It means to check whether the risk is becoming more or less probable and the risk impact has changed or not. This phase is called as risk monitoring. This phase also identifies and manages new risks. To monitor the progress of the product, some of the factors should be considered such as the number of requirements change requests that provides the risk probability and its impact. Risks need to be monitored at regular intervals in a project. At each management review, every major risk should be considered separately, and also it is required to check whether the risk is more or less likely to occur and if the risk severity and effects have changed.

5.2.1 Reactive Vs Proactive Risk Strategies

Q34. Differentiate between reactive vs proactive risk strategies.

Answer : (Model Paper-I, Q10(a) | Nov./Dec.-16(R13), Q10(a))

Reactive Vs Proactive Risk Strategies

Reactive Vs proactive risk strategies usually refer to two different approaches of risk management. Following are the details of describing each of these approaches.

1. Reactive Risk Strategy

It is one of the older approaches in risk management. In this case, usually the resources which can be utilized during the occurrence of risks are maintained separately. When the risks are encountered, these resources are used to nullify them. If the risks are certainly more strong such that they are intolerable even while applying these resources, then this project is moved to *crisis management team*. Here, the projects are strived rigorously to overcome the risks. This approach is also referred as *"fire fighting model"*. Since, the software development team rushes to the risk only when it occurs (i.e.) they will not take any preventive measures prior to risks. Hence, this kind of approach is now rarely used.

2. Proactive Risk Strategy

Consider the following thought "prevention is better than cure".

Proactive risk strategy is one of the risk management strategies, which considers the above thought to be the fundamental hypothesis. In this case, a document is prepared by stating all the probable risks, which are going to be encountered throughout the project, even before the project begins. This document also specifies the impact of the risks on the project along with their priorities (i.e., each risk is provided with a priority number which may be useful while scheduling plans to curb them on its occurrence). Later, another document containing all the details of methods for avoiding these risks are outlined. It has to be noted that, these risks cannot be eliminated completely hence, this document help us to drive these risks effectively to the lowest possible level.

5.2.2 Software Risks

Q35. What are the types of software risks?

May-18(R15), Q11(b)

OR

Explain Software Risks. Nov./Dec.-17(R15), Q10(a)

OR

What types of risks occur during software development? Discuss.

Answer : Nov./Dec.-16(R13), Q11(b)

Risks Encountered during Software Development

In general terms software risks refer to the probable threats expected to be faced by a given software. Risks can have two important characteristics,

(i) If risks occur, there are definite losses and

(ii) Occurrence of risks is uncertain.

In order to analyze the consequences and occurrence of risks, following are the different categories of risks.

Business Risks

Business risks often goes against the success of software. These risks are of many types. Details on them are stated below.

(a) **Market Risk**

A product with high valued features which is no way useful even to a common person is referred as *market risks*.

(b) **Sales Risk**

The sales risk occurs when the people who are responsible for marketing a given product and do not understand the terminology of the marketing product (developed). Sales risks usually occur, when there is no definite purpose of developing a given product.

(c) **Strategic Risk**

A developed product which is unable to fulfill all the requirements of a given organization.

(d) **Budget Risk**

The product which is unable to earn the specified amount or budget.

(e) **Management Risk**

The product which is unable to earn the attention from senior management due to the change in focus and in people.

Project Risks

If project risks becomes inevitable, then it is definite that the project development work will extend the scheduled time. This in turn causes the systematic rise in project cost. In worst conditions, project risks will wrap almost all the entities involved in project development process (i.e.,) the staffing and the organization, budgetary, schedule resources etc.

Technical Risks

Technical risks introduce problems which cannot be solved easily. Hence, delaying the estimated completion time of the projects, increasing cost and making the project unimplementable. It can affect various other entities like,

- Verification problems
- Implementation problems
- Maintenance problems
- Interface problems etc.

Also, it may have various other effects such as,

- Technical uncertainties and obsolescence
- Specification ambiguity etc.

Apart from the above mentioned risks, a software project suffers from the following risks. These risks were stated by Mr. Charette.

4. **Predictable Risks**

These are the risks which occurred in the past projects.

5. **Unpredictable Risks**

These are the risks which may occur all of a sudden. It is extremely difficult to realize these risks in advance.

6. **Known Risks**

These are the risks which are known only when a close inspection of the project plan, business and technical proximities of a project and its resources etc., is performed.

Q36. Explain about process risks.

Answer : Model Paper-III, Q11(a)

Process risks are the risks encountered when the software engineering process is not defined inaccordance to the required specification or when the design and testing are not performed in the planned fashion. The probability of occurrence of process risk is 40%. Its impact on the organization is marginal due to which there is less reduction in technical performance. However, if process risk is high, then the risk associated with the product development is also high.

The following are certain risk item issues considered with respect to process risk,

1. Is there a document developed that specifies the description about the software process to be used within the project?
2. Does the team members perform the software process in accordance with the documented specification?
3. Are the Formal Technical Reviews (FTR) being performed regularly by development and testing team?
4. Are the results of FTR documented such that they include both the defects encountered and the resources used?
5. Is there a need of configuration management for maintaining consistency among various requirements associated with the project (or) the system.
6. Is there a need of a mechanism for controlling the modifications done with respect to the end user requirements that can effect the overall product.

Q37. What do you mean by risk management? Explain how to select the best risk reduction technique when there are many ways of reducing a risk?

Answer : Nov./Dec.-18(R16), Q10(a)

Risk Management

For answer refer Unit-V, Q32, Topic: Risk Management.

Risk Reduction Techniques

The following are the certain types of risk along with their associated risk reduction techniques.

(i) **Personnel Short Falls**

This type of risk can be controlled by adopting any one of the risk management techniques such as Staffing with top talent, Job matching, Team building, Key personnel agreements, Training.

(ii) Unrealistic Schedules and Budgets

This type of risk can be controlled by adopting any one of the risk management techniques such as detailed cost and scheduled estimation, design to cost, incremental development, software reuse and requirements scrubbing.

(iii) Developing the Wrong Software Functions

This type of risk can be controlled by adopting any one of the risk management techniques such as organization analysis, machine analysis, user surveys, prototyping and early user's manuals.

(iv) Developing the Wrong User Interface

This type of risk can be controlled by adopting anyone of the risk management techniques such as prototyping, scenarios, task analysis and user characterization.

(v) Gold Plating

This type of risk can be controlled by adopting any one of the risk management techniques such as requirements scrubbing, prototyping, cost benefits analysis and design to cost.

Q38. Discuss the seven principles of risk management which were identified by SEI.

Answer :

The Software Engineering Institute (SEI) plays an important role in the software development life cycle. It provides seven principles as a framework to achieve effective software risk management as shown in figure. The purpose of risk management is to identify potential problems before they occur in the software. These principles are used as risk handling activities across the life cycle of product and project in order to achieve their objectives. These seven principles are as follows,

1. Encourage open communication
2. Encourage teamwork
3. Continuous process
4. Integrated management
5. Global perspective
6. Considering the forward looking view and
7. Develop a shared product vision.

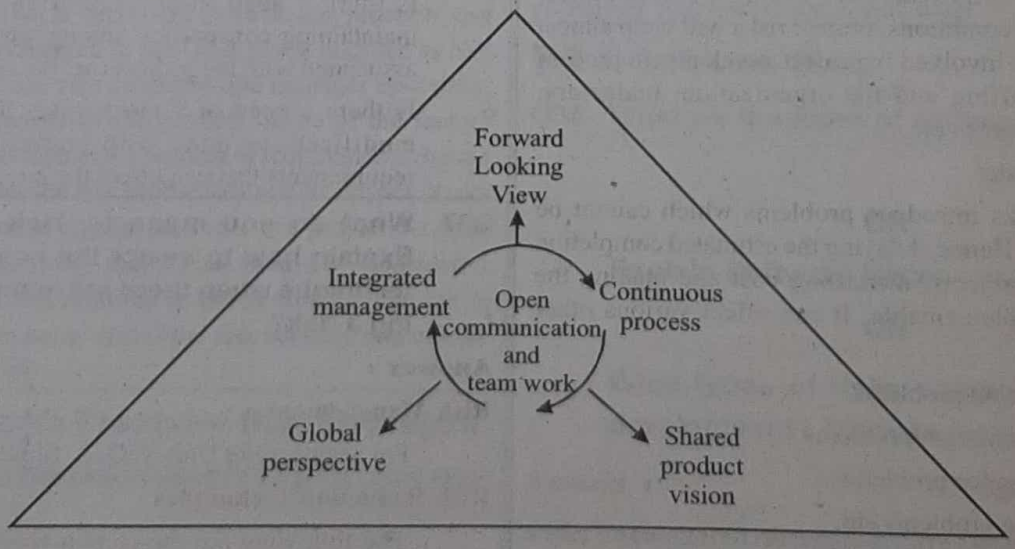


Figure: Seven Principles of Risk Management

1. Encourage Open Communication

An effective risk management process must encourage the open communication at all levels of software project. This process enables communication (formal and informal) across the organization. It helps to encourage the free flow of information among the entire group of software development team. Potential risk is avoided by the software developer.

