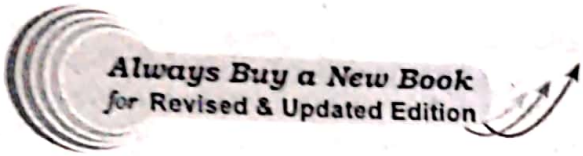


STUDENT'S FRIENDLY Edition

Software Engineering

B.E. V-Sem (Computer Science and Engineering)

OSMANIA UNIVERSITY (Hyderabad)

[AS PER AICTE MODEL CURRICULUM FOR THE ACADEMIC YEAR 2020-2021]

PROFESSIONAL PUBLICATIONS™

The Destination Towards Knowledge & Success

Head Office :

#12-2-825/826, Safa Arcade Building, 4th Floor, Door No. 562, Pillar No. 15, Above AXIS Bank, Mehdipatnam, Hyderabad - 500028, Telangana, India.

Phone : 040 - 6663 1899. **Mobile :** 7893 48 9991

e-mail : professionalpublications99@gmail.com

CONTENTS

Syllabus as per 2020-21 Curriculum

UNIT-WISE SHORT & ESSAY TYPE QUESTIONS WITH SOLUTIONS

Unit/Topic Number(s)	Unit/Topic Name(s)	Question Nos.	Page Nos.
UNIT - 1	**INTRODUCTION SOFTWARE ENGINEERING, PROCESS MODELS AND AN AGILE VIEW OF PROCESS**	**Q1 - Q65**	**1 - 40**
Part-A	**SHORT QUESTIONS WITH SOLUTIONS**	**Q1 - Q20**	**1 - 5**
Part-B	**ESSAY QUESTIONS WITH SOLUTIONS**	**Q21 - Q65**	**6 - 40**
1.1	Introduction to Software Engineering	Q21 - Q26	6 - 10
1.2	A Generic view of process		
1.2.1	Software Engineering	Q27	11
1.2.2	Process Frame Work	Q28 - Q30	11 - 12
1.2.3	CMM Process Patterns	Q31 - Q35	13 - 17
1.2.4	Process Assessment (ש.ע.פ)	Q36	18
1.3	Process Models : Prescriptive Models		
1.3.1	Waterfall Model	Q37	18 - 19
1.3.2	Incremental Process Models	Q38 - Q40	20 - 21
1.3.3	Evolutionary Process Models	Q41 - Q49	22 - 27
1.3.4	Specialized Process Models	Q50	28
1.3.5	The Unified Models	Q51 - Q52	29 - 31
1.3.6	Personal and Team Process Models	Q53	32
1.3.7	Process Technology	Q54	33
1.4	An Agile View of Process		
1.4.1	Introduction to Agility and Agile Process	Q55 - Q58	33 - 34
1.4.2	Agile Process Models	Q59 - Q65	35 - 40
UNIT - 2	**SOFTWARE ENGINEERING PRINCIPLES, SYSTEM ENGINEERING AND REQUIREMENTS ENGINEERING**	**Q1 - Q45**	**41 - 70**
Part-A	**SHORT QUESTIONS WITH SOLUTIONS**	**Q1 - Q10**	**41 - 42**
Part-B	**ESSAY QUESTIONS WITH SOLUTIONS**	**Q11 - Q45**	**43 - 70**
2.1	Software Engineering Principles		
2.1.1	SE Principles, Communication Principles, Planning Principles	Q11 - Q13	43 - 45
2.1.2	Modeling Principles, Construction Principles, Deployment	Q14 - Q16	46

2.2	System Engineering						
	2.2.1 Computer – Based Systems			Q17			47
	2.2.2 The System Engineering Hierarchy			Q18			48
	2.2.3 Business Process Engineering	Q19	-	Q20			49
	2.2.4 Product Engineering			Q21			50
	2.2.5 System Modeling	Q22	-	Q28	51	-	57
2.3	Requirements Engineering						
	2.3.1 A bridge to Design and Construction			Q29			58
	2.3.2 Requirements Engineering Tasks			Q30			58
	2.3.3 Initiating Requirements Engineering Process, Eliciting Requirements, Developing Use Cases	Q31	-	Q43	59	-	66
	2.3.4 Building The Analysis Model, Negotiating Requirements, Validating Requirements	Q44	-	Q45	67	-	70
UNIT - 3	**BUILDING THE ANALYSIS MODEL AND DESIGN ENGINEERING**	**Q1**	**-**	**Q46**	**71**	**-**	**104**
Part-A	**SHORT QUESTIONS WITH SOLUTIONS**	**Q1**	**-**	**Q8**	**71**	**-**	**72**
Part-B	**ESSAY QUESTIONS WITH SOLUTIONS**	**Q9**	**-**	**Q46**	**73**	**-**	**104**
3.1	Building The Analysis Model						
	3.1.1 Requirements Analysis Modeling Approaches			Q9			73
	3.1.2 Data Modeling Concepts			Q10			74
	3.1.3 Object-Oriented Analysis	Q11	-	Q12	74	-	75
	3.1.4 Scenario-Based Modeling	Q13	-	Q15			76
	3.1.5 Flow-Oriented Modeling	Q16	-	Q19	77	-	80
	3.1.6 Class – Based Modeling	Q20	-	Q24	81	-	83
	3.1.7 Creating a Behavioral	Q25	-	Q26	84	-	85
3.2	Design Engineering						
	3.2.1 Design Within the Context of SE			Q27	86	-	87
	3.2.2 Design Process and Design Quality	Q28	-	Q33	88	-	90
	3.2.3 Design Concepts	Q34	-	Q39	91	-	95
	3.2.4 The Design Model	Q40	-	Q44	96	-	101
	3.2.5 Pattern-Based Software Design	Q45	-	Q46	102	-	104

UNIT - 4	**CREATING AN ARCHITECTURAL DESIGN, MODELING COMPONENT LEVEL DESIGN, PERFORMING USER INTERFACE DESIGN**	**Q1 - Q43**	**105 - 140**	
Part-A	**SHORT QUESTIONS WITH SOLUTIONS**	**Q1 - Q10**	**105 - 107**	
Part-B	**ESSAY QUESTIONS WITH SOLUTIONS**	**Q11 - Q43**	**109 - 140**	
4.1	Creating an Architectural Design			
	4.1.1 Software Architecture	Q11 - Q13	109 - 110	
	4.1.2 Data Design	Q14 - Q15	111 - 112	
	4.1.3 Architectural Styles and Patterns	Q16 - Q17		112
	4.1.4 Architectural Design	Q18 - Q21	113 - 116	
4.2	Modeling Component-Level Design			
	4.2.1 Definition of Component	Q22		117
	4.2.2 Designing Class – Based Components	Q23 - Q27	118 - 121	
	4.2.3 Conducting Component – Level Design	Q28		122
	4.2.4 Object Constraint Language	Q29		123
	4.2.5 Designing Conventional Component	Q30 - Q31	123 - 127	
4.3	Performing User Interface Design			
	4.3.1 The Golden Rules	Q32 - Q34	128 - 129	
	4.3.2 User Interface Analysis And Design	Q35 - Q36	130 - 131	
	4.3.3 Interface Analysis	Q37 - Q41	132 - 136	
	4.3.4 Interface Design Steps	Q42	137 - 138	
	4.3.5 Design Evaluation	Q43	139 - 140	
UNIT - 5	**TESTING STRATEGIES DEBUGGING, PRODUCT METRICS AND SOFTWARE QUALITY**	**Q1 - Q85**	**141 - 192**	
Part-A	**SHORT QUESTIONS WITH SOLUTIONS**	**Q1 - Q13**	**141 - 143**	
Part-B	**ESSAY QUESTIONS WITH SOLUTIONS**	**Q14 - Q85**	**144 - 192**	
5.1	Testing strategies			
	5.1.1 A Strategic Approach to Conventional Software Testing	Q14 - Q28	144 - 152	
	5.1.2 Test Strategic For O – 0 Software	Q29		153
	5.1.3 Tactics : Software Testing Fundamentals	Q30		154

	5.1.4	Black Box And White Box Testing	Q31	-	Q39	154	- 160
	5.1.5	Basis Path Testing	Q40	-	Q43	161	- 163
	5.1.6	Control Structure Testing	Q44	-	Q45		164
	5.1.7	O – O Testing Methods	Q46	-	Q49	165	- 166
5.2		Debugging : Debugging Techniques, The Art of Debugging	Q50	-	Q52		167
5.3		Product Metrics					
	5.3.1	A Framework for Product Metric	Q53	-	Q56	168	- 169
	5.3.2	Metrics for each Phase of Software Development	Q57	-	Q70	170	- 181
5.4		Software Quality : Definition	Q71	-	Q73	182	- 183
	5.4.1	Quality Assurance : Basic Elements	Q74	-	Q77	184	- 186
	5.4.2	Formal Approaches			Q78		187
	5.4.3	Statistical Software Quality Assurance	Q79	-	Q80	187	- 188
	5.4.4	Software Reliability	Q81	-	Q82		189
	5.4.5	ISO 9000 Quality Standards	Q83	-	Q84	190	191
	5.4.6	SQA Plan			Q85		192
		UNIT WISE IMPORTANT QUESTIONS				IQ.1	- IQ.3

Model Question Papers with Solutions (As per the New External Exam Pattern)

Model Paper-1	MP.1 - MP.2
Model Paper-2	MP.3 - MP.4
Model Paper-3	MP.5 - MP.6

UNIT 1

INTRODUCTION TO SOFTWARE ENGINEERING, PROCESS MODELS AND AN AGILE VIEW OF PROCESS

PART-A

SHORT QUESTIONS WITH ANSWERS

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

Answer :

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 :

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.

Answer : Model Paper-I, Q1

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

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

	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.

Answer : Model Paper-III, Q1

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, Q2

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.

   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 :

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, Q2

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,

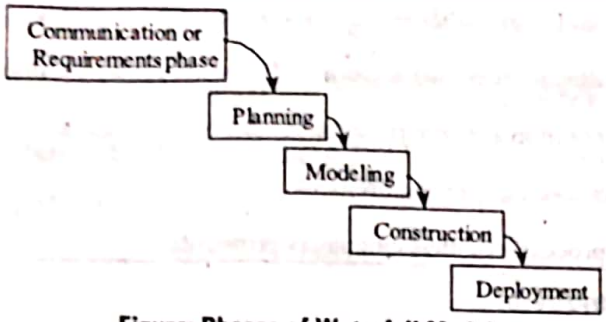


Figure: Phases of Waterfall Model

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, Q2

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 :

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 :

	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 :

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 :

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 :

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 ANSWERS

1.1 INTRODUCTION TO SOFTWARE ENGINEERING

Q21. Explain the evolving role of software.

Answer : Model Paper-I, Q11(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.

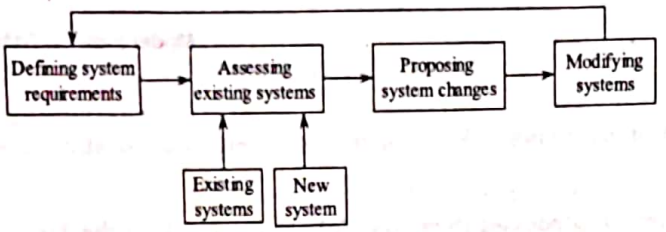


Figure: Software Evolutionary Process

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.

Q22. Define the term Software. Describe its various characteristics.

Answer :

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,

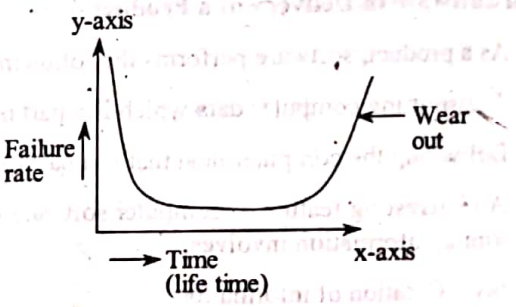


Figure (i): Graph Depicting the Failure Curve for Hardware

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.

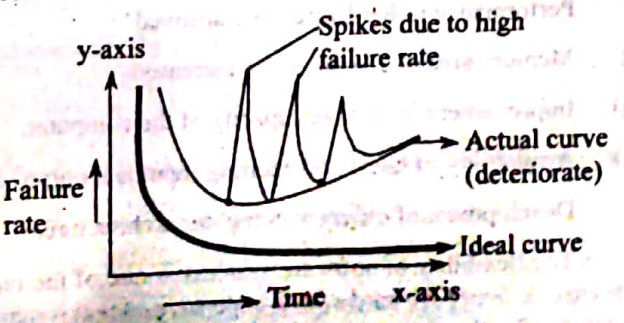


Figure (ii): Software Failure Curve

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.

Answer :

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.

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 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 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.

Q26. Explain the various software myths.

Answer : Model Paper-II, Q11(a)

Software Myths

Software Myths are the beliefs 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

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

Answer :

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

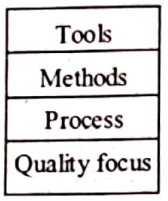


Figure: Layers of Software Engineering

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 Process Frame Work

Q28. What do you mean by process framework? Explain the five generic process framework activities.