Encourage Teamwork

2. The success of any software development project is mainly based on good team work. The goal of team management is to bring groups together with the talents, skills and knowledge for risk management activities. Encourage teamwork means motivating and coordinating the team members to set an ultimate goal for software development. They must be capable of identifying the risks and perform effective risk management activities. It considers the knowledge, skills and talents of the employee working on this type of risk.

3. **Continuous Process**

The most common scenario of any organization is that, the risks can be changed constantly. The key of continuous process is to maintain the proper attention throughout the software process. The continuous process enables to focus on risks as they change constantly and identifies new risks in the software process.

4. **Integrated Management**

In this principle, the team must identify the risks that are important in the organization. Integrated management is responsible for integrating the identified risks into the software process.

5. **Global Perspective**

Every software developing group comes across the same point of view towards the achievement of software objectives. In simple way, every group has the same definition of software success. The ultimate goal of software development group is to identify and solve the software risks that may be components, functional or business problems within the project boundary of the system.

6. **Considering the Forward Looking View**

While dealing with multiple interrelated projects, the software development group must focus on the same goal. This means, the organization may need to keep all groups with the same success point. These groups predict the future risks that may change constantly in software life cycle. So that effective risk management activities and the repeated plans are provided to manage the future events.

7. **Develop a Shared Product Vision**

The entire group of related developers, technical managers and administrative managers must have a clear vision of the software goals. If all the group members share the same vision of the software success then it becomes effective for software risk management.

Q39. List the major risks in a software project. What are the major ways to abate the risk of cost and schedule overruns?

Nov.-15(R13), Q11(b)

Answer :

Major Risks Involved in Software Project

For answer refer Unit-V, Q38.

Major Ways to Minimize the Risk of Cost and Schedule Overruns

1. **The Project Team**

In many complex projects, selecting a good team is one of the major task. This effectively contributes in controlling the cost and schedule overruns. Before selecting the team, the project managers must check team's capabilities, and see whether or not they match the project requirements, their cost estimates and how efficient are they in meeting the deadlines. An improper skill set not just create problems but also present significant drags in the project. However, selecting a diverse team is often advisable as they posses many different ideas, work styles, thoughts, skill sets and experiences.

2. **Proper Planning and Project Scheduling**

The project planning is one of the essential factors in avoiding the cost and schedule overruns. Before initiating the coding stage, project schedule and other necessary scopes must be planned. If scheduling is performed improperly, it can lead to wrong cost estimates and can increase the idle time of some team members. A Gantt chart or advance scheduling tool can help to produce an effective project schedule. It is essential as every project holds relative aspects, goals, milestones and results.

3. **Performing Mind Maps**

The mind mapping is a traditional technique which helps the programmer to visually organize the information. It helps the team to become more creative, solve problems and forecast the risk associated prior to the project initialization. The mind mapping can be performed on many levels by taking the inputs from many stakeholders.

4. **Attempt to Stick with Methodology**

The attempt to stay with in the scope which was originally planned has became a biggest challenge during the software development. This means that the developers include certain features of their choice where as clients demand something else and also the testing team perform modification in other features. Such changes can bring unexpected cost and schedule overruns. So, it is essential that the team should stick to the methodology that was originally planned.

5. **Being Ready for Unexpected and Change Control**

Due to the lack of risk management plans, many projects fail. The developer team faces unexpected problems or challenges which they are unaware of. In the attempt of solving this, the project can go over budget. Apart from this, the team must be efficient in controlling the changes that arise during software development. They can also implement change control plan template.

6. **Implementing Stakeholder Management**

The project manager must effectively implement the stakeholder management techniques. The managers must communicate among the team members so as to work seamlessly (Also they should keep track of external stakeholders i.e., clients and stick them to what was agreed upon in the plan. This helps to minimize the delays occurring in project and avoid working upon unnecessary thing thereby, making the scheduling work better.

7. **Keeping Track and Measuring The Progress**

It is the duty of the project manager to constantly keep the track of the progress in the project and various metrics to measure in projects. This helps to predict the project delays at early stage and its necessary solution.

5.2.3 Risk Identification

Q40. Describe the methods for risk identification.

Nov./Dec.-17(R15), Q10(b)

OR

How risk is identified? Explain.

(Refer Only Topic: Risk Identification Process)

Answer : **(Model Paper-II, Q10(b) | May-18(R15), Q10(a))**

Risk

In general terms, risks are nothing but the probable threats expected to be faced by a given software. Here, the two important concepts called '*known*' and '*predictable*' risks are to be considered.

(a) **Known Risks**

'*Known risks*' are usually measured by closely inspecting the given project plan and other crucial aspects of the project. These include business, technical vicinity and also various other authentic information sources etc.

(b) **Predictable Risks**

These are the risks which were encountered in past.

Apart from the above mentioned categories, a project manager further expresses two broad categories of risks (for sake of simplicity) i.e., *generic risks* and *product specific risks*.

(a) **Generic Risks**

Generic risks lower down the software quality and can act as a major source of threat to the success of the product.

(b) **Product Specific Risks**

As the name suggests, these are the risks which are specific to the product i.e., the type of language chosen, the specimen working and also the vicinity in which the product is underdevelopment.

Risk Identification Process

The foundational step in risk identification process is to develop a checklist consisting of various risk items. The checklist is given below.

1. **Characteristics of End User**

As the name suggests, the primary focus of this item (in the checklist) considers the potentiality of the customer as well as the notion of interaction of the software developer with the end users at specified intervals. These are determined to summarize the risks associated with them.

2. **Business Analogy**

In this case, the risks associated with the management aspects (i.e., the regulation drawn by the management or various other factors) are of primary focus.

3. **The Project Members and their Experience**

Here, the risks which are encountered due to the lack of abilities and experiences of project members are identified.

4. **Size of the Project**

Here, the risks associated with the project size are visualized.

5. **The Vicinity of Development**

In this case, the risks associated with the resources are deeply scrutinized.

A risk item checklist is displayed in the form of a table which gives a primary vision of identification of the risks.

Following are certain important questions that are derived for determining the extent to which a project is at risk.

- Does the clients possess certain realistic expectations?
- Is the project team a collaboration of all talents?
- Is the end user involved during the complete discussion of the requirement specification process?
- Does the software team possess good exposure to the technology on which it is working?
- Does the software development team well versed with the project requirements?
- Is nature of the software stable?
- Are all the team members capable of dealing with the project?
- Are the top level development officials and the customer representatives committed to go on with the product?

Components and Drivers Associated with Risks

This was initially brought up by the U.S air force. According to them, identification of risk drivers that deeply affects the software components (i.e., schedule, cost, support, performance etc.) is the duty of the software manager. Here, each software component can be defined as follows,

1. **Schedule Risk**

   As the name suggests, schedule risk refers to the extent of uncertainty that the schedule will be maintained as desired and on time delivery of the resultant product.

2. **Cost Risk**

   It refers to the extent of uncertainty in maintaining the budget of a product being developed.

3. **Performance Risk**

   It refers to the extent of uncertainty prevailing on the relevancy of the product and its satisfaction to the customer.

4. **Support Risk**

   It refers to the extent of uncertainty encountered during the process of product maintenance (i.e., rectification, correction, adoption etc).

5.2.4 Risk Projection

Q41. What is meant by risk projection? Discuss in detail the various steps in risk projection.

OR

Elaborate on risk projection steps. **(Model Paper-II, Q11(a) | May/June-19(R16), Q10(a)**

OR

Describe the methods for Risk Projection.

Answer : **Nov./Dec.-17(R15), Q11(b)**

Concept of Risk Projection or Risk Estimation

In general terms, risk projection addresses the risk in the following two ways,

(i) Probability of occurrence of risks and

(ii) The consequences associated with those risks.

Risk Projection Steps

Following are the four risks projection steps,

1. Develop a measure using which probability of existence of risk can be known.
2. Depict the outcome of a risk.
3. Project the effects which can be observed on both the project as well as on the end product being developed.
4. Finally, measure the appropriateness (i.e., trueness) of the risk projection.

The main aim of the projection steps is to provide suitable priorities to the prevailing risks so that the appropriate resources can be allocated which in turn will be helpful in dealing with those risks.

Generating the Risk Table

A risk table is extremely helpful to a project manager in sorting out various types of risks depending on their priorities. Hence, full attention is to be provided to high priority risks (which may eventually affect the project as well as the products).

An example of a risk table is given in table below.

Various Types of Risks	Category of Risks	Probability of Occurrence of Risks	Impact	Risk Mitigation, Monitoring and Management Plan (or) RMMM Plan
X	Project size risk	30%	Castastrophic (1)	
Y	Business impact risk	70%	Critical (2)	
Z	Process definition risk	40%	Marginal (3)	
A	Staff size and experience risk	60%	Castastrophic (1)	
B	Customer characteristic risk	50%	Marginal (3)	
E	Product size risk	60%	Critical (2)	
C	Technology to be built risk	80%	Critical (2)	
D	Business imapact risk	45%	Negligible (4)	

Table: Risk Table

The significance of each column can be realized by taking a glance of the above mentioned table. For example, the first column enrolls "*various types of risks*". These are usually those risks which are encountered by a given organization. The entries in this column can be obtained by considering the checklists (which usually accommodates the risk items). The second column enrolls "*category of risks*", in which seven categories are used. The third column enrolls the "*probability of occurrence of risks*". Here, the probability values are nothing but the estimation predicted by each project member. The fourth column contributes to "*impact values*". These impact values can be one among the following i.e., catastrophic, critical, marginal and negligible with their respective values 1, 2, 3 and 4 respectively. These impact values are obtained by averaging the risk components such as cost, support, performance and schedule. Before going to RMMM column, following are certain important conclusions related to the risk table.

- The project manager, now thoroughly analyzes the table and by using the impact as well as probability values, various entries are sorted. While doing so, the entries possessing high probability and impact values forms the initial entries, followed by the entries with low probability and impact values.
- The above step is important because the highest attention can be provided to the risks which frequently occurs (having high probability). At the same time they possess high impact values. Now, the project manager, with an intention to separate these entries from other entries, draws a line in the risk table. The entries which lie above this line are referred as "*entries with first order prioritization*". Now, the same procedure is repeated for the entries lying below this line to form the "*entries with second order prioritization*".

A documentation is prepared by referring to the risks which lie above the line. This documentation is attested in RMMM plan column.

A diagrammatical view of the way the entries of the first column are preferred are represented as follows.

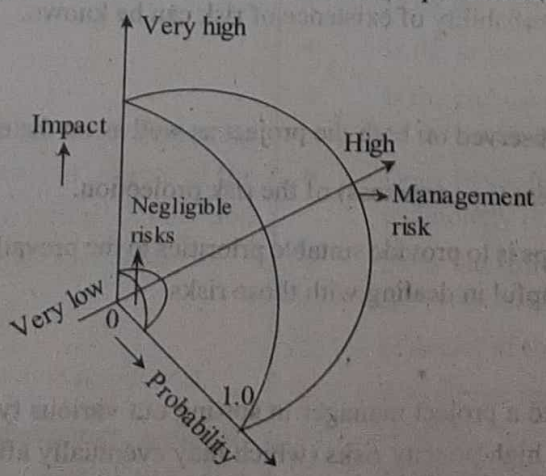


Figure: Diagrammatic Overview of Sorting of Risks

Q42. What is meant by risk assessment? What are the different steps to be performed in risk assessment? Explain.

Answer : Model Paper-III, Q11(b)

Risk Assessment

Whenever risks become inevitable, following are the three factors through which their effects on current environment can be determined.

(i) Risks nature

(ii) Risks scope and

(iii) Risks timing.

1. **Risks Nature**

This factor describes the problems associated with the given risk.

2. **Risks Scope**

This factor describes the intensity (how deeply the given risk affects) of the risk in the current situation.

3. **Risks Timing**

This factor describes the duration during which the given risk will exert its effect.

Steps to be Performed in Risk Assessment

The effects of risks can be determined by the following steps,

Step1: It has to be noted that, each risk is a collaboration of several risk components. When a given risk becomes inevitable, then the probability of occurrence of these components is measured.

Step2: Calculate the impact values for each component.

Step3: Draw the risk table and fill the entries as described below.

Generating the Risk Table

For answer refer Unit-V, Q41, Topic: Generating the Risk Table.

Finally, the risk exposure value is calculated as,

$$\text{Risk exposure} = \text{Probability of occurrence of risk} * \text{Effect (or cost) of risk on the project}$$

5.2.5 Risk Refinement

Q43. What do you mean by the term "risk refinement"? Explain.

Answer :

Risk Refinement

Risk refinement is a process of refining the already stated risk definition and representing it in a more detailed form. The process of refinement is initiated when project planning process is thoroughly analyzed. This refinement has been proved useful to mitigate, monitor and manage these risks in a more convenient manner. The process can be carried out by following a standard notation of risk representation referred to as Condition Transition Consequence (CTC). This notation states that, "If a condition is specified, then there exist a concern that may possibly result in a consequence". The above statement can be formulated as,

"Given that <condition> then there exist a concern that (possibly) <consequence>".

From the above CTC statement, the following conclusion can be made.

If majority of the reusable software components adhere to the design standards and remaining fails to satisfy, then in such situation there exist a concern which specifies that a built-in system may be integrated with 70% of the planned reusable components and the remaining 30% of components must undergo custom engineering.

The following is the way of refining the above stated condition.

Subcondition 1

There are some reusable modules that were designed by external users, basically a third party, who have no knowledge about the internal design standards.

Subcondition 2

The design standard associated with the reusable component interfaces are not defined properly due to which they may not adhere to some existing reusable components.

Subcondition 3

The language used for implementing some reusable components cannot be used in the targeted environment.

Advantages

The advantages of refining the risk definition are,

(i) It helps in segregating the important risks from the other risks.

(ii) It helps in performing thorough risk analysis without any difficulty.

5.2.6 RMMM

Q44. What is RMMM? Explain various methods followed to mitigate, monitor and manage risks.

Dec.-19(R16), Q11(a)

OR

Explain about risk mitigation, monitoring and management.

Answer : **May-13(R09), Q7(b)**

Risk Mitigation, Monitoring and Management (RMMM)

Any risk analysis activity should include the following steps,

(i) Risk avoidance

(ii) Risk monitoring and

(iii) Risk management as well as contingency planning respectively.

It is one of the famous assumptions that "prevention is better than cure". Hence, the software team always takes an initiative before the risk approaches can resort to risk avoidance strategy. For effectively implementing the risk avoidance strategy, one has to depend on suitable plans referred as "*risk mitigation plans*" (the word "mitigation" refers to the process through which the occurrence of risks can be eliminated).

Consider an example, wherein one can analyze the process of risk mitigation, monitoring and management. Consider the risk as "high staff turnover".

Following are the few effective steps which can be termed as a process to mitigate the given risk (high staff turnover).

- Initially, project managers should mingle with all the project members and analyze the factors for occurrence of this risk.
- Prior to initiation of project they try to nullify those factors which can be easily resolved.
- As the project, proceeds, a strategy is devised which effectively promotes the successful running of a project even when the team members are not available. It has to be noted that, this step is taken before the occurrence of a risk.
- Analyzing the staff to be working on different aspects of the project, their status is determined and this information is spread among the project members.
- Documentation is developed in a timely manner and it is ensured that activity is active throughout the project development.
- The status of all the project members is examined.
- When a critical employee is identified, an additional employee is associated who can act as a backup for such employees.

Consider, the second step of RMMM.

Following are the important concepts to be monitored effectively.

- The behavioral as well as various other aspects such as attitude, reactivity etc., of all the project members need to be considered.
- The type of collaboration among the team members.
- Monitoring the genuine problems with their solutions as well as their positive outcomes.
- Job requirements within and outside the given organization.

Now, consider the final instance of RMMM i.e., *risk management and contingency planning*. At this instance, it is usually considered that all the above mentioned factors failed and the risk turned active. Hence, if staff turnover is considered as a risk, then its probable effects can say that the project is running behind the scheduled time as most of the employees had given up their resignations.

Now, if such conditions arise, the project leaders get relaxed to the major extent if he/she has implemented the risk mitigation steps (as stated above) effectively. This is because, while implementing the risk mitigation steps, the project leader preserves the information in documents, maintenance of backup employees and each piece of information is spread among the project members. Now, the project leader simply reschedules the project temporarily and grants the backup team to access existing information. While doing so, the new employees acquire the normal speed within less time. Also, the staff members who have planned to walkout of the organization are made to curb their work and start addressing the new employees with the required information.

Analyzing the above mentioned facts, one can remain relaxed and ascertain that these risks can be easily avoided. But implementing RMMM can introduce additional costs with huge volumes of redundant data. Also implementing RMMM on large projects can turn out to be a new project. It has to be noted that, risks are encountered even when the given project is delivered to the customers. This usually occurs due to the failures in the development process.

5.2.7 RMMM Plan

Q45. With a sample risk information sheet, explain the RMMM plan.

OR

Provide the format of risk information sheet.

May/June-19(R16), Q10(b)

Answer :

In every software development activity, risk management plays a major role. The essential steps of risk management process are grounded under the side heading called "Risk Mitigation, Monitoring and Management Plan". Hence, the activities performed during risk management is enrolled into documents and is considered as the RMMM plan. But, it is also a trend that instead of developing the entire RMMM plan, some software development staff organizes separate sheets called "risk information sheets" for each risk. Usually, the RMMM plan is referred by the software managers in order to develop the overall software plan. For feasibility in editing, manipulating, sorting etc., the risk information sheets are implemented in the databases. The format for risk information sheet is shown in figure.