Answer : Model Paper-III, Q11(a)

Diagrammatic representation of software process framework is given in figure,

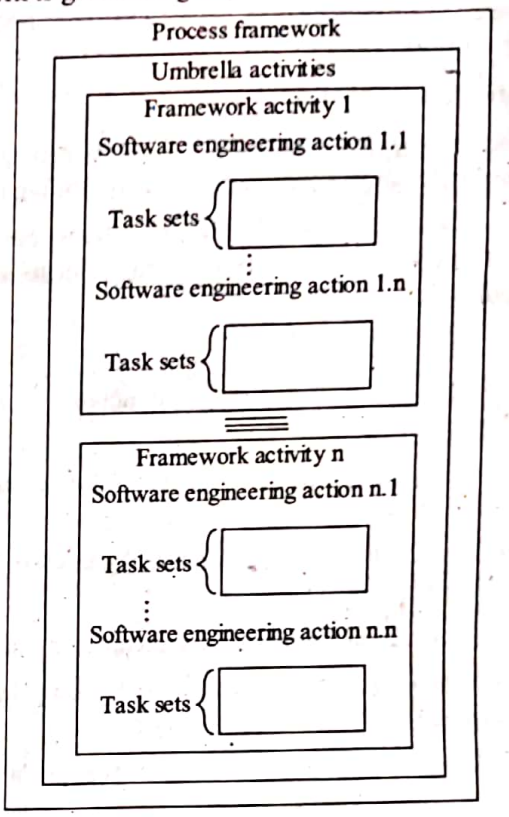


Figure: Process Single Word

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 CMM Process Patterns

Q31. What is CMM? Discuss how various maturity levels of CMM can be measured.

Answer : Explain

Capability Maturity Model (CMM) 7 Marks

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,

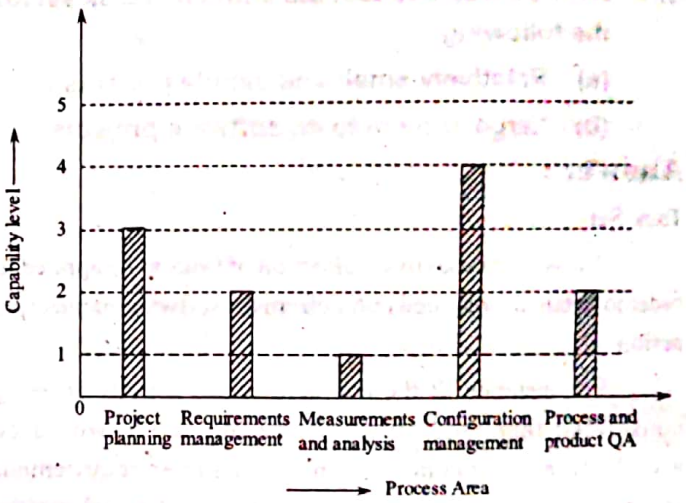


Figure: Graph Depicting the Process Area Capability Scenario

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 :

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 level. (CMM levels) L1(a)

Answer :

Process Maturity Levels

For answer refer Unit-I, Q31.

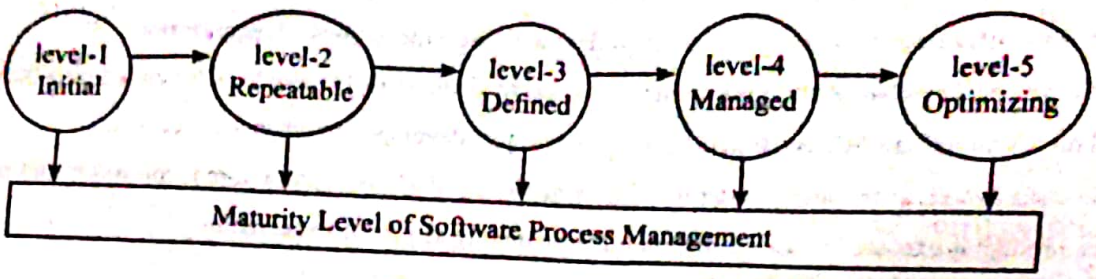


Figure: Levels of Maturity

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.

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?

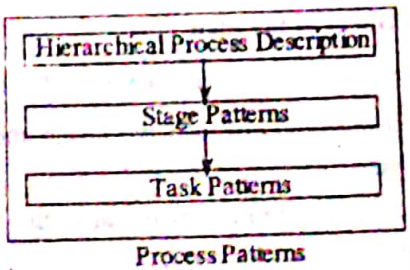
Related Patterns

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,



1.2.4 Process Assessment

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

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.

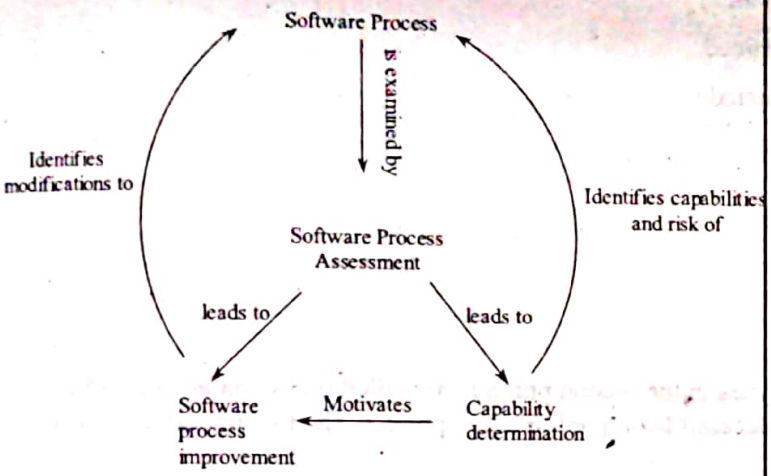


Figure: Software Process and Methods Applied for Assessment and Improvement

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/IEC15504)

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 : 200 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.

1.3 PROCESS MODELS : PRESCRIPTIVE MODELS

1.3.1 Waterfall Model

Q37. What is waterfall model? How is it different from other engineering process models?

Answer : Model Paper-I, Q11(b)

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,

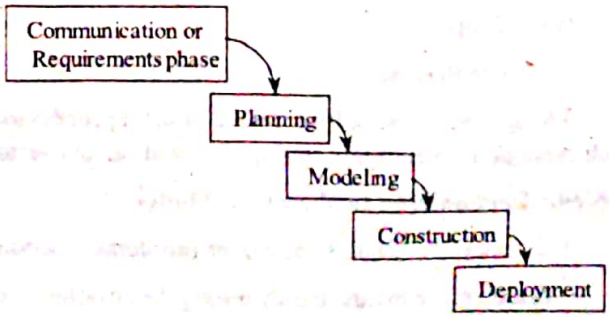


Figure: Phases of Waterfall Model

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

Q38. Describe the incremental software development process model.

Answer :

Incremental Model

Following is a diagram depicting an overview of incremental model.

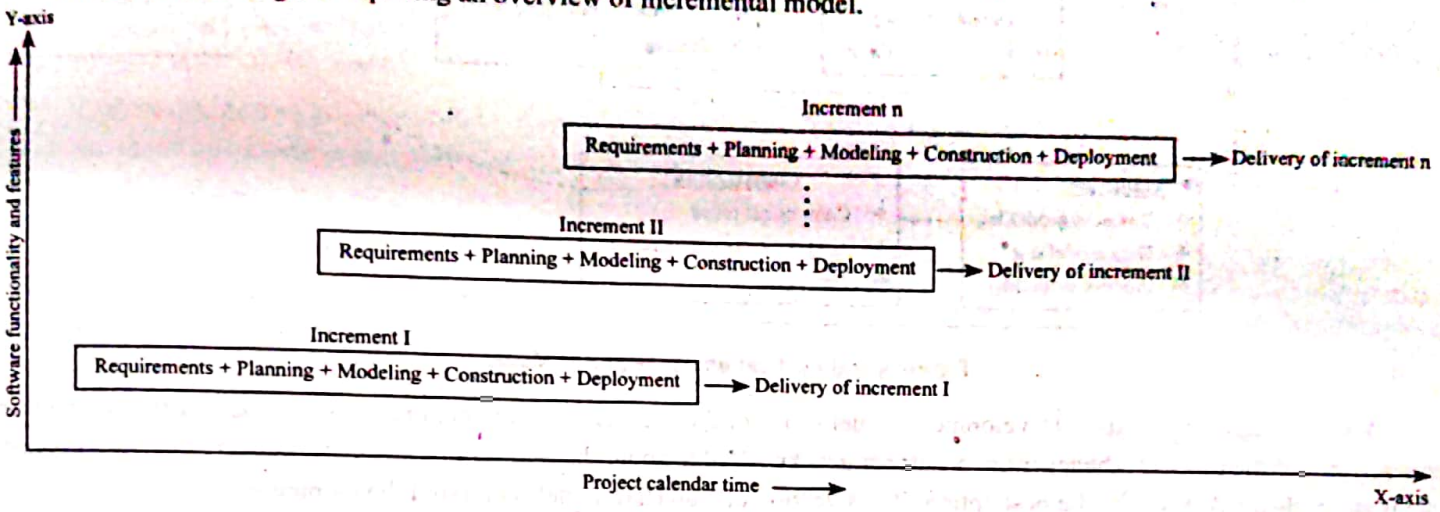


Figure: Representation of Incremental Model

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.

Q39. Illustrate on RAD process model.

Answer :

RAD (Rapid Application Development) Model

Diagrammatic representation of RAD model is given below.

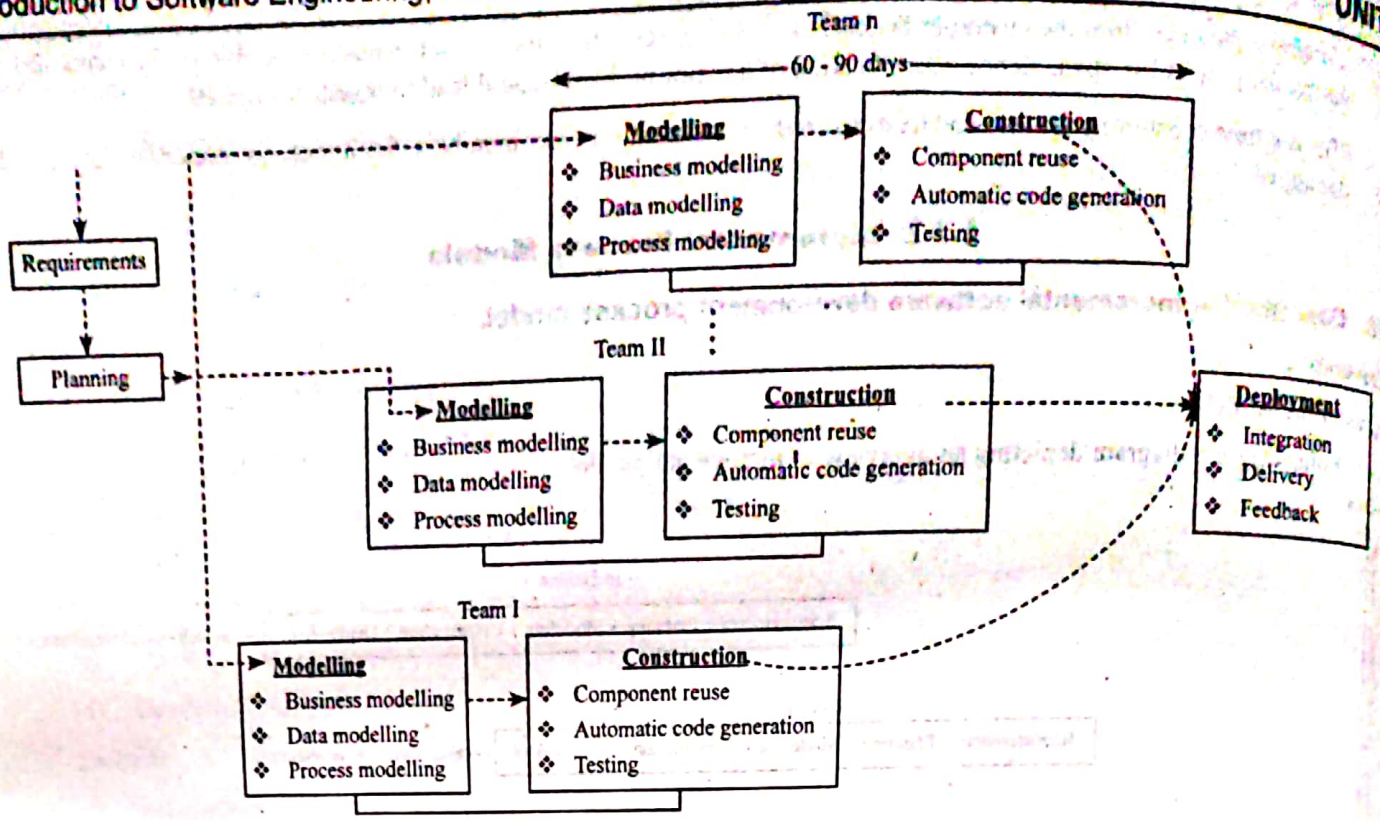


Figure: Rapid Application Development Model

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.

Q40. Explain the prerequisites for applying RAD model.

Answer :

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

Q41. 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 "tuned" to satisfy the needs of the customer along with enabling the developer to better understand about what needs to be done.