Risk Information Sheet			
RISK ID	DATE	PROBABILITY OF OCCURRENCE	IMPACT VALUE
DESCRIPTION			
REFINEMENT/CONTEXT			
MITIGATION/MONITORING			
MANAGEMENT/CONTINGENCY PLAN/TRIGGER			
CURRENT STATUS			
ORIGINATOR (NAME)		ASSIGNED (NAME)	

Figure: Format of Risk Information Sheet

As project proceeds and RMMM documented, the risk mitigation and monitoring steps begin. Risk mitigation is concerned with reducing the problem to the acceptance level, whereas risk monitoring is concerned with project tracking. Its role is to identify which risks are responsible for which problems (occur during the project).

It has three main objectives,

(i) To identify whether risks that are measured really occur during the project execution or not.

(ii) To confirm whether steps that are defined to reduce the risk are applied in proper manner or not.

(iii) To collect the required information for future risk analysis.

5.3 QUALITY MANAGEMENT

5.3.1 Quality Concepts

Q46. Write short notes on,

(i) Quality

(ii) Quality control

(iii) Quality assurance

(iv) Cost of quality.

OR

Explain about quality control, quality assurance.

(Refer Only Topics: Quality Control, Quality Assurance)

Answer : **May-13(R09), Q8(a)(b)**

(i) Quality

Quality is a measurable attribute associated with almost every entity of this world. It has two terms which are of primary importance. They are,

(a) Quality of Design

Quality of design refers to conforming specifications related to design of a software, provided by the designers. Some of quality design characteristics are,

- System design,
- Specifications related to the software and also
- Its requirements.

(b) Quality of Conformance

It refers to an extent that the software being manufactured satisfies all the specifications issued by the designers. Quality of conformance looks only for implementation of quality design characteristic issues as its target objective. Hence, to have good quality of conformance, one has to see that the production of software must satisfy all design specifications, requirement specifications and also quality specifications laid prior to the software development process.

One of the latest approximations delivered by a scientist Robert Glass suggests, software quality to be one of the entities of user satisfaction.

User satisfaction = A complaint product + High quality + Delivery of product within the estimated schedule and time

(ii) Quality Control

Quality control is usually a process in which the entire software development process is inspected, analyzed deeply and suitably tested so as to make sure that the produced product satisfies all the specifications laid prior to the software development process. Moreover, a feedback loop must be provided to the work product process.

When the product is analyzed, a feedback report is developed and is provided to the development team. This is an important activity through which, limitations existing in the product can be easily eliminated. Hence, quality control mechanisms check whether the product is upto date with all specifications and requirements.

(iii) Quality Assurance

Quality assurance can be treated as a process of judging efficiency and effectiveness of quality control process. Hence in this process, the documents developed during the quality control process are examined deeply and are reported to the management authorities. Through this, the management can be focused to the product's quality. Hence, any dissatisfaction reported in this process is usually dealt by the quality managers.

(iv) Cost of Quality

Cost of quality usually refers to the cost incurred while performing quality related activities. Cost of quality usually comprises of three components as shown below,

Figure: Cost of Quality's Components

(a) Appraisal Cost

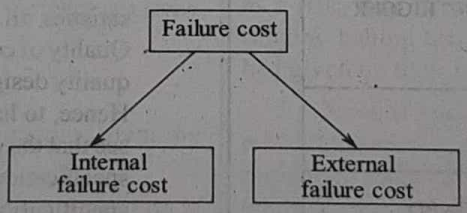
Appraisal costs are nothing but the cost of inspecting each process individually and many processes combinely, maintenance and testing.

(b) Prevention Cost

When training, technical reviews, quality planning, test equipment etc., are conducted. There associated cost comes under prevention cost.

(c) Failure Cost

Failure cost is the cost of removing bugs from a project. If the product is free of bugs, then this cost would be zero. Failure cost is of two types (i.e.) internal and external.



❖ **Internal Failure Cost**

Usually, this type of cost accounts when any bug creeps into the product prior to its delivery. Hence, internal failure costs are associated with the activities involved in finding and repairing of these bugs. Example, repairing communication network, repairing hardware crashes, etc.

❖ **External Failure Cost**

These are the costs resulted when a bug arises after the product is delivered to the end user. Example activities accounting for external failure cost, warranty work resolving queries of customers, replacement of products helpline support etc.

Q47. What is software quality control?

Answer :

Model Paper-I, Q10(b)

Software Quality Control

Software quality control is defined as the process of regulating the software development process such that methods of software quality assurance and the quality standards are practised. Basically, the quality control process is meant for comparing the outcome of the software process with the set of project standards.

If the product does not satisfy the requirements then feedback loop should exist in quality control such that performance of the process can be improved, thereby reducing the defects in a software product.

The primary key to a rich quality product is to reduce variations, from one product to another product. Variation control plays a central role in quality control for meeting the requirements of software.

For instance, if a DVD manufacturer wishes to minimize or reduce the variations while duplicating the DVDs the replicas of DVD may be created. But sometimes the disk duplication machines may get exhausted due to fluctuations between the DVD samples. Thus, variations must be properly controlled to assist quality control.

In order to test the product quality, two strategies are devised. They are,

(i) Quality reviews

(ii) Automated software assessment.

(i) Quality Reviews

In this strategy, software processes used in product development and related documents are reviewed by a group of people, for checking whether project standards are met or not. If any variations arise from the standards then they are notified to one project manager.

(ii) Automated Software Assessment

In this strategy, software and the produced documents are reviewed by a programmer. It also includes measuring and comparing the software quality with the expected requirement level.

Q48. What do you mean by Good enough software? Explain with an example.

May/June-12, Set-1, Q8(a)

Answer :

"Good Enough" Software

Good enough software not only delivers excellent features and functionalities that the users expect but also delivers unknown or special functions and features containing known bugs. The intention behind it is to correct and improve the software in the next version. The vendor of the software hopes that the users will be satisfied with the improved version and will ignore the bugs in proceeding version.

Example

Consider if a large company with good marketing budget is successful in delivering version 1.0 then it can easily convince its customers that it can get an improved version in its second attempt. But a smaller company cannot take such a risk because the negative buzz created by its customers leads to a drastic fall in the company's sales forcing the vendors to shut down the company.

The delivery of embedded system applications or hardware-integrated application software with known bugs can cause legal prosecution to the company. Therefore "good enough" software should be delivered with caution if and only if it is able to solve the software quality problems.

Q49. Discuss the statement, "quality is a complex and multifaceted concept".

Dec.-11, Set-3, Q4

Answer :

The conclusion "Quality is a complex and multifaceted concept" is made by "Garvin". He described quality from the following five perspectives,

1. **Transcendental Based View**

   In practice, it is difficult to define quality, however, it can be recognized well. Moreover the feedback of the quality varies from one customer to another and depends on how they perceive the product. The quality of the product depends on the customer's reaction towards it. So it is necessary for the developers to interact with the customers regularly to make sure that their specifications reflect the customer's needs well.

2. **Manufacturing Based View**

   Quality is defined in terms of conformance to specification. According to this definition, an organization develops a well-written specification to develop a product rather than involving too many people for making judgements throughout the development program.

   A well-written specification gives only about 40 to 60 percent defects in the software project.

   It can also be said that this perspective manufactures the product according to the specification within accepted tolerances. However, quality requires "Precion of effort". The organisation must understand its aim before improving the "Precion of effort". Later, it must design its specifications.

3. **User Based View**

Quality is defined as "fitness for use". Based on this concept, the software developers provide all the properties of their software product for conformation. But sometimes, this concept proves ineffective because, after deployment, the product may fail to work as per expectations of users. Thus appropriate quality measures can be obtained by considering the view points of the individual users along with their context of use.

4. **Product Based View**

The internal characteristics of the product is its quality. These characteristics include attributes of the quality like correctness, functionality, integrity, fault tolerance, efficiency, privacy and safety. It also includes reliability, availability, usability, adaptability, installability, portability, maintainability and flexibility. The quality is rich if the software has more attributes discussed above.

5. **Value-Based View**

Quality can also be defined based on the amount a customer is willing to pay. This means a customer can pay more for a "Good enough" quality product.

5.3.2 Software Quality Assurance

Q50. Define software quality assurance. State various SQA activities. Dec.-19(R16), Q10(b)

OR

Explain the activities of software quality assurance group to assist the software team in achieving high quality.

Answer : May/June-19(R16), Q11

SQA

For answer refer Unit-V, Q46, Topic: Quality Assurance.

Software quality assurance is a critical component in software engineering. It consists of various tasks that are related to two constituencies,

(a) Software engineers perform technical activities and

(b) Software Quality Assurance (SQA) group performs quality assurance planning, analysis, reporting, forecasting and record keeping.

In order to address quality and activities for controlling it, software engineers implement technical methods, conduct formal technical reviews and perform testing.

Software quality assurance tasks must be assigned in a way that quality is ensured through all the development and maintenance activities. To achieve a high quality end product the software quality assurance group develops a charter that helps the software team during the development process. The Software Engineering Institute (SEI) suggests some software quality assurance activities that focuses on quality assurance, record keeping, oversight and analysis and reporting.

SQA Activities

An independent SQA group performs following activities in order to implement activities suggested by SEI.

(i) Preparing SQA Plan

In order to ensure quality, a plan governing the activities performed by SQA group and software engineering team is generated. With this plan, SQA group can get the information regarding the documents they are supposed to produce, evaluations that must be performed, feedback to be given to software project team, error reporting and tracking procedures to be used, audits and reviews to be done and standards relevant to the project to be followed.

(ii) Reviewing Software Engineering Activities

The SQA group verifies whether the software engineering activities are as per the software process or not. Any deviations from the defined process are identified, documented and tracked by the SQA group. Once they finish with identifying, documenting and tracking, they must ensure that corrections are made in the software product.

(iii) Reviewing Process Description

The software team chooses a process for the activities they must perform. The SQA group takes the process description and checks whether the description is as per the software project plan, internal software standards, organizational policies and externally imposed standards.

(iv) **Auditing Selected Software Work Products**

SQA group reviews few selected products and any deviations from a set of work products. With this, the software processes are identified, documented and tracked. They ensure that deviations are corrected by the software team. Moreover, they continuously update the project manager regarding the review results.

(v) **Ensuring Proper Handling and Documenting of Deviations**

The SQA group while reviewing may come across certain deviations in various activities and products. They ensure that these deviations must be managed as per the defined procedure. These deviations can be encountered at any stage of the software development cycle.

(vi) **Recording and Reporting Non-compliance Items**

The SQA group continuously tracks and reports any non-compliance item they encounter to the senior management. They ensure that the non-compliances are correctly resolved by the senior management.

Q51. Discuss about product standards and process standards.

Answer :

Standards

Software quality can be assured if the defined standards are followed. Beginning from the requirements phase, till the product reaches the customer, each activity involved must be according to these standards. In software engineering, standards can be of two types,

1. **Product Standards**

They are applied to software product under development. It specifies several standards about the structure (or format) of various documents created during development, coding styles to be followed etc.

2. **Process Standards**

They refer to standards that are applied while the software is being developed. It specifies the process of designing, implementing and validating the software product. These two standards are important for the following reasons,

(i) The standards are built by selecting the most appropriate or best practice that a company follows. These best practices may be result of long research and experience of the company. By following standards one can avoid repeating mistakes made in past.

(ii) By following standards, all engineers will adopt the same set of activities or practices. As a result, work done by one person can be easily understood and continued by others.

(iii) If standards contain best practices, then the framework provided by them can be used to implement quality assurance. Hence, Quality Assurance (QA) is important for both customer and development organization to evaluate the quality of a given software product.

Q52. Is it possible to assess the quality of software if the customer requirements keeps changing? What it is supposed to do?

Answer :

It is not possible to assess the quality of software with respect to customer requirements. Since requirements of software act as the basis for quality measurement, if requirements are not sure then quality lacks conformance. Basically, quality of software lacks due to the following factors,

1. If the requirements are not specific in nature.
2. If the software development does not support specific standards.

A quality software should be bug free and delivered on time in order to meet the customer requirements. It must run on any type of operating system.

In general, assessment of software quality is little tricky in nature as the quality has its own form with respect to the product. In order to assess the quality, first, the quality is measured then it is reported back based on the quality level. So, one should know well about the pros and cons of the quality for a specific product. For this a theory is needed for testing the quality of the product.

If software specifications, as per the customer needs are not specific enough then, there is a possibility of occurrence of bugs. Bugs may also occur if,

(a) Software is not designed properly.

(b) Complex software with poor documentation, time pressure or poor skill set.

If customer keeps constantly changing the requirements in maintenance phase of software development then, it leads to changes in the software product which in turn introduces errors in the software. These bugs are removed with testing.

Following measures must be considered while assessing the software quality,

(i) Customer should be clear enough about his/her requirements. Moreover, he/she must communicate properly to the software engineer.

(ii) To ensure high quality in software, a systematic plan of actions must be carried out.

(iii) Interaction between customer and software team should be clear enough for gathering requirements.

(iv) Testers should possess full-fledged knowledge about the specific software in order to remove the bugs in an efficient way.

Despite of using above measures, assessing quality of a product becomes a challenging task. Thus, customers, software engineers, testers are all responsible for ensuring quality software.

5.3.3 Software Reviews

Q53. Explain in detail on various aspects of software reviews.

Answer :

Software Reviews

Software review is the process of inspecting the software at different points with an intention of pointing out defects and errors normally occurring in the software, hence, improving its quality. Basically, there are many types of software reviews each having its own value or importance.

Formal Technical Review (FTR)

For answer refer Unit-V, Q56, Topic: Formal Technical Reviews.

Software Defects and Cost

The primary purpose of formal technical review is to uncover errors. It is most widely accepted analogy that about 60 to 65% of bugs prevail during the designing phase of software engineering and FTR claims to uncover about 75% of errors. Hence by doing so, cost of effort applied in other software development activities can be lowered to large extent.

"If 1.0 unit is the cost of detecting an error due to reviewing process, then for 6.5 units, 15 units and between 60-100 units will be the cost of detecting the same error, before testing phase and after the disposal of the software to end user respectively".

Amplification and Removal of Defects

Defects can be generated and detected using defect amplification model given below,

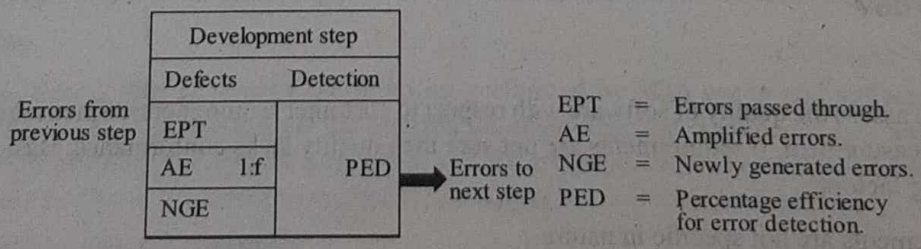


Figure: Pictorial Representation of Defect Amplification Model

The basic usage of defect amplification model is that it provides information related to defect amplification and detection during initial design, detail design as well as coding steps of software development activity. In the above figure, the rectangle represents a single development step in the entire development process. In this step, few errors may get developed at same instant and other errors may creep from previous steps. Among these errors, strengths of few errors get boosted (can also be referred as amplified) with a value of 'f', where 'f' is an amplification factor. In the next partition of the rectangle percentage of efficiency for error detection which is the strength of reviewing the step with an intention of detecting an error. Further, if certain errors which do not get detected are passed or flown to next step.

Defect Amplification with Reviews

Let us consider an initial design involving 10 initial errors or defects which are amplified to 24 errors prior to testing process i.e., detail design contains 2 errors are passed from initial design. The percentage efficiency comes 50% from 70%. In coding, 5 errors are passed from the detail design. Then percentage efficiency becomes 60%. During this, errors are amplified to 24 i.e., before testing. After testing process, the undiscovered errors are reduced in a step by step manner i.e., from 24 to 12 to 6 and finally to 3. This is represented as,

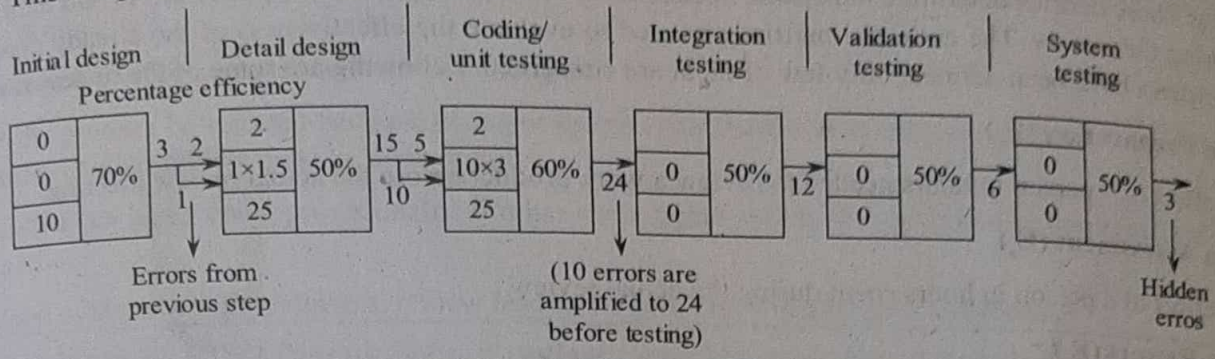


Figure: Defect Amplification With Reviews

Defect Amplification Without Reviews

Let us consider the same constraints as in above approach i.e., 10 initial errors. Here 10 errors are amplified to 94 errors prior to testing process. From initial design to coding phase, 10 errors are amplified to 94 undiscovered errors and their percentage efficiency also gets increased from 0 to 20%. After testing process starts, the 94 errors are reduced to half of it i.e., 47 errors undiscovered. In each step, they are reduced half of the percentage.