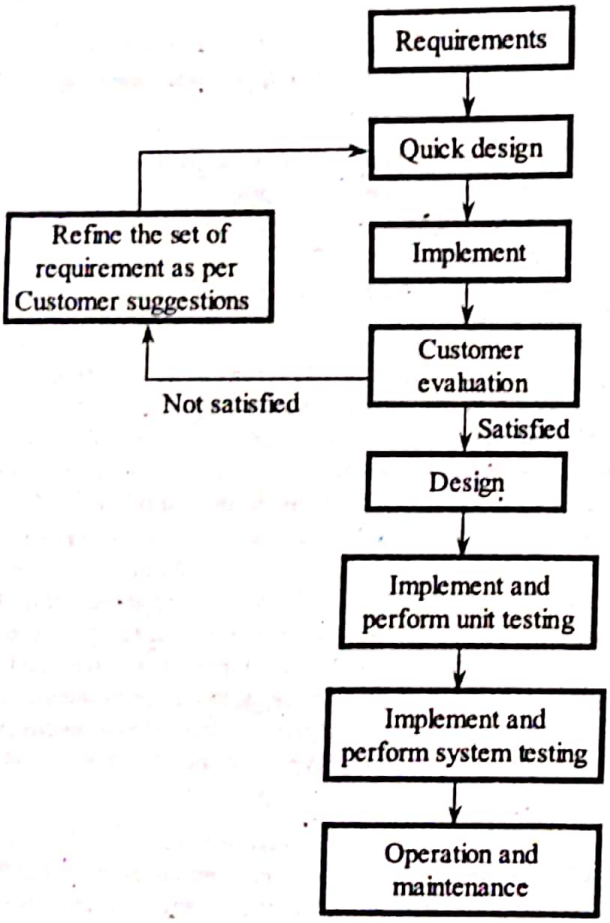


Figure: Prototyping Software Life-Cycle

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.

Q42. 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?

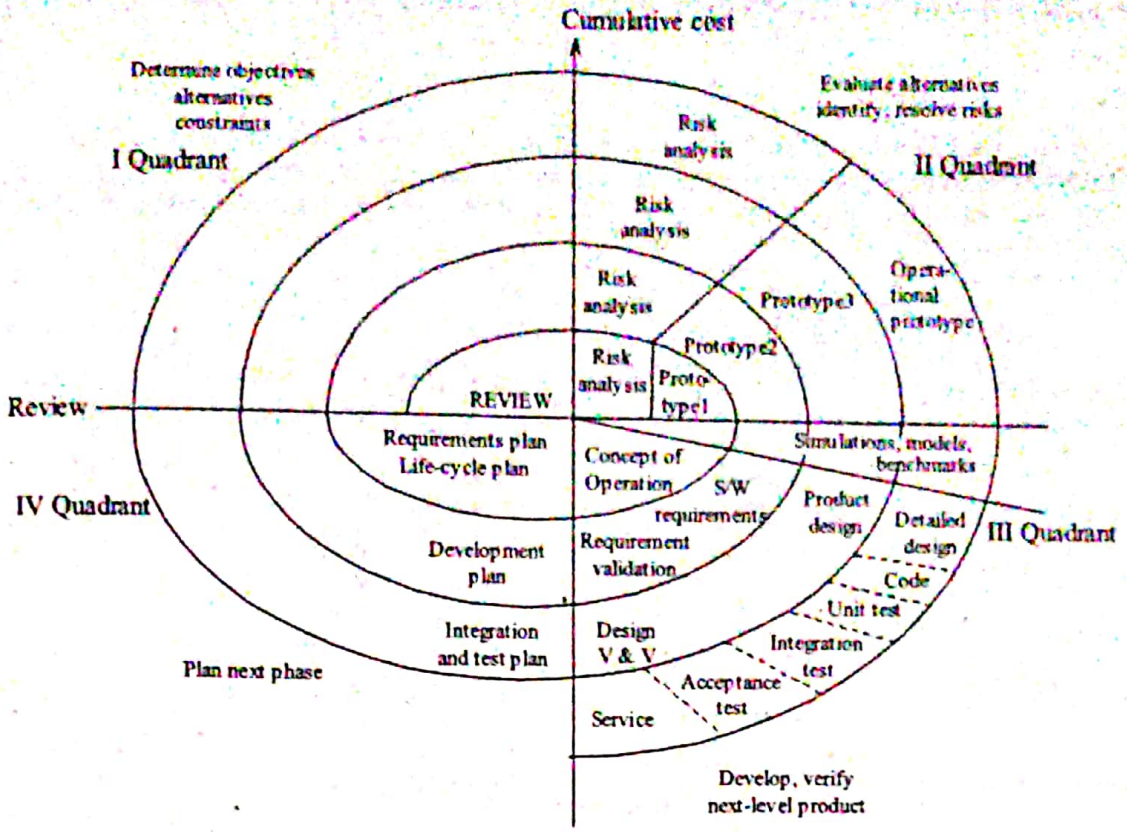
Answer : Model Paper-III, Q11(b)

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.

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.

Q43. Discuss the suitability of concurrent development model for system engineering projects.

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.

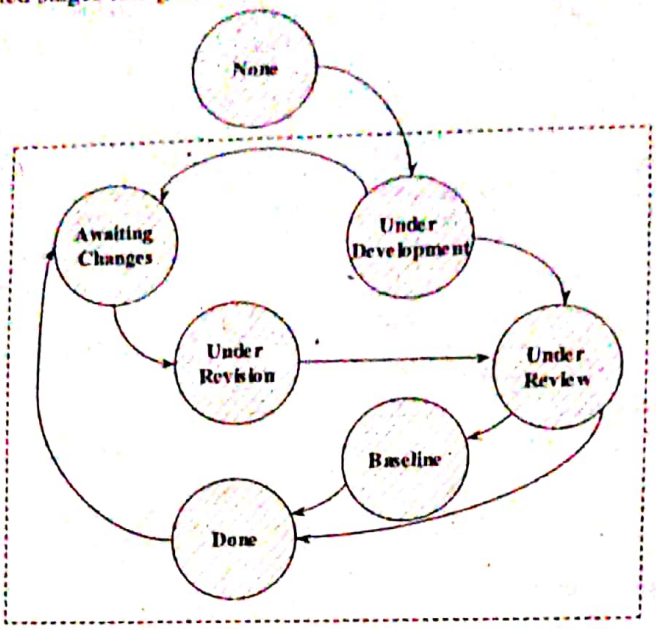


Figure: Stages of Software Engineering Tasks

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.

Q44. Compare and contrast between waterfall model and spiral model with neat diagrams.

Answer :

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, Q37.

Spiral Model

For answer refer Unit-I, Q42.

Q45. What is an evolutionary process model? Differentiate between evolutionary and incremental process models.

Answer : Short note (a)

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, Q41.

2. Spiral Model

For answer refer Unit-I, Q42.

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.

Q46. Explain the working of specialized process models.

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, Q47.

2. Formal Methods Model

For answer refer Unit-I, Q48.

3. Aspect-oriented Software Development

For answer refer Unit-I, Q49.

Q47. 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.

Q48. 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.

Q49. 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 Specialized Process Models

Q50. What are specialized process models? Explain in brief about component based model.

Answer :

Specialized Process Model S.N

Specialized Process models are the models that acquire the features of traditional models like waterfall model, prototyping model, iterative model etc. These models are used when specialized or sophisticated software development methods are adopted.

Specialized process models are as follows,

1. Component based development model.
2. Formal methods model.
3. Aspect oriented software development model.
4. Unified process model.

Component Based Model

Component based development model 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, webservices or modules that provide specific (required) functionality with well defined interfaces that enable the components to be easily embedded into the software.

The foremost 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 technology,

(i) Conventional software module

(ii) Object-oriented class

(iii) Package.

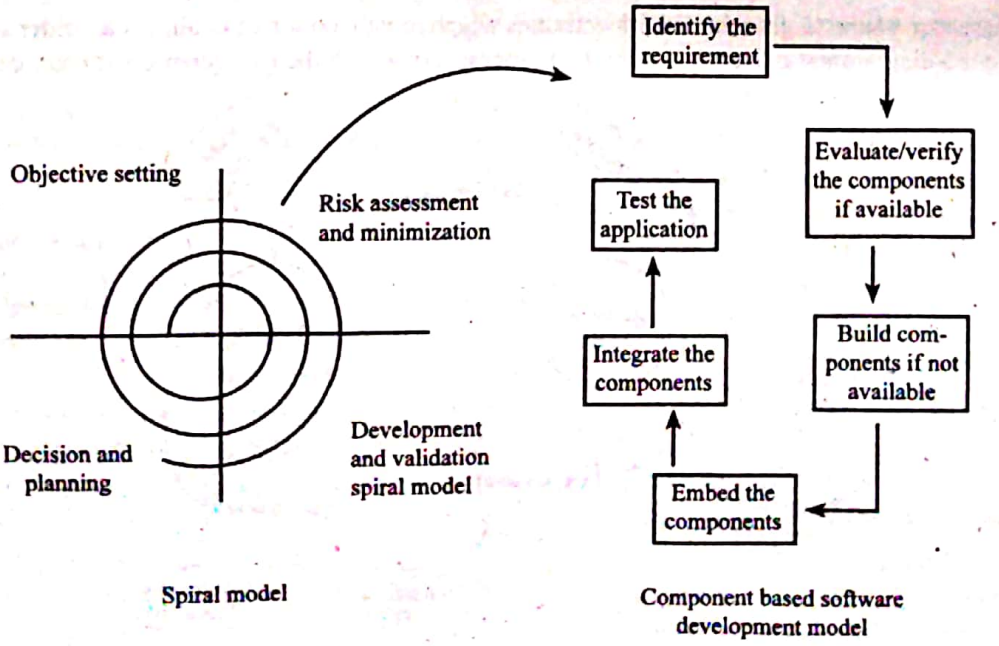


Figure : Component based model

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, a research evaluation is performed on the component-based products such that they fit into the need of application domain, which is 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).

Advantages of Component Based Development

(i) One of the major advantage of the component based development is its reusability factor

(ii) Complexities in development process are effectively managed in component based development

(iii) Time-to-market is significantly less as initial component is dropped in the market for the purpose of testing.

Disadvantages of Component Based Development

(i) It is difficulty to choose a better component among components performing similar functions

(ii) It is difficult to maintain the quality of individual components, when different component are interfaced and merged together.

1.3.5 The Unified Models

Q51. Give an overview of different phases of unified process model.

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.

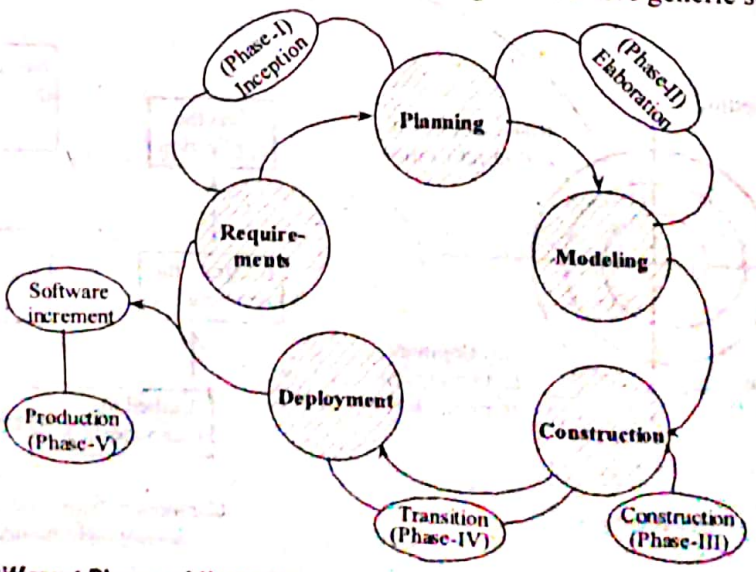


Figure: Different Phases of Unified Process along with Five Generic Software Activities

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.

Q2. What is meant by unified process? Elaborate on the unified process work products.

Answer :

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.

1.3.6 Personal and Team Process Models

Q53. Compare the personal and team process models.

Answer :

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.

1.3.7 Process Technology

Q54. Write short notes on Software Process, Software Product and Process Technology tools.

Answer :

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 Products

Generally, a software process is divided into a set of different projects and the projects in turn may be divided into software products. A software project can be developed by using a software development process and the outcome of projects are referred to as software products. Each project is initiated with some requirements such that these requirements must be fulfilled in order to achieve a good software and the software process should also satisfy the user needs. The main act of executing the activities is referred to as a software project and the output that results during the execution of a software are referred to as products. A software process is seen as an abstract type that can have many projects associated with it and a project can also have many products produced in it, which is shown in the figure.

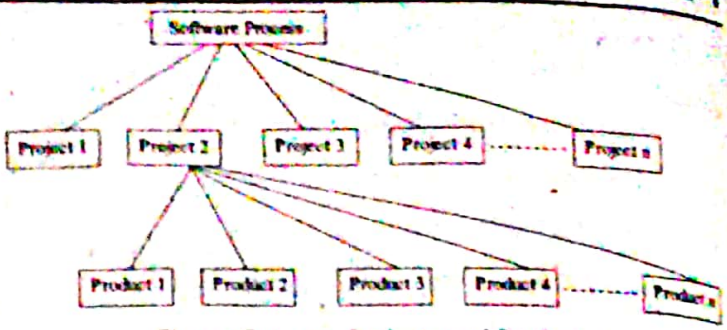


Figure: Process, Projects and Products

Process Technology Tools: Process technology tools were developed to assist the software developers in representing the primary elements that are required for a process such as categorization of work tasks, analyze the processes, control and monitor progress and manage technical quality. These tools even helps in organizing the usage of additional software engineering tools that are suitable for the particular work task.

1.4 AN AGILE VIEW OF PROCESS

1.4.1 Introduction to Agility and Agile Process

Q55. Briefly write about Agility.

Answer : Model Paper-II, Q11(a)

The Agility can be defined as the ability of being quick. It is basically referred to as nimbleness, the which is again ability of efficient modification of body's position or capability of responding to changes. This requires the individual combination of skills such as balance, coordination, speed, reflexes, strength.

Agility is essential in modern software process such as changes in developing a software, changes in selecting the team members, changes caused due to new technology, all types of changes effecting the product or project created by the developers. Also, the agile team makes a response to the changes by determining whether or not the software is created by a single person working in teams. Consequently, they focus upon people skills and their effectiveness to collaborate for the success of the project.