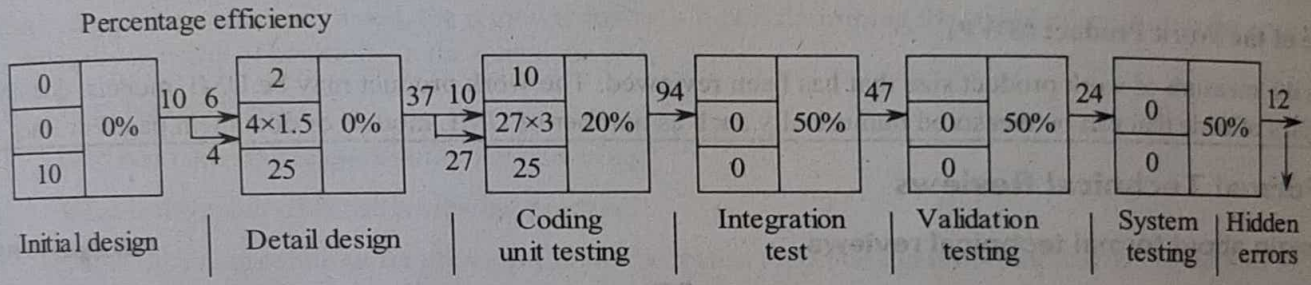


Figure: Defect Amplification Without Reviews

Hence, defect amplification without conducting the reviews leads to increase in total cost.

If we compare the above two methods, then total cost of amplification without reviews will be more than total cost of with reviews. It is otherwise called as incremental defect approach.

Q54. Discuss about software reviews. May-18(R15), Q10(b)

OR

Explain the use of Software Reviews. Nov./Dec.-17(R15), Q11(a)

OR

What is a software review? Who does it? Explain the importance of it. May/June-12, Set-4, Q2

Answer :

Software Review

For answer refer Unit-V, Q53,.Topics: Software Reviews, Formal Technical Review (FTR).

Who Conduct Software Review

The software engineers and others will conduct the technical reviews along with their team.

Importance of Software Review

(i) The correction of errors that are found initially in the process is cost-effective.

(ii) Errors are amplified along with progression of the process. Hence, the minor errors that are not corrected initially can be amplified into a big set of errors later in the project.

(iii) Reviews reduce the amount of rework that is required late in the project. By this, lot of time is saved.

Q55. Explain about review metrics and their use in detail.

Answer : May/June-12, Set-3, Q5

Review Metrics

Review metrics are the metrics that are gathered after conducting each review to determine the working of the quality control activities. These metrics determine the efforts needed to conduct the review and types of errors and their impact, that are not covered during the review. The collected metrics are used to evaluate the effectiveness of the already conducted reviews. Though many metrics have been defined, only few of them are concerned. Following are some of the review metrics,

1. **Effort in Preparation (E_p)**

   It is the effort of a person in hours needed to review a work product before the actual review meeting.

2. **Effort in Assessment (E_a)**

   It is the effort of a person in hours spent during the actual review.

3. **Effort in Rework(E_r)**

   It is the effort of an individual (in hours) spent to remove the bugs that are not covered during the review.

4. **Identification of Major Errors (Err_{major})**

   It is the identification of the number of errors that can be termed as major.

5. **Identification of Minor Errors (Err_{minor})**

   It is the identification of the number of errors that can be termed as minor.

6. **Size of the Work Product (SWP)**

   It is the measure of work product size that has been reviewed. The work product may be UML models, document pages or lines of code that can be measured numerically such as number of UML models or document pages or lines of code.

5.3.4 Formal Technical Reviews

Q56. Explain about formal technical reviews. Nov./Dec.-18(R16), Q10(b)

OR

Discuss about formal technical reviews. May-18(R15), Q11(a)

OR

What is the significance of formal technical review? Explain.

Answer : Nov./Dec.-16(R13), Q10(b)

Formal Technical Reviews

Formal Technical Review (FTR) is a Software Quality Assurance (SQA) activity. It is one of most important tool for filtering bugs from a software. In this case, a software engineer reviews the work of other software engineering.

Review Meeting

For any FTR format, the following constraints must be considered,

(a) Number of meeting members or reviewers should range between 3 and 5.

(b) Each reviewer must prepare in advance for the meeting by not taking more than 2 hours.

(c) The meeting duration must be less than two hours.

By the above constraints, it can be understood that the FTR does not focus on the entire design of the software but only specific part of software. 'Walkthroughs', are done for either a module or a set of modules. By these reviews, FTR can detect many errors from the selected module.

FTR mainly focuses on the work product, such as any part of requirement specifications, components detailed design, components source code listing etc.

After completing the development of work product, the producer or the software engineer informs the team leader that the product is ready to be reviewed. The team leader then contacts a review leader for reviewing the product.

Few copies of the product are produced by the reviewer among which two (or) three are given to some other reviewers for identifying defects. These other reviewers prepare a detailed document carrying information regarding the product along with the problems associated with it. These reviewers prepare these points in not more than two hours and after that, a meeting is conducted which will be attended by all the reviewers along with the producer.

During this meeting, a reviewer will note all the important points discussed. When producer describes the walkthroughs, reviewers address problems from the document they prepared. These issues are recorded and when the meeting ends, all the members must decide,

(i) Should the product be accepted without any changes?

(ii) Should the product be rejected because of major errors encountered in it that cannot be recovered? If these errors are eliminated, another review must be done or not.

(iii) The product can be accepted provisionally, if it has some minor errors, then it should be corrected and no further review is required.

At the end of the meeting, reviewers, review leader and producer together take a decision.

Q57. What is meant by FTR? Discuss about review reporting and record keeping.

Model Paper-II, Q11(b)

Answer :

FTR

For answer refer Unit-V, Q56, Topic: Formal Technical Reviews.

Concept of Record Keeping and Review Reporting

Generally, the formal technical reviews are conducted by a designated reviewer by consulting various project members. Once a reviewing process gets completed, the reviewer makes a report describing the issues raised during the reviewing session. This report usually consists of solutions to the questions like,

(i) What was reviewed?

(ii) Who had taken the responsibility for reviewing?

(iii) What is the result of entire reviewing process?

Hence, after obtaining solutions for above questions the review report task gets finished and this report now forms important information of entire project.

Advantages

By using a review report,

(i) Defects in the product can be identified.

(ii) The producer can easily cross check when changes are made to the product.

Q58. Explain briefly on the following with respect to formal technical reviews,

(a) Review guidelines

(b) Sample driven reviews.

Answer :

(a) Review Guidelines

The reviewer must initially be prepared on the aspects to be reviewed. This is important, because unplanned reviewing can cause certain unhealthy problems. Hence, following are certain guidelines essential for the reviewing process.

(i) Before beginning reviewing task, it is often prescribed to the reviewer to initially go through previously dealt reviewing tasks. This is important so as to eliminate problems which were encountered previously from current product.

(ii) It is prescribed to all reviewers to undergo suitable training before conducting reviews. This training matures all the reviewers to understand various technical as well as sociological aspects of the team members.

(iii) Reviewing should be suitably planned and then implemented. It should be considered as a separate task during the software development process. On completion of reviewing, even the modifications to the current software development process should be suitably planned to assure expected results.

(iv) It is often essential to consider checklists while reviewing. This remains important to plan out a review process.

(v) It is often fruitful to lower down the number of reviewers.

(vi) On conducting a reviewing task, it is fruitful to display the review report so that it will remain helpful to all other reviewers.

(vii) Solutions to the problems encountered during reviewing process should be obtained only after the completion of reviewing process.

(viii) It is often prescribed to the reviewer not to indulge in unnecessary debates which may consume precious time.

(ix) It is important for the reviewer to stay on the prospective of reviewing and schedule.

(x) The primary focus of reviewing process should be on the product which is under development. Reviewing process should be conducted in such a way that, at the end of meeting, all the participants should get encouraged rather frustrated.

(b) Sample Driven Reviews

In general, it is often recommended that almost every software project should undergo technical reviewing so that its granularity can be known. But due to lack of time and other resources, technical reviewing gets neglected though it is one of the finest methods to ensure software quality. Hence, basing on these illustrations, Thelin and few of his colleagues came up with a new methodology referred as "Sample driven reviews". In this methodology, each of the software samples are examined closely and efforts are made to distinguish the erotic samples from less error prone samples. Once the separation process is completed, the erotic samples are then attempted with all the possible formal technical reviewing concepts, so that degree of deficiency of these samples can be lowered to maximum extent. The entire process is referred as sample driven reviews. In order to attain the virtues of sample driven reviews effectively, the following steps are followed,

Step1: Divide the given project into number of work products, consider a fractional part of a given work product and examine it thoroughly (let the fractional part be represented by f_n and 'n' be the given work product). Number of errors encountered in 'n' are recorded, and represented as 'e_n'.