Further, the idea of agility is more than just responding to changes. This includes motivating team structures and behavior for conducting interaction among members, among technologists and business people and between software engineers. Apart from this, it focuses upon quick delivery of operational software and does not include the intermedi work products. On the other hand it assumes customer as an important entity of the development team and strives to remove the "us and them" behavior. Also, it enables the software developers to plan a flexible project plan.

Any software process can manage the concept of agility. But, in order to fulfill this, design of the project must be such that it should permit all the members of the project not just to carry-out the tasks but also to organize them and develop plans essential for understanding the flexibility of the agile development approach. Aside this, it also performs the deletion of necessary work products, generating incremental delivery strategy for handing over the working software to the customer as soon as possible.

Q56. What is an Agile Process?

Answer :

Agile Process

Agile processes are the processes that are evolved in response to the documentation and bureaucracy-based processes like the waterfall model. Depending upon the following principles, agile approaches build an optimum project.

1. An efficient software model must be chosen for the progress of a project.
2. The software must be build and delivered quickly in small iterations.
3. The software must be capable of accepting the changes in requirements even in the later stages of software development.
4. Face-to-face communication between the clients or customers must be done rather than preparing the documentation.
5. The customers involvement and their continuous feedbacks about the software is essential for producing high quality software.
6. A simple design that evolves and improves regularly over a period of time must be considered for managing the different scenarios rather than the detailed design.
7. The empowered development teams must decide the delivery time of the software.

The agile process solves many key assumptions for many software processes.

1. It is not easy to determine what software requirements will stay and what software requirements will change during the project development. In a way similar to this, the changes in customer priorities are difficult to determine.
2. It is not easy to determine the extent of design prior to performing the construction phase (which proves the design). In simple terms, the design and construction of many softwares is left blank and then both the activities are carried out in team so as to prove the design model.
3. It is not easy to determine analysis, designing, construction and testing as it appears to be.

The above assumptions give rise to uncertainty such as "How the developers create process which can control unpredictability". However it can be resolved by incorporating adaptability. Hence, agile process must show adaptability.

A continuous adaptation produces many problems, so an incremental adaptability is recommended in agile software process. Now, the incremental adaptation is procured by an agile team which basically works upon the feedback given by customers. Many factors contribute to escalate the customers feedback such as operational prototype or a part of operational system. Therefore, incremental development strategy must be necessarily implemented.

Q57. List Agile Principles.

Answer :

Agile Principles

The principles of Agile development are as follows,

1. Satisfy the customer by providing quick and continuous software services.
2. Adopt the changing requirements in the development process.
3. Develop the working software which is central for the progress of the system.
4. Provide working software rapidly beginning from the initial weeks will final months by scheduling a short time period.
5. Active and collective hardwork and participation of the developers and business people throughout the project.
6. Direct and face-to-face conversation within the development team.
7. Provide a proper environment that motivates the developers to meet all the requirements.
8. Encourage sustainable development .
9. Improve the project by developing a good design and technology.
10. Maintain a constant pace.
11. Develop the best architecture, design and requirements for self organizing teams.
12. Develop an effective system by modifying and enhancing it regularly depending upon the changes.

Q58. Discuss the importance of human factors in agile software development.

Answer :

The 'Human Factors' or 'People Factors' is given utmost importance by the supporters of agile software development. In simple terms, the factors such as ability, expertise, creativity and skills of the personnel are focussed in order to change the process into a particular people and team. Thus changing the process according to the people and teams requirements is essential.

If the characteristics of the process are deduced by the members of software team then one or many key traits are found in people in agile team and team itself. They are,

(i) Competence
(ii) Common Focus
(iii) Collaboration
(iv) Decision-making ability
(v) Fuzzy problem-solving ability
(vi) Mutual trust and respect

i) Self organization.

Competence: The competence under the frame of reference of agile development focuses upon the inherent talent, creativity, skills associated with a particular software and the complete information of the processes which the team decides to use. Also, it is necessary that every member of the agile team should be aware of skill and knowledge of the processes.

(ii) **Common Focus:** The common focus under the frame of reference of agile development may emphasize that all members of the agile team undertake many different tasks and apply different skill to the project. They mainly focus upon the idea of deliverying the working software increment to the customer within the specified time period. This is achieved by consistent adaptation in order to make the process suitable for accomplishing the team requirement.

(iii) **Collaboration:** The collaboration under the frame to reference of agile development may emphasize upon evaluation, analysing and making use of information for interacting with software team. It also focuses upon generating information helpful for stakeholders and for business value. These can be procured by collaborating upon team members.

(iv) **Decision Making Ability:** The ability of decision making under the frame of reference of agile development refers that the software team must be given authority to make decisions upon technical and non-technical project issues.

(v) **Fuzzy Problem-Solving Ability:** The ability of fuzzy problem solving in the frame of reference of agile development is concerned with the job of the software managers to identify that the agile team must consistently handle changing errors. Also, in some scenarios the team realizes that, the problem getting solved now may not be the same in future. Thus, these experiences benefits the problem solving team in later years.

(vi) **Mutual Trust and Respect:** The mutual trust and respect under the frame of reference of agile development is concerned with the jelled team which displays certain properties called trust and respect which in turn make them strong, signifying that complete is greater than sum of the parts.

(vii) **Self-Organization:** The Self-organization under the frame of reference of agile team indicates three things,

(a) The agile team must streamline in order to accomplish the task.

(b) The team must streamline the process inorder to incorporate local environment.

(c) The team must streamline the schedule of the task to attain efficient delivery of software increment.

The self-organization enhances collaboration and team morale.

1.4.2 Agile Process Models

Q59. Explain in detail various other Agile Process Models.

Answer :

The various other Agile Process Models are,

1. Adaptive Software Development(ASD)
2. Scrum
3. Dynamic System Development Method (DSDM)
4. Crystal
5. Feature Drive Development(FDD)
6. Agile Modelling(AM)
7. Agile Unified Process(AUP).

1. **Adaptive Software Development (ASD):** The Adaptive Software Development (ASD) was proposed by Jim High Smith. It is a technique for developing complex softwares, systems and accentuates human collaboration and self-organization of the team members. There exist three phases in this model. They are,

(i) Speculation
(ii) Collaboration
(iii) Learning

(i) **Speculation:** In this phase, the project is started and adaptive cycle planning is carried out which uses certain information. The information includes project initiation information such as customer's mission statement, project limitations such as delivery dates or description of user and general needs necessary for specifying set of release cycles for the project. Even though the cycle planning is absolute and designed based upon future requirements it will always undergo some changes. After the elicitation of information from first cycle, the entire plan is reviewed and altered so that it is adjusted and suitable for the environment in which ASD team is operating.

(ii) **Collaboration:** Many people collaborates inorder to enhance the talent and generate creative output. Among all the methods in agile, the collaboration phase is recurring and difficult to perform. It emphasizes upon interaction, team work and individualism because, a single person's assumptions are necessary for making collaborative thinking. Aside this, the trust also contributes when people work together. It includes,

- ❖ Identifying fault without enmity.
- ❖ Support without ill-feeling.
- ❖ Working very hard.
- ❖ Skilled to support the work in hard.
- ❖ Discussing the problems so that they produces effective action.

(iii) **Learning:** During the development of the components in adaptive cycle, the ASD team focuses upon 'learning'. Here, software developers over estimates their comprehension ability corresponding to the technology, process and project. Thus, learning the team may improvise their level of real understanding.

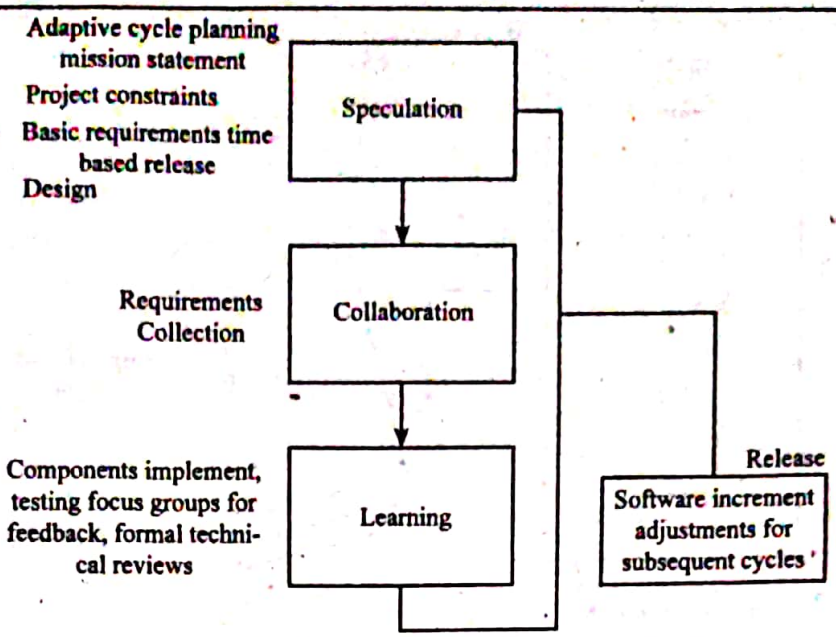


Figure: Adaptive Software Development

For remaining answer refer Unit-I, Q49, Q50, Q51, Q53 and Q54.

Q60. Explain in detail about Scrum Process Model.

Answer :

Scrum

The scrum process model is proposed by Jeff Sutherland in early 90s. Its name got its origin from an activity occurred at the time of rugby match. Later enhancements to the model were conducted by Schwaber and Beedle.

The principles of the scrum model stay continuous with the agile principles. These principles suggest guidelines to the development activities existing in the process which undergo the framework activities such as analysis, design, evolution and delivery. Every individual framework activity conducts work tasks with respect to process pattern.called sprint. The work task with in a sprint is adjusted corresponding to the current problem and then later, scrum team specifies it and alteres it in real time.

A set of software process patterns is often used by scrum team. These patterns are preferred for projects which have strict deadlines, ephemerlal requirements and business criticalities. But these processes illustrate set of development actions. They are,

1. **Backlog:** It signifies priortized list of project requirements or features serving as a business value to the customer. The agile team can include items into the list at any given time which is managed by the project manager who also computes and updates the priorities.

2. **Sprints:** It specifies work units for elicitating requirements specified in backlogs to be adjusted into predefined time box for a period of 30 days. The sprint does not support modifications. Therefore allowing the members of the team to work for a short-period of time in stable work environment.

3. **Scrum Meetings:** It is typically concerned with the discussion meeting among scrum team which does not last more than 15 minutes of time and it can be carried out regularly. The queries raised here are,

(i) How much the team members have progressed since the commencement of last meeting?

(ii) What kind of difficulties and impediments are faced by team members?

(iii) What plans are designed inorder to get fulfilled by next meeting?

The team meetings are controlled and managed by a team lead who is referred to as scrum master. He computes the responses obtained from each person in the meeting. With such meeting, the team can determine potential risks, issues etc. Hence, these meetings are necessary for knowledge socialization and encourage self-organizing team framework.

4. **Demos:** It emphasizes upon the delivery of the software increment to the customer. Upon receiving, the customer validates or demonstrates and evaluates the software. Another important concern is that the demo may lack complete planned functionality but certainly offer functions which can be finished within specified time period.

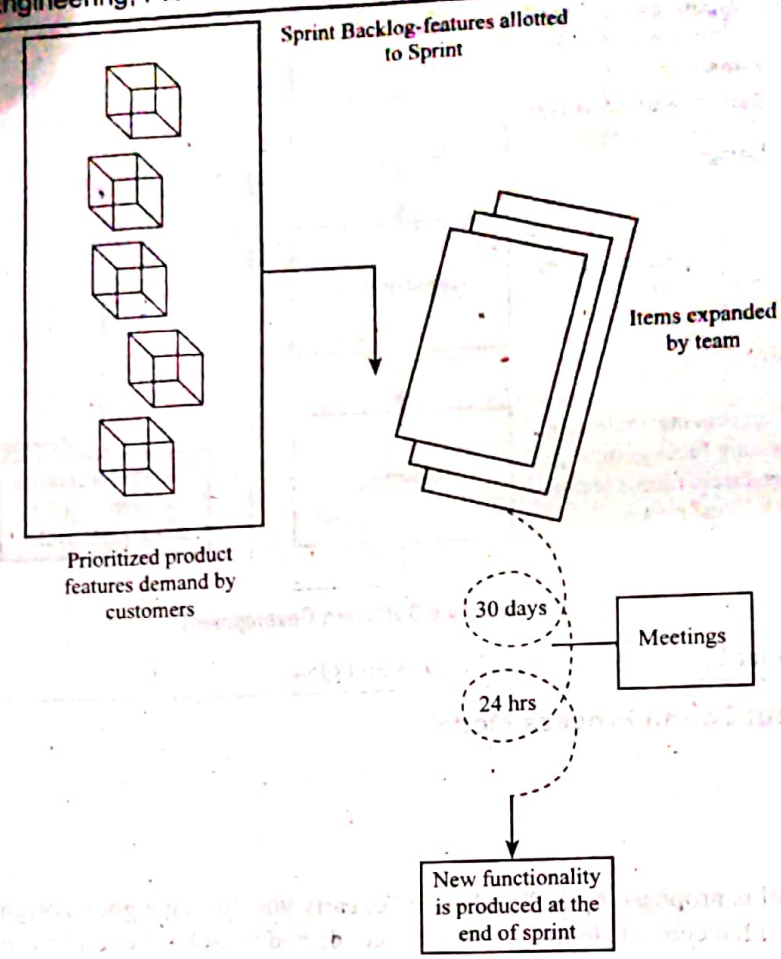


Figure: Scrum Process Flow

Q61. Describe briefly Dynamic Systems Development Method (DSDM).