Step2: Multiply e_n with $\frac{1}{f_n}$ $\left(\text{i.e., } e_n * \frac{1}{f_n}\right)$. This is essential to obtain the overall errors in a given work product 'n'.

Step3: Place the work product possessing the highest overall error value at the beginning followed by the rest, in the decreasing order.

Step4: Apply every possible formal technical review mechanism to all the work products possessing high overall error values.

While implementing above mentioned steps, one has to remember that the fraction selected should represent the overall work product. Moreover, it should be suitably larger enough to provide ease while reviewing.

Q59. What is role of formal technical reviews in quality control? Discuss various steps and procedure to conduct a FTR.

Answer : Dec.-19(R16), Q11(b)

Role of Formal Technical Reviews in Quality Control

For answer refer Unit-V, Q56, Topic: Formal Technical Reviews.

Various Steps and Procedures in Conducting Formal Technical Reviews

The various steps and procedures for conducting an FTR are given as follows,

(i) Review Meeting

For answer refer Unit-V, Q56, Topic: Review Meeting.

(ii) Record Keeping and Review Reporting

For answer refer Unit-V, Q57, Topic: Concept of Record Keeping and Review Reporting.

(iii) Review Guidelines

For answer refer Unit-V, Q58, Topic: Review Guidelines.

Q60. Differentiate between peer and casual reviews.

Dec.-11, Set-1, Q6(b)

OR

Elaborate on peer reviews in detail.

(Refer Only Topic: Peer Review)

Dec.-11, Set-1, Q6(a)

Answer :

Peer Review

Peer review is a technique used in software testing. In this technique, the software product, code and documentation is tested by its developer as well as by his/her colleagues. This is done to remove any bugs early and efficiently. The testing is performed for evaluating the quality of product and its technical content. The peer review can be performed by any number of peers according to the wish of the producer. It is effective for detecting the bugs and errors in the early stages of software development.

Approaches Adapted in Peer Reviewing

- Peer reviews consist of several formal and informal approaches.
- Informal approach involves checking of software by colleagues (also called buddy check).
- Formal approach involves the technical peer review, walkthroughs and software inspections.

Advantages of Peer Reviews

- The peer reviews provide great improvement in the quality of software.
- A peer review process provides different forms of reviewing as inspection, team review, peer desk check, pair programming or passaround etc.
- Peer review is effective for reducing the project associated risks.
- Peer review like inspection is the best approach for reducing both low-level and high-level risks.

Types of Peer Review

The different types of peer reviews are as follows,

(i) Inspection

The inspection involves a well-defined multistage process. This process assigns specific roles to individual participants. It is performed in a systematic and rigorous way.

(ii) Team Reviews

The team review involves the simplification of the inspection. Here few stages are omitted and the role of the participants is combined. It is performed in a planned and structured way. It is less formal and less rigorous than inspections.

(iii) Walkthrough

Walkthrough review involves the description of the work product to the colleagues by the author. Here, dominant role is played by the author of the work product. It is performed in an informal way by not defining any procedure, not specifying existing criteria, not preparing management reports and not generating any metrics. It is a semi-formal meeting.

(iv) Pair Programming

Pair programming involves the development of the program by the two developers simultaneously at a single workstation. The work is continuously reviewed by them.

(v) Peer Deskcheck

Peer deskcheck involves the examination of the work product by only a single person besides the author. Inspite of being employing the defect check lists and specifying analysis methods, the peer deskcheck is considered to be an informal review.

(vi) Passaround

The passaround involves the examination of the work product by the multiple reviewers concurrently.

Casual Review

A casual review is an informal review which consists of few people. It is the least formal type of review. An informal review follows no documented process. In this review there isn't a structured meeting or a plan to carry on the process. An informal review goal is individual, depending on the author's need. Performing an informal review is very easy.

The informal review is not a formal process, it may be documented if the reviewers consider it necessary. (The material may be as informal as a computer listing or hand written documentation). It is the inexpensive way to get benefits. The date and time of the agenda for it will not be addressed in the plan.

5.3.5 Statistical Software Quality Assurance

Q61. Explain about statistical software quality assurance.

Answer : Nov./Dec.-12(R09), Q7(b)

Statistical Software Quality Assurance (SQA)

Following are few important steps favouring statistical quality assurance in case of software,

Step1: It initially begins with collection of information on defects occurring in a given software and categorizing them.

Step2: Cause of each defect is recognized.

Step3: Pareto Principle "80% of defects can be traced to 20% of all possible causes" is used. By implementing above principle 20% of the genuine causes for the errors are determined.

Step4: Apply suitable mechanisms on these 20% causes to correct them.

These steps are major ways of determining the genuine causes of creation of errors and hence, striving to lower them to larger extent.

In order to support the above mentioned consequence, consider the following example,

Example

Assume that, there is a software development organization '*A*', which has collected data on various types of errors by analyzing the software for certain period of time. Here, the errors can be the ones which were encountered during software development as well as the ones which were encountered after the software had been delivered to end users. Causes of an error can be any of the following cases.

Sl. No.	Abbreviation	Expansion
1.	MIS	Miscellaneous
2.	HCI	Inconsistent human/computer interface
3.	IID	Inaccurate or incomplete documentation
4.	PLT	Error in programming language translation of design
5.	IET	Incomplete or erroneous testing
6.	EDR	Error in data representation
7.	EDL	Error in design logic
8.	ICI	Inconsistent component interface
9.	MCC	Misinterpretation of customer communications
10.	VPS	Violation of programming standards
11.	IES	Incomplete or erroneous specifications
12.	IDS	Intentional deviation from specifications

Once causes are found, a table indicating the causes and their percentages of errors introduced by them is prepared. These percentages reveal that the causes EDR, IES and MCC contributes to about 53% introduction of errors in the overall project and the causes such as EDL, IES, PLT, EDR respectively correspond to introduce genuine errors. Hence, once such approximations are known various mechanism can be applied to curb the errors caused by them.

Hence, by using statistical quality assurance methods, organizations had observed drastic reduction in errors every year, few software development organization claimed referred about 50% reduction in errors.

Q62. Define six sigma (6σ). Give steps in it.

Answer :

Six Sigma

Six sigma (6σ) has now-a-days over taken almost all the methods for implementing statistical quality assurance and it is a leading methodology in many top ranked industries.

Six sigma is a planned approach implemented with a view of enhancing the given organization's operational performance by taking into consideration the observed data and statistical records. Thus, decreasing the short comings observed during production and service related processes to large extent.

Steps

Six sigma suggests following three steps which are also referred as core steps.

Step1: Initially gather the data related to customer requirements, their deliverables and finally estimate the required project goals in most sophisticated manner.

Step2: With an objective of obtaining the quality performance, compute the current proceedings and their outputs.

Step3: Obtain the vital causes of defects by closely analyzing defect metrics.

The above mentioned steps are referred as core steps. But to further improve current proceedings, following two steps are proved to be effective.

Step4: Improving the current process by removing major causes of bugs.

Step5: Controlling over the current process such that any changes made do not regenerate the causes of errors that were already removed.

Two more important steps are used by the organizations in which software development process is under progress.

Step6: Designing the process by considering customer requirements and curbing the core causes of defects.

Step7: Checking that the model developed satisfies its purpose of development.

Finally, it has to be noted that steps 1 to 5 are referred as DMAIC or Define Measure Analyze Improve and Control respectively and steps 6 and 7 as DMADV as Define Measure Analyze Design Very respectively.

5.3.6 Software Reliability

Q63. Discuss about measures of software reliability and software availability.

Answer :

Model Paper-I, Q11(a)

Software Reliability and Availability

Software reliability refers to the probability of software running with out any failure. The software failures mainly occur due to the design pattern followed or due to implementation problems but the software doesn't wear out like the hardware components. Software availability refers to probability that a program or software is available.

Measures

Measures of software reliability and availability are listed in the following table,

	Metric	Meaning	Example Systems
(i)	POFOD (Probability of Failure on Demand)	This is a measure of the likelihood that the system will fail when a service request is made. For example, a POFOD of 0.001 means that 1 out of 1000 service requests may result in failure.	Safety-critical and non-stop systems, such as hardware control systems.

(ii)	ROCOF (Rate of Failure Occurrence)	This is a measure of the frequency of occurrence with which unexpected behaviour is likely to occur. For example, ROCOF of 2.100 means that 2 failures are likely to occur in each 100 operational time units. This metric is sometimes called the failure intensity.	Operating systems, transaction processing systems.
(iii)	MTTF (Mean Time to Failure)	This is a measure of the time between observed system failures. For example, an MTTF of 500 means that 1 failure can be expected for every 500 time units. If the system is not being changed, it is the reciprocal of the ROCOF. Mean time failure is the measure of software reliability which is expressed as MTBF = MTTF + MTTR where MTTF is Mean-Time-to-Failure MTTR is Mean-Time-to-Repair	Systems with long transactions such as CAD systems. MTTF must be greater than the transaction time.
(iv)	AVAIL (Availability)	This is a measure of how likely the system is to be available for use. For example, an availability of 0.998 means that in every 1000 time units, the system is likely to be available for 998 time units. Availability of software is measured with the following expression. $\text{Availability} = \left(\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}\right) \times 100\%$	Continuously running systems such as telephone switching systems.