Answer :

Dynamic Systems Development Method(DSDM)

The Dynamic Systems Development Method(DSDM) is a technique of agile software development. It facilitates the structure for developing and maintaining the system under strict deadlines through the use of incremental prototyping.

The DSDM is typically an iterative software process which requires sufficient work in every increment inorder to provide activity to the up coming increment. Another concern is about the DSDM consortium i.e., www.dsdm.org is a famous group of member companies which together plays the role of keeper in the method. The consortium specifies DSDM life cycle which has three different iterative cycles and other two life cycle activities.

1. Feasibility study
2. Business study
3. Functional model iteration.

1. **Feasibility Study:** The feasibility study suggests primary business requirements and limitations corresponding to the application to be developed. Later it computes to check the feasibility of the application to carryout DSDM process.
2. **Business Study:** The business study suggests functional and information requirements which drives the application the opportunity to provide business value. Apart from this, it also specifies primary architecture of the application and ensures maintainability requirements with respect to application.
3. **Functional Model Iteration:** The functional model iteration ensures set of incremental prototypes suggesting functions for the customers. The idea behind this is to integrate new requirements through user feedback.

The other two life cycle activities are:

(i) Design and build iteration

(ii) Implementation.

(i) **Design and Build Iteration:** It confirms the revisiting of the prototypes created at the time of functional model iteration. It ascertains that every single prototype is engineered to facilitate operational business value for end users. In certain scenarios, the design and build iteration and functional model can occur parallely.

(ii) **Implementation:** It suggest locations for software increment into operational environment.

It is necessary to note that:

(a) The increment is not 100%.

(b) Once the increment is done, the modifications can be made.

Q62. Write short notes on,

(i) Crystal

(ii) Feature Driven Development (FDD).

Answer :

(i) Crystal

The crystal family of agile methods was proposed by Alistair Cockburn and Jim Highsmith. This method focuses upon developing a software with limited resources by implementing "planned and regulated activities (maneuverability). The basic goal of this is to deliver a working software and then using it in later developments.

The maneuverability is achieved through specifying group of methodologies having common core elements. But, the roles, pattern for the process, work products and practices does not match with each other.

The motive behind this model is to permit agile teams to choose the member for crystal family suitable for project and environment.

(ii) Feature Driven Development(FDD)

The Feature Driven Development(FDD) was proposed by Peter Coad and his team members. It was a practical process model for object oriented software engineering. Later, extensions were made by Stephen Palmer and John Felsing to improvise the Coad's work. In a way similar to previous agile process, the FDD operates upon the idea i.e.,

1. It focuses upon the collaboration of the people working in FDD team.
2. It implements feature-based decomposition and then combines software increments to solve the problems and complexities in projects.
3. It carries out technical discussion through verbal graphical and texts.

Apart from this, FDD undertake activities to ensure quality of software. The activities could be incremental development strategy, designing and code inspections, applications with respect to software quality assurance audits, metric collections and pattern usage for analysis, designing and construction.

Moreover, the FDD model encompasses a feature set which integrates the related features into business related categories and is specified as:

<action><-ing>a(n)<object>

The template is given as:

<action> the <result><by|for|of|to>a(n)<object>

Where as,

object = place, person, thing, roles, moments of time or intervals of time etc.

Examples

1. Showing the technical specification of the product.
2. Placing the product into the shopping cart.

In general, the feature is a client-valued function which requires a completion time of nearly 15-days or less. Additionally, the FDD approach characterizes five collaborating frameworks. They are,

1. **Overall Development of a Model:** It emphasizes in complete development of the model by giving more shape to it rather than content.
2. **Development of List of Features:** It encompasses list of features integrated into sets and subject areas.
3. **Developing a Plan with respect to Feature:** It encompasses development plan class owners and feature set owners.
4. **Developing Design with respect to Feature:** In encompasses a design package.
5. **Developing with respect to Feature:** It encompasses the completion of client-value function.

The benefits provided by the feature are:

1. The features are small blocks with portable functionality, so the users can easily specify them, easily associate them with one another and easily review them for errors.
2. The features can be streamlined into hierarchical business related sets.
3. The feature hierarchy supports project planning, scheduling and tracking.

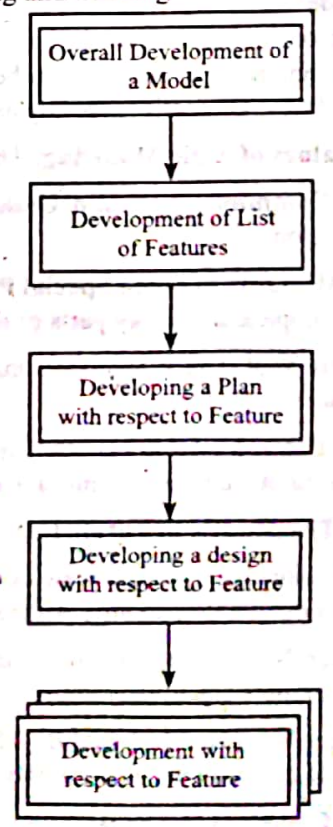


Figure: Feature Driven Development

Q63. Write about Lean Software Development(LSD).

Answer :

Lean Software Development(LSD): The lean software development model operates upon lean manufacturing principles of software engineering. These principles are as follows:

1. Deletion of waste
2. Developing quality-in
3. Creating knowledge
4. Delays in commitment
5. Keeping the regard of people and honouring them
6. Rapid Delivery
7. Complete Optimization

The software process operates upon these principles.

For instance, the principle-deletion of waste corresponding to agile software project is translated as:

1. Stop the inclusion of additional features or functions.
2. Computing the effect of cost and schedule of new requirement.
3. Deleting the additional steps occurring in the process.
4. Setting up the mechanisms for searching the information.
5. Conforming that the testing is preformed efficiently and errors are identified.
6. Minimizing the time for requesting and receiving the decision which impacts the software or the process used for developing it.
7. Organizing the methods used for forwarding the information from one stakeholder to another.

Q64. Explain in detail about Agile Modeling(AM).

Answer :

Agile Modeling

Agile Modeling aims at understanding and communicating with the problem and solution space rather than documenting. The simplest approach to agile modeling is to use whiteboards or white plastic static cling sheets that covers a huge wall area. Markers, digital cameras and printers can be used to capture UML sketch upon these whiteboard or sheets. This enables the designer to identify the alternatives rapidly and also the path to a good object-oriented design.

Practices and Values of Agile Modeling: The various practices and values of agile modeling are as follows.

1. **Support Communication and Understanding:** It aims at supporting communication and understanding rather than documentation.
2. **Apply UML Only to Some Special Parts:** Do not apply UML to entire software design. Apply only to small sections of unusual, complex and tricky parts of design.
3. **Use Simple Tool:** Use simple tools that are capable of supporting rapid input and change and the tools that support large visual spaces.
4. **Model in Pairs:** Model in pairs so that the purpose of agile modeling is served. Modeling in pairs enables the developers to identify, understand and communicate well. Allow all the developers to participate by rotating the pen sketching.
5. **Develop The Models in Parallel:** Develop the models in parallel by switching them back and forth.

   Example: Sketch a dynamic-view of UML interaction diagram on one whiteboard and the complementary static-view of UML class diagram on another whiteboard.
6. **Use Simple Notations:** Use simple and frequently used UML notations. It is not mandatory to draw exact UML details as the developers are already aware of each other.
7. **OO Design Must Be Modelled By The Developers Only:** Developers must model OO design by themselves rather than creating the diagrams for the programmers for implementation.
8. **Test The Code:** It is important to test the code so as to give the true and exact design. All the other diagrams before testing are incomplete hints for explorations.

Q65. Briefly discuss Agile Unified Process(AUP).

Answer :

Agile Unified Process(AUP)

The Agile Unified Process(AUP) operates upon the concept of "Serial in Large" and "Iteration in Small" for developing computer based systems. The model uses unified process activities such as inception, elaboration, construction and transition. In simple terms, the model carries out a serial overlay where in linear sequence of software engineering activities are carried out. This typically allows the agile team to imagine complete process development for software project. But with the progress in every activity, the iteration takes place to procure agility and then handing over the error-free and meaningful software increments to the end-users.

The AUP performs the following activities:

1. **Modeling:** It encompasses the UML representation of business and problem domains. The agility can be confirmed if models are sufficient enough to proceed.
2. **Implementation:** It encompasses the interpretation of models into source code.
3. **Testing:** It encompasses sequential testing to identify errors and ascertains that source code has all the requirements.
4. **Deployment:** It encompasses the software increment delivery and feedback from end users.
5. **Configuration and Project Management:** The configuration management resolves issues like management of changes, risk management work products management etc. On the other hand, project management keeps track and monitor the progress of the team and co-ordinates activities of the team members.
6. **Environment Management:** It encompasses the process infrastructure such as standards, tools and various other support technology.

UNIT 2

SOFTWARE ENGINEERING PRINCIPLES, SYSTEM ENGINEERING AND REQUIREMENTS ENGINEERING

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. List out any five principles of software engineering. Model Paper-I, Q3

Answer :

1. **Achieving Quality**
   The quality of projects cannot be defined as per the project and its outset. In the early stages of life cycle, the modern process framework tries to know the incompatibilities between cost schedule, features and quality. Thus, quality cannot be achieved till these factors are clearly understood.
2. **High-quality Software**
   This principle is redundant with others in most of the situations.
3. **Early Product Generation**
   It might involve rework multiple times because product is provided to the customer as early as possible.
4. **Capturing Problems Before Requirements**
   The conventional requirements specification process have some issues related to the parameters of the problems. As the solution evolves, the tangibility of the problem also arises. The simultaneous evolution of problem and solution continues till it is sure enough that the problems are clearly understood and can be transformed into production.
5. **Evaluation of Design Alternatives**
   In the water fall model the requirements phase is followed by the architecture phase, and the requirements specification includes the architecture as well. This feature of the waterfall model is quoted in principle #5. However, the requirement specification and architecture of an artifact is performed concurrently. In any modern process the architecture and requirement artifacts notations are explicitly decoupled.

Q2. List out any five modeling principles.

Answer :

1. Software team should focus on the development of a good software but not on the development of models.
2. Create only those models that help in building software easily and quickly.
3. Create an easy and understandable model that describe the software which helps in developing softwares that are simple.
4. Use the model notations or rules according to the type of application or software.
5. Created model should have an explicit purpose in the process.

Q3. What is deployment? Model Paper-II, Q3

Answer :

The deployment activity consists of three actions such as delivery, support and feedback. The deployment can occur multiple times while software moves towards completion. This is because the modern software models are evolutionary in nature. The delivery cycle provides customers with operational software environment that provides usable functions and features. The delivery of software increment ensures an important milestone for the software project. Several principles must be followed while team prepares for software delivery.

Q4. What is system modeling?

Answer :

System modeling is one of significant element of system engineering process. The engineer will develop the models with below features irrespective of their focus on particular view.

- → It can define the processes to fulfill the requirements of view under consideration.
- → It can represent the behaviour of processes and assumptions on which behaviour depends on.
- → It can define exogenous as well as endogenous input to model.
- → It can depict the connections which allow the engineers to understanding the view.

Q5. What is the purpose of business process engineering.

Answer : Model Paper-III, Q3

The major purpose of business process engineering is to define the architectures which allow the business to use information effectively. The specification of appropriate computing architecture is required along with the development of software architecture which increases the organizations unique configuration of computing resources. The business process engineering is method of creating the overall plan to implement the computing architecture.

The real-time and embedded systems belong to reactive systems category most of the systems in this category will control the machines and processes which need to operate with degree of reliability. An economic or human loss can be faced when system fails. Because of this the tools of system modeling and simulation are used for eliminating the surprises while developing the computer based system. Such type of tools are applied while system engineering process and when hardware, software, databases and people's roles are specified.

Qo. List all the different tasks that performed during requirement engineering.

Answer :

The different tasks that are performed during requirement engineering are,

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation and
7. Management.

Q7. Define scenario.

Answer :

Scenario describe the way a 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 scenario 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 depicts different sort of information at different abstraction level.

Q8. Write a short note on data objects.

Answer : Model Paper-I, Q4

Data objects can be described as representation of composite information. It essence it contains number of different properties or attributes. Thus, width could not be considered as a valid data object while dimensions can be counted as an object.

(a) A thing such as display, report.

(b) An occurrence such as telephone call.

(c) An external entity such as unspecified object or event that produces or consumer information.

(d) An event such as alarm.

(e) A role such as sales person.

(f) An organizational unit such as accounting department, resources department.

(g) A place such as data warehouse.

(h) A structure such as file.

Q9. Discuss briefly on the approaches of requirements modeling.

Answer : Model Paper-III, Q4

Requirement modeling comprises a view called structured analysis. Let us assume data and processes which translates data as separate entities. Data objects are designed in such a way that it specifies the related attributes and relationship. Processes which transforms data objects are designed in a way that shows transformation of data into data objects flow in a system. Object oriented analysis is the second approach in analysis modelling which emphasizes on classes and the way they interface with each other. It also shows the impact on customers requirements. Requirement model integrates the feature of structured analysis and object oriented analysis . Software terms generally selects only one approach for functioning and leaves the other. The major query over here is which models efficiently works with the available software requirements, instead of which is the best model.

Requirements models shows the problem that can encounter in the software from different point of views. Scenario based models shows the way of user interaction with system and sequence of activities that come across with use of software. Class based model depicts the objects that will change the system, operations which are imposed to change the system by the objects, collaborations that to occurs in the specified classes and relationship that exists between the objects. Behavioral models shows the change in the state of system or classes. Flow models depicts the transformation in data objects as they flow from several system functions.

Q10. What are the main uses of the requirements models?

Answer : Model Paper-II, Q4

The main uses of requirements model are,

1. It can be used to represent the system in various forms with which customer needs can be described.
2. It can be used to define certain requirements to be validated after the development of software.
3. It acts as a basis for software design.
4. With requirements model, a detailed description of system functionality with respect to application's architecture, components and user interface can be obtained.

PART-B

ESSAY QUESTIONS WITH ANSWERS

2.1 SOFTWARE ENGINEERING PRINCIPLES

2.1.1 SE Principles, Communication Principles, Planning Principles

Q11. Briefly explain the principles of software engineering.

Model Paper-II, Q12(a)

Answer :

The conventional software engineering principles are

1. **Achieving Quality**

   The quality of projects cannot be defined as per the project and its outset. In the early stages of life cycle, the modern process framework tries to know the incompatibilities between cost schedule, features and quality. Thus, quality cannot be achieved till these factors are clearly understood.

2. **High-quality Software**

   This principle is redundant with others in most of the situations.

3. **Early Product Generation**

   It might involve rework multiple times because product is provided to the customer as early as possible.

4. **Capturing Problems Before Requirements**

   The conventional requirements specification process have some issues related to the parameters of the problems. As the solution evolves, the tangibility of the problem also arises. The simultaneous evolution of problem and solution continues till it is sure enough that the problems are clearly understood and can be transformed into production.

5. **Evaluation of Design Alternatives**

   In the water fall model the requirements phase is followed by the architecture phase, and the requirements specification includes the architecture as well. This feature of the waterfall model is quoted in principle #5. However, the requirement specification and architecture of an artifact is performed concurrently. In any modern process the architecture and requirement artifacts notations are explicitly decoupled.

6. **Using Appropriate Process Model**

   A single process cannot be universally accepted. However, a class of processes can be accepted and is usually referred by the term process framework.

7. **Employing Techniques before Tools**

   This principle uses two points in a wrong way.

   (i) Automation is one of the best methods of standardizing, promoting and delivering goods.

   (ii) A disciplined software expert with no tools may produce less than a disciplined software engineer provided with good tools.

8. **Focus on Generating Appropriate Product Instead of Speeding Up**

   Most of the software experts misstate this principle as "Early performance problems in a software system are a sure sign of downstream risk."

9. **Code Inspection**

   Inspections are always overhyped by most of the organizations. Moreover, inspections are not that much suitable, as most of the global design trade-offs and architectural issues are rarely uncovered by the undirected inspections.

10. **Good Management over Good Technology**

    This principle is the most motivating principle for most software professionals. However, they disagree with the use of the word "meager". The phrases "team with meager quality" and "good management" are mutually exclusive, as per these professionals.

11. **People are Key to Success**

    Most of the software professionals comment that the importance given to this principle is less.

12. **Follow with Care**

Most of the modern technologies are not always supported if schedules, costs and features are traded off.

13. **Understanding Customer Priorities**

The priorities of customers and stakeholders must be considered equally. If customer is given more importance, then this misconception becomes the most exaggerating one.

14. **The More they See Need**

This principle focuses on hiding the functionality of the product from the customers. However, it should be stated as "The more users see, the better they understand". The reason being, even stakeholders understand. The in and out of the product and difficulties in meeting the expectations with constrained developers and limited resources

15. **Plan to Throw One Away**

Avoid this principle and try to avoid evolving a product from the scratch. Rather, use an immature prototype to build a mature baseline. Most of the successful software projects, use this advice and has proven to be the leading projects. There are many existing components like GUI, network, operating system, middleware and DBMS that can be used to create new products.

16. **Design for Change**

This principle says that it is necessary to forecast and build a framework with the capability of getting updated with the changes that are not yet known. Indeed, it is a difficult task, but giving an attempt to it is a good idea. The reason being risk management and a recurring theme of the successful software projects usually involve such an exercise.

17. **Design without Documentation is not Design**

The design and documentation of software are separate processes i.e., they are separately written. However, such a strategy becomes counter productive specially with high-level programming languages and visual modeling. Thus, by writing the documents precisely and properly, a better support can be provided to the next phases. Despite of the advantages of documentation, the members of the engineering team use source code, test baselines, and design notations, as their primary artifacts. This principle is restated by walker Royce as "software artifacts should be mostly self-documenting."

18. **Use Tools but be Realistic**

According to walker Royce, this principle does not deserve to be in the list of top 30 principles. The reasons being, complex coding techniques should not be used unless there exist equally valid reasons for using them. Projects that are nontrivial usually have such compelling reasons.

19. **Encapsulation**

This principle should be given more importance as the mainstream practice in programming notations, object-oriented component-based and modern designs .

20. **Use of Capturing and Cohesion**

There exist no clear definitions of coupling and cohesion. Not only this, coupling and cohesion are difficult to apply and measure. Adaptability and maintainability of any software can be measured by the amount of coupling and cohesion and consequently the amount of scrap and rework.

21. **Use of McCabe Complexity Measure**

These metrics are rarely used in project management that is not automated and are suitable only for the ones that are automated.

22. **Don't Test your Own Software**

The software developed by the software developers can be tested either by themselves or by a separate testing team. Each method has its own pros and con.

23. **Analyzing Causes of Errors**

Error detection is a good idea when there are chances that errors will repeat in the construction phase. But in complex software systems, analysis of errors is tedious job. The reason being in the early stages of the project itself, the amount of analysis and design is huge. Thus, performing such activities results in lower ROI. It is recommended to make errors in the engineering stage and analyze their causes in the production stage.

24. **Different Languages in Different Phases**

Instead of employing a single language for all phases use different languages in different phases of life-cycle.

25. **Reduce the Intellectual Distance**

The structure of a software made closely related to the real-world structure thereby reducing the intellectual distance.

26. **Take the Responsibility**

A software engineer must take the responsibility of using the best methods for producing the design, ensuring the correct production of software.

27. **Avoid using the Tricks**

Avoid using the tricks in program code as it obscure the required functionality.

28. **Increase in the Software's Entropy**

The complexity of the software increases when the system is subjected to continuous modifications.

29. **People and Time are not Substituents**

It is better to use a single metric i.e., person-months throughout the life cycle.

30. High Expectations

The employees must excel their performance in order to fulfill the expectations of the software manager.

Q12. Briefly discuss the communication principles.

Answer :

The customer requirements need to be gathered through one of the communication activity before they are analyzed, modeled or specified. The communication principles are illustrated as follows,

Principle 1 : Listen

Always focus on the words of speaker and then respond to them. If something is not dear then ask for clarification. Do not interrupt frequently and do not become disputable or debatable in your words or actions while communicating with someone.

Principle 2: Prepare Before Communication

Try to understand the problem before meeting research to understand the business domain jargon. And prepare an agenda before conducting a meeting.

Principle 3: Someone must Facilitate the Activity

A leader must be maintained for every communication meeting to manage the conversation to move in productive direction, to mediate the conflicts and to assure that other principles are followed.

Principle 4: Face to Face Communication is best

It works better when relevant information is represented.

Principle 5: Take Notes and Document Decisions

There is a chance for things to fall into the cracks. So one of the participant must record or write the decisions and important points in the meeting.

Principle 6: Strive for Collaboration

Collaboration and consensus might occur while collective knowledge of team members is used for describing the product or system functions or features. Every collaboration strives to develop trust among team members and creates common goal for team.

Principle 7: Stay Focused, Modularize the Discussion

When more number of people are involved in any communication then there is a chance for the discussion to deviate to some other topic. So, the facilitator must keep the conversation modular.

Principle 8: If something is Unclear, Draw a Picture

A sketch or drawing can better explain clearly when something is not understandable

Principle 9

(a) Once a year agree to something, move on;

(b) If you cannot agree to something; move one;

(c) If a feature or function is unclear and cannot be clarified at the movement, move on.

communication consumes time like other activities. The participants must recognize that most of the topics need discussion. So it is better to move on sometimes rather than iterating endlessly to achieve communication agility.

Principle 10: Negotiation is Not Contest or Game it Works Best When Both Parties Win

The software engineer and customer need to negotiate the functions and features, priorities and delivery dates most of the time. The parties will have common goal if the team collaborates in a well manner. The negotiation thus demands a compromise from all of the parties.

Q13. Write about the planning principles.

Answer :

The planning activity contains a set of management and technical principle which allow the software team for defining the road map along the journey to strategic gaol and tactical objectives.

The planning principles are illustrated as follows,

Principle 1: Understand the Scope of the Project

It is of no use to follow the road map when destination is not used. The scope provides destination to the software.

Principle 2: Involve Customer in Planning the Activity

The customer will define the priorities and even establish the project constraints. The software engineers must negotiate the order of delivery, timelines and other issues of project for accomodating the realities.

Principle 3: Recognize that Planning is Iterative

The project plan is always not stable. Things change more often and the plan must be adjusted to accomodate these changes. In addition to this, the iterative, and the incremental process models will dictate replanning depending upon the feedback of users.

Principle 4: Estimate based On What you Know

The purpose of estimation is to indicate effort, task duration and cost depending on the teams present understanding of work. The estimates become unreliable when information is vague or unreliable.

Principle 5: Consider Risk as you Define the Plan

The contingency planning becomes necessary when team defined risks with high impact and probability. In addition to this the project plan must be adjusted to accomodate the likelihood of risks.

Principle 6: Be Realistic

People mostly will not work 100% of the day. There is a chance for noise to enter into the human communication. Omissions and ambiguity are facts of life and changes occur more of then. The best software engineers also commit mistakes. Such realities must be considered while porject plan is established.

Principle 7: Adjust granularity as a you Define Plan

Granularity can be defined as level of detail which is introduced while project plan is developed. The "Fine granularity" plan offers a work task detail planned through short time increments. A "Coarse granularity" plan offers broader work tasks which are planned through longer time periods. The granularity moves from fine to coarse when project timeline delays. The project can be planned in significant detail in the following weeks or months.

Principle 8: Define How you Intend to Ensure Quality

The plan must recognize in what way the software team assures quality. They need to be scheduled if formal technical reviews are conducted. The pair programming must be defined in plan if they are used while construction.

Principle 9: Describe How you Intend to Accomodate Change

The best plans are also sometimes avoided due to uncontrolled change. The software team is responsible to recognize the changes.

Principle 10: Track the Plan Frequently and make Adjustments as Required

The software projects get delayed some times. So, the progress must tracked constantly for problem areas and situations in which the scheduled work does not conform to actual work. The plan must be adjusted when a delay is encountered.

2.1.2 Modeling Principles, Construction Principles, Deployment

Q14. Discuss the modeling principles.

Answer :

The principles of modeling are as follows,

1. Software team should focus on the development of a good software but not on the development of models.
2. Create only those models that help in building software easily and quickly.
3. Create an easy and understandable model that describe the software which helps in developing softwares that are simple.
4. Use the model notations or rules according to the type of application or software.
5. Created model should have an explicit purpose in the process.
6. Focus on creating useful models rather than focusing on perfect model.
7. Check whether the model is communicating properly or not rather than concentrating on the representations.
8. Spend extra time on examining the models that seems to fail, even if the situation requires the new one.
9. Obtain the feedback from the reviewers team at the earliest.
10. Develop the models in such a way that, they should manage the changes if required

Q15. Explain the principles of construction.

Answer :

The construction process contains set of coding and testing tasks that encorporate for a successful software ready to be delivered to the customer.

A set of principles and concepts that are applicable to coding and testing are as follows,

1. **Coding Principles**

The coding principles guide the coding task and they are aligned with respect to programming style, languages and methods. The fundamental principles of coding are illustrated as follows,

Preparation Principles

1. Understand the problem that is to be solved.
2. Understand the basic design principles and concepts.
3. Select the programming language which meets the needs of software and the environment.
4. Select the programming environment which offers tools which help in making the work easier.
5. Create unit tests which will be applied when coding of a component completes.

Coding Principles

1. Drive the algorithms by following structured programming practice.
2. Select the data structures which meet the design needs.
3. Understand software architecture and then create interfaces which are consistent.
4. Maintain a simple conditional logic.
5. Make nested loops to be easily testable.
6. Select variable names that are meaningful and then follow the other local coding standards.
7. Make code to be self-documenting.
8. Make a clear and understandable visual layout.

Validation Principles

1. Conduct a code walk through whenever required.
2. Perform unit tests and then correct the uncovered errors.
3. Refactor the code.

2. Testing Principles

Testing is nothing but a process of executing program with the intent to find an error. A good test case is said to be the one with high probability to find as-yet-undiscovered-error. And a successful test uncovers the as-yet-undiscovered error. The major goal is to design the tests systematically uncover various classes of errors as well as to do so in minimum amount of time and effort.

Principle 1: All Tests must be Traceable to Customer Requirements

The primary goal of testing is to uncover the errors. It assumes that the defects which cause the program fail in meeting its requirements are the severe defects.

Principle 2: Tests must be Planned Long Before Testing Begins

Test planning must begin soon after the completion of analysis model. All of the tests must be planned and designed before coding begins.

Principle 3: The Pareto Principle Applies to Software Testing

The pareto principle implies that so percent of all the errors which are found while testing are traceable to 20 percent of the components. They must be isolated and tested thoroughly

Principle 4: Testing must Begin "in the Small" and Progress toward Testing "in the Large"

The first tests which are planned and executed will focus on individual components. While there is a progress in testing the focus now moves on to find errors in clusters of components and the entire system.

Principle 5: Exhaustive Testing is not Possible

Many of the path permutations is large for moderately sized program. So, if would be impossible to execute all the combination of paths while testing. But it is possible to cover program logic and to assure that the conditions in the component level design are exercised.