Table: Software Reliability and Availability Measures

Q64. Using a schematic diagram and suitable example to show the order in which the following are estimated in the COCOMO estimate technique: Cost, Effort, Duration, and Size. Nov./Dec.-18(R16), Q11

OR

Explain the COCOMO model for estimation.

Answer : March-17(R13), Q10(b)

COCOMO Estimation Technique

COCOMO stands for Constructive Cost Model which carries a set of software estimation models. It was developed by Barry Boehm in the year 1991. The basic form of this model was based on the following equation,

Effort (in person-months) = $c * (\text{size})^k$

Where c and k are constants whose values are different for different types of systems classified by Boehm.

Boehm estimated that software development of project can belong to any one of the following class depending on the complexity that exist in the development.

(i) Organic

(ii) Semidetached

(iii) Embedded.

Boehm wants every developer not only to analyze the characteristics but also analyze the development environment and team. Practically, if all the above three categories relates to application utility and system programs are considered then, the compilers and linkers are considered to be utility programs, while data processing programs are considered as application programs. System programs communicates directly with the hardware. It embraces meeting, timing constraints and concurrent processing. For example, real time systems, operating systems.

(i) **Organic**

In organic type category, a well understood application program is developed. The development team is assumed to be small but they possess experience as they worked on similar type of projects.

(ii) **Semidetached**

In semidetached type category both experienced and in experienced staff accompanies. In such cases the team members may have inadequate experience about projects or related systems or may not be familiar with the elements of the software being developed.

(iii) **Embedded**

In embedded type category, the software which is being developed is strongly connected to complex hardware. It may also contain rigid procedures or regulations which are to be followed while the development is ongoing.

Basic COCOMO Models

Basic COCOMO model is used for estimating the values of project parameters. This model doesn't give the actual estimate, but rather provides an approximate estimation. The expression for representing basic COCOMO model is.

$Eff = p_1 \times (\text{KLOC})^{p_2}$ Person/Month (P/M).

Time-dev $= q_1 \times (\text{Eff})^{q_2}$ months

Here,

KLOC is the estimated size of the product and is expressed in kilo lines of code.

p_1, p_2, q_1, q_2 are the constants

Time-dev is the estimated time required to develop the product and it is expressed in months.

Eff is the effort required to develop the product and it is expressed in Person/Month (P/M).

Boehm proposed that, without taking into account the instructions count on the line, every individual line of source text must be computed as a single LOC. For instance, if an instruction consists of x number of lines, it is calculated as x * LOC.

Effort Estimation

The formulas for estimating the effort based on code size for the three classes of software products are,

(i) **Organic Type Class Effort**

$\text{Eff} = 2.4\ (\text{KLOC})^{1.05}$ P/M

(ii) **Semi Detached Type Class**

$\text{Eff} = 3.0\ (\text{KLOC})^{1.12}$ P/M

(iii) **Embedded Type Class**

$\text{Eff} = 3.6\ (\text{KLOC})^{1.20}$ P/M

Time Estimation

The formulas for estimating the development time depending on effort for the below three classes of software and products are,

(i) Organic Type Class

Time-dev = 2.5 $(Eff)^{0.38}$ months

(ii) Semidetached Type Class

Time-dev = 2.5 $(Eff)^{0.35}$ months

(iii) Embedded Type Class

Time-dev = 2.5 $(Eff)^{0.32}$ months.

Now, plot the estimated characteristics for different size of softwares.

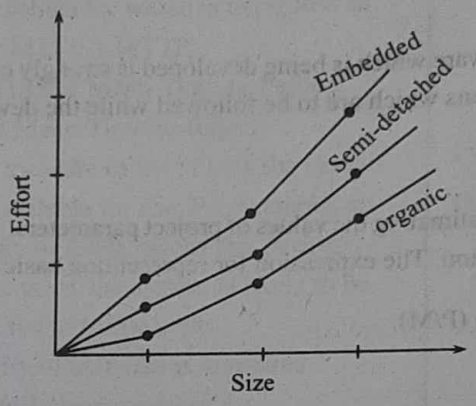


Figure: Effort and Size

From the above graph, it can be inferred that the effort and size are directly proportional to each other. That is, the effort increases with the increase in the product size.

Now, plot a graph against time to develop the product and its size.

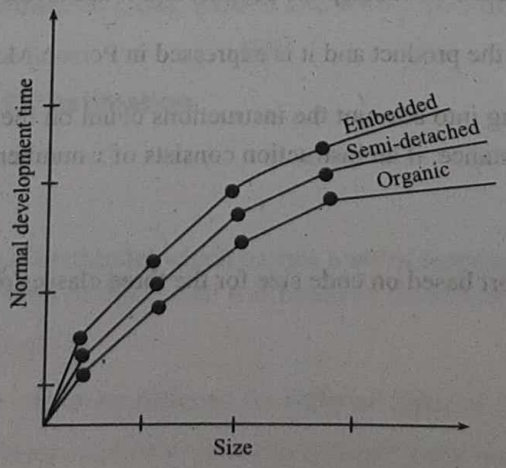


Figure: Time and Size

From the above graph, it can be noticed that development time is sublinear function to the size of the software product. That is, increase in size of the product by two times doesn't mean that time taken to develop a product is doubled, rather the development time rises gradually, but not at once. Development of large software products can be accomplished concurrently. That is, software developers carry out development activities parallely which reduces the time needed to successfully finish the project.

Moreover, it can also be observed that the amount of development time taken by all the three classes of product are same. Project cost is obtained by multiplying effort with manpower cost per month. Because people usually consider that in the entire development of software products, manpower cost is the only cost that have been incurred but it is not the fact. There are many other costs that incurs while developing the software such as hardware, software, administration, office space.

It is significant to understand that efforts and estimations which are acquired on the basis of COCOMO model are nominal effort estimate and nominal duration estimate. Here, nominal indicates that one should try to finish the project in the prescribed duration and cost. Because if the project is completed in a much shorter span then the cost will increase. On the other hand, if the project takes much longer time than the nominal. Then the cost will not be decreased, it remains the same.

Example

Consider that the size of an organic type of software product has been estimated to be 32,000 lines of source code. The average salary of software engineers is Rs 15,000/- per month. Determine the effort required to develop the software product and the normal development time.

Solution

Given that,

The size of an organic type software product = 32,000 lines.

Average salary of an engineer = 15,000

(i) Estimation of Development Effort

The basic COCOMO estimation for organic type is,

$$\text{Effort} = 2.4\,(\text{KLOC})^{1.05}\ \text{PM}$$

$$= 2.4 \times (32)^{1.05}$$

$$= 2.4 \times 38.05$$

$$= 91.3\ \text{PM}$$

(ii) Estimation of Development Time

Nominal development time,

$$T_{dev} = 2.5\,(\text{Effort})^{0.38}\ \text{month}$$

$$= 2.5 \times (91)^{0.38}$$

$$= 2.5 \times 5.5518$$

$$= 13.87$$

$$\approx 14\ \text{months}$$

(iii) Estimation of Cost

Cost incurred to develop the product = (Months × Average salary of an employee)

$$= 14 \times 15{,}000$$

$$= ₹\ 210{,}000/-$$

5.3.7 The ISO 9000 Quality Standards

Q65. Write a detailed note on ISO 9000 quality standards.

(Model Paper-I, Q11(b) | Nov./Dec.-16(R13), Q11(a))

Answer :

Basic elements of ISO 9001:2000 are,

1. Create a quality management system
2. Report the quality management system
3. Specify a quality policy which depicts the importance of the system
4. Make quality improvements
5. Carry out the management reviews
6. Create record keeping methods
7. Meet the customer needs
8. Improve methods for updating documents
9. Authorize the operational activities
10. Handle the project monitoring devices etc.

To assure the quality of product along with its services, quality assurance system is preferred as it ensures that the product is made exactly according to the needs of the customer. This type of system can be used in any kind of business. Moreover there are certain quality standards such as ISO 9000 that verifies the major parts of the organization as well as the product to ensure its quality

ISO 9001:2000 standards carry almost 20 requirements to be met for quality assurance of the product. As these standards apply to any business, there exist certain special guidelines to ensure the quality of a software process.

For getting ISO 9001 standard, organization must describe the policies and procedures with respect to each of the following requirements.

The following are the areas covered by the ISO 9001 model for quality assurance,

1. Software process control
2. Management responsibility
3. Contract review
4. Quality system
5. Design control
6. Internal quality audits
7. Control of quality records
8. Servicing
9. Training
10. Statistical techniques
11. Product identification and traceability
12. Document and data control
13. Inspection and testing
14. Corrective and preventive action.

After getting registered with these standards, the organization is considered as certified and issued with a certificate. The company is surveyed twice every year after getting certified.