Q16. Write in short about deployment

Answer :

The deployment activity consists of three actions such as delivery, support and feedback. The deployment can occur multiple times while software moves towards completion. This is because the modern software models are evolutionary in nature. The delivery cycle provides customers with operational software environment that provides usable functions and features. The delivery of software increment ensures an important milestone for the software project. Several principles must be followed while team prepares for software delivery.

Principle 1: Customer Expectations for Software must be Managed

The customer will actually expect more than team assures to deliver. So there is a chance for customers to get disappointed. This leads to unproductive feedback which therefore runs the team morale. So, a software engineer must be careful and conscious about customer conflicting messages.

Principle 2: A Complete Delivery Package must be Assembled and Tested

The CD Rom and other media containing the executable file, support data files, support documents etc need to be assembled and beta-tested with actual users. The installation scripts along with operational features must be exercised in possible computing configuration.

Principle 3: A Support Regime must be Established before Software Delivery

The end users will expect the responsiveness and accurate data when a problem or question arises. The customer might get dissatisfied when support is adhoc, nonexistent. In order to provide support it must be planned, material must be prepared and a relevant recording must be established so that software team conducts categorical assessment of requested support types.

Principle 4: Relevant Instructional Materials must be Provided to End-users

The software team will deliver more than the software. So, relevant training aids must be built, trouble shooting guidelines must be provided and the additional features of software must be provided as description.

Principle 5: Buggy Software must be First Fixed and then Delivered

Certain organizations deliver low-quality software and provide a warning to the customers that bugs will be fixed in next version. But this is a mistake. In software business the customers forget about the high quality product released later or but they will always remember the problems passed by low quality software.

2.2 SYSTEM ENGINEERING

2.2.1 Computer – Based Systems

Q17. Write short notes on computer based systems.

Answer :

A computer based system can defined as a set or order of elements that are organized for implementing some predefined goal by processing the data. The objective might be to support certain business functions or to develop the product which can be sold for producing the business revenue. The computer based system uses different elements to accomplish the goal. The software can be defined as computer programs, data structures and related work products to serve the required procedure control or logical method effect. The hardware can be defined as

electronic devices which offer computing capability, the inter connectivity devices which allow the data flow and electromechanical devices which offer external world function. People can be defined as users and operators of hardware as well as software. A database can be defined as a huge collection of organized data which can be accessed through software and persists over time. Documentation can be defined as the descriptive data which represents the use and / or system operation. The procedures can be defined as the step which defines particular use of system element or procedural context of system storage. All these elements are combined in different ways to transform the data.

A complex characteristic of computer based systems is that the elements of one system can also represent a macro element of larger system. It is nothing but computer based system which is part of a more larger computer based system. An example of it would be a robot and data entry device, each of which is a computer based system. The manufacturing cell is defined in next level of hierarchy. It is a computer based system that contains its own elements. It also consists macro elements which are called numerical control machine, robot and data entry device.

The manufacturing cell and its macro elements are composed of system elements along with generic labels, software, hardware, people, database, procedures and documentation. They share a generic element in certain cases. The system engineer is responsible for defining the elements of particular computer based system in context of complete system hierarchy.

2.2.2 The System Engineering Hierarchy

Q18. Illustrate the hierarchy of system engineering.

Answer :

Model Paper-I, Q12(a)

The system engineering consists of a set of top-down and bottom up methods for navigating the hierarchy as illustrated in below figure.

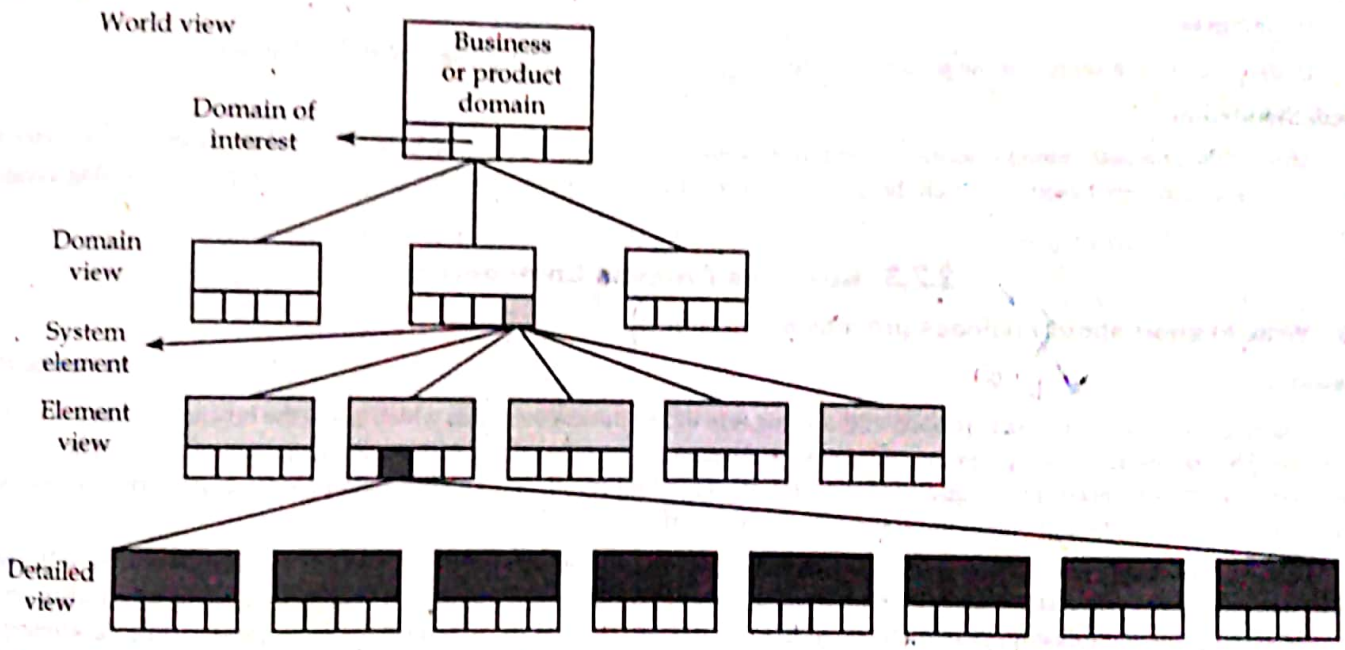


Figure: System Engineering Hierarchy

The process of system engineering initiates with "World View". The complete product or business domain will be thoroughly examined to assure that proper business or technology context is established. The world view focuses on specific domain of interest. The need for targeted system elements will be analyzed in particular domain. In the final stage, the analysis, design and construction of targeted system element is begin.

The world view contains a set of domains (Di), each of which is a system or system of system in its right.

$$wv = \{D_1, D_2, D_3, \ldots D_n\}$$

Every domain contains particular elements that perform some or the other role to establish goals of domain or component

$$D_i = \{E_1, E_2, E_3, \ldots E_n\}$$

In the final stage the element is implemented by specifying technical components (C_i) to achieve required function for element

$$E_j = \{C_1, C_2, C_3, \ldots C_k\}$$

System Modeling

System modeling is one of significant element of system engineering process. The engineer will develop the models with below features irrespective of their focus on particular view.

→ It can define the processes to fulfill the requirements of view under consideration.

→ It can represent the behaviour of processes and assumptions on which behaviour depends on.

→ It can define exogenous as well as endogenous input to model.

→ It can depict the connections which allow the engineers to understanding the view.

A system model can be developed by considering the below defined restraining factors.

1. **Assumptions**

   The assumptions minimize the possible permutations and variations by allowing model to reflect the problem in a very reasonable manner.

2. **Simplifications**

   The simplifications allow model to be built in timely manner.

3. **Limitations**

   Limitations allow to bound to the system.

4. **Constraints**

   Constraints guide the way in which model is built and method adopted while implementing the model.

5. **Preferences**

   Preferences which represent the preferred architecture for complete data, functions and technology.

System Simulation

Most of the computer based systems communicate with real world in reactive fashion. The real world entities are observed by hardware and software based on which the system will froce the control on machine processes and people causing events to occur.

2.2.3 Business Process Engineering

Q19. Write in short about business process engineering.

Answer : Model Paper-III, Q12(a)

The major purpose of business process engineering is to define the architectures which allow the business to use information effectively. The specification of appropriate computing architecture is required along with the development of software architecture which increases the organizations unique configuration of computing resources. The business process engineering is method of creating the overall plan to implement the computing architecture.

The real-time and embedded systems belong to reactive systems category most of the systems in this category will control the machines and processes which need to operate with degree of reliability. An economic or human loss can be faced when system fails. Because of this the tools of system modeling and simulation are used for eliminating the surprises while developing the computer based system. Such type of tools are applied while system engineering process and when hardware, software, databases and people's roles are specified.

There are three architectures which need to be analyzed and designed in the context of business objectives and goals.

1. **Data Architecture**

It provides the framework for information needs of business or business function. The separate building blocks of architecture are data objects which are used by business. The data object consists of a set of attributes which will define certain aspect, quality, characteristic or descriptor of data which is described. One of the set of data objects is defined and their relationships are observed. A relationship represents the connections of objects with each other. The application architecture contains the system elements which will transform the objects in data architecture for business purpose. The application architecture tends to perform such transformation. In broader context, the application architecture will contain the role of people and business procedures which are not automated.

The technology infrastructure offers foundation for data and application architectures. It contains hardware and software whch are used for supporting applications and data. This again consists of computers, operating system, networks, telecommunication links, storage technologies and architecture which is designed for implementing the technologies. These architectures can be modeled by defining the hierarchy of business process engineering activities.

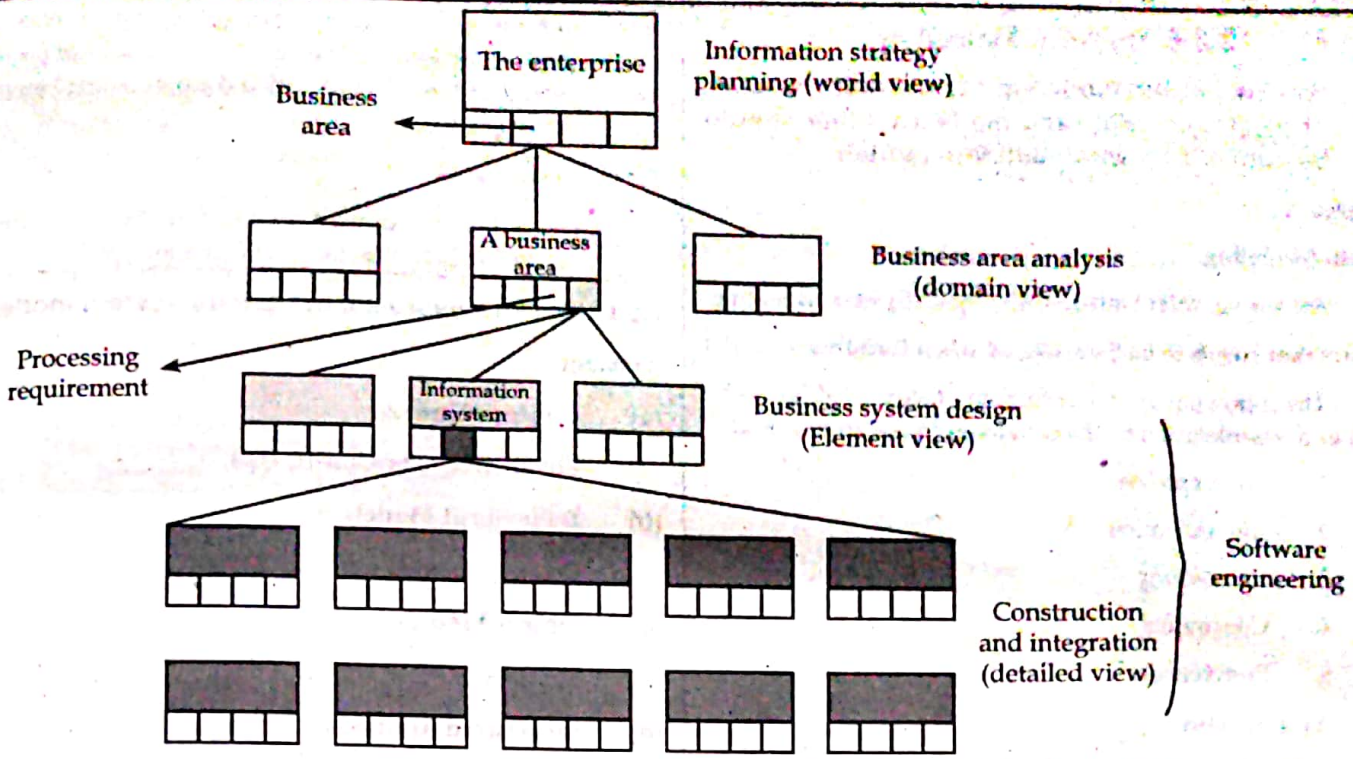


Figure: Business Process Engineering

2.2.4 Product Engineering

Q21. Explain about product engineering.

Answer :

The product engineering process is divided into three processes, which performs a set of activities,

(i) Development process

(ii) Project management process

(iii) Software configuration management process.

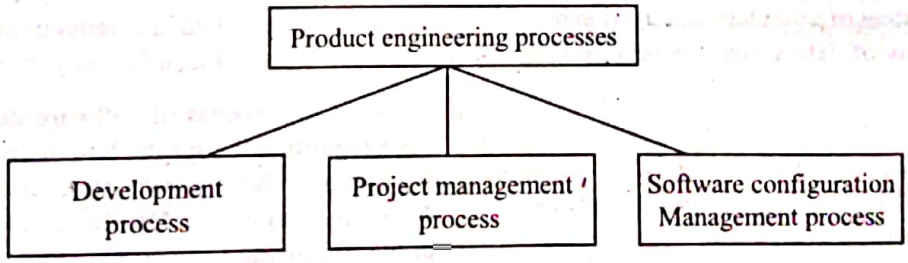


Figure: Product Engineering Processes

(i) **Development Process:** This process involves various activities related to software development and quality assurance. These activities must be performed by a group of persons such as programmers, designers, librarians, writers etc.

(ii) **Project Management Process:** The purpose of project management process is to plan and control the development activities so that a high quality and low cost software can be produced. It entirely depends on the development activities, which must be performed by the project management team.

(iii) **Software Configuration Management Process:** The purpose of this process is to manage the change and rework of components that are not handled by a development process. Despite of changes in software development the change and rework must be controlled in such a way that it does not violate the products integrity. The objectives such as low cost and high quality are successfully met the activities of the configuration control are executed by a community known as Configuration Control Board (CCB).

Thus, these three components are combined to form the product engineering process, which completely focuses on the software projects and products.

2.2.5 System Modeling

Q22. What is system modeling? Explain the process of creating models and the factors that should be considered when building models.

Answer :

System Modeling

For answer refer Unit-II, Q18, Topic: System Modeling.

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.

Q23. Give an overview of various system models.

Answer :

(a) Context Models

For answer refer Unit-II, Q24.

(b) Behavioral Models

For answer refer Unit-II, Q25.

(c) Object Models

For answer refer Unit-II, Q26.

(d) Structured Methods

For answer refer Unit-II, Q27.

(e) Data Models

For answer refer Unit-II, Q28.

Q24. What is meant by context model? With a neat diagram explain the context model of ATM system.

Answer :

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.

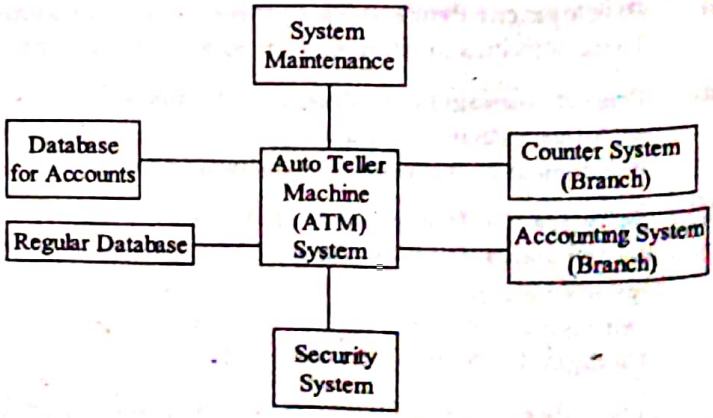


Figure (1): High Level Representation of Bank ATM System

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.

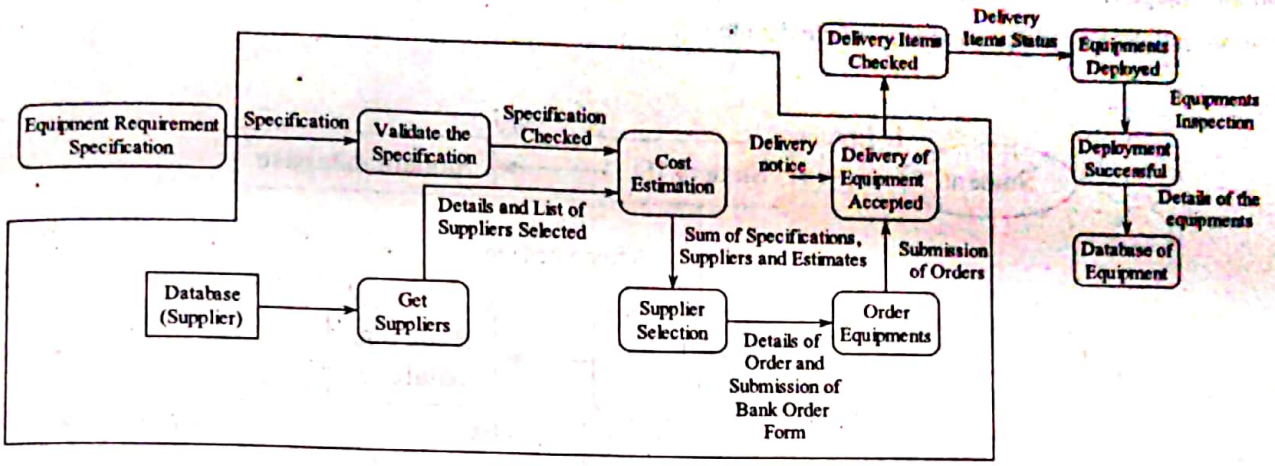


Figure (2): Process Model of Equipment Procurement

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.

Q25. What is a behavioural model? Discuss various types of behavioural models with examples for each.

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,

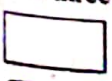
1. Data flow model
2. State machine model.

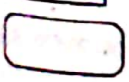
1. Data Flow Model

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

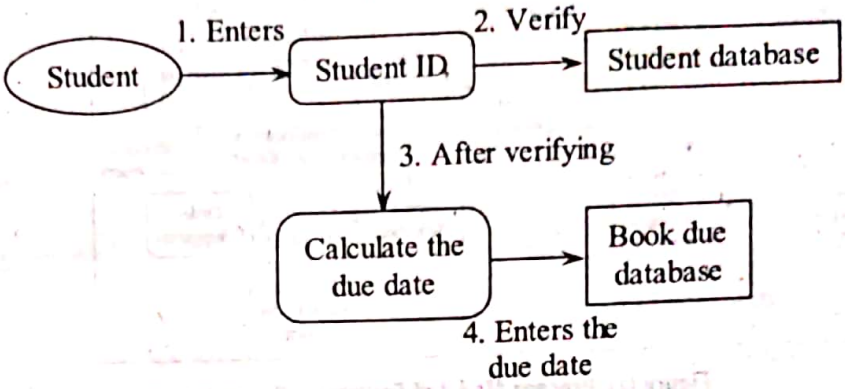


Figure (1): DFD of Library System

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.

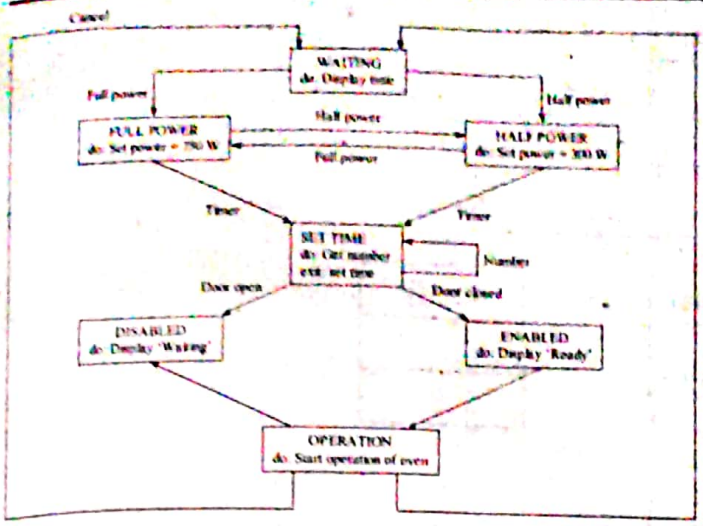


Figure (2): Microwave Oven's State Machine Model

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.

Q26. 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.

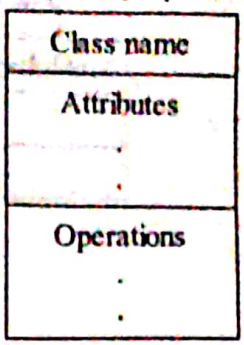
An object class consists of three sections,

(i) Class name

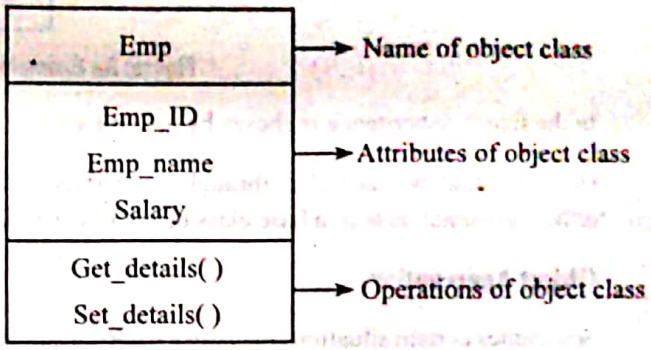
(ii) Attributes of class

(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.

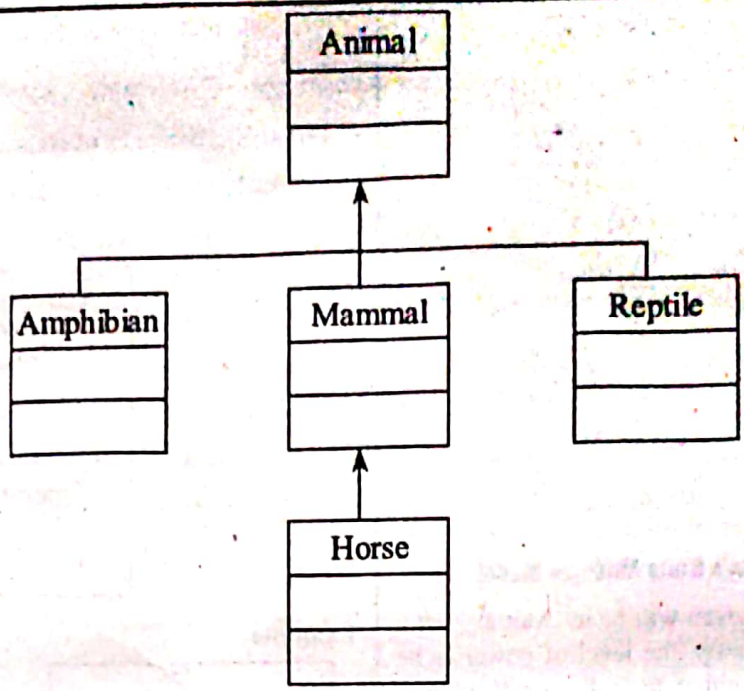


Figure: An Example of Inheritance Hierarchy

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.

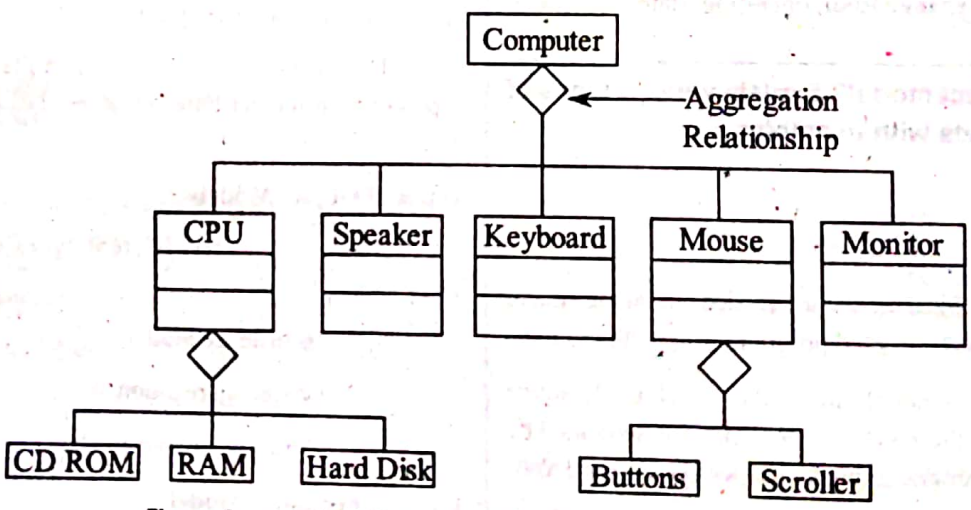


Figure: Aggregation Relationship Existing Among Various Objects

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.

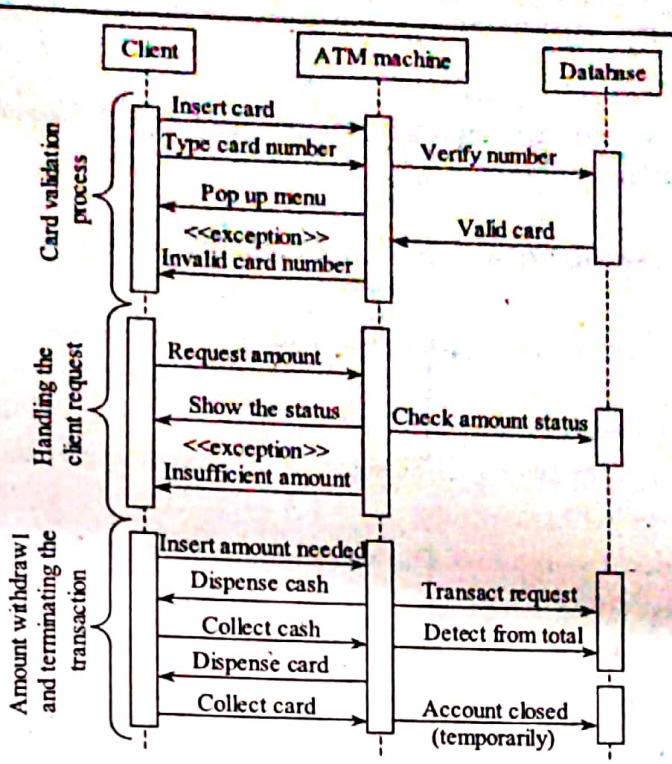


Figure: Sequence Diagram Depicting ATM Withdrawl Transaction

Q27. Discuss about different structured methods used in software development.

Answer :

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.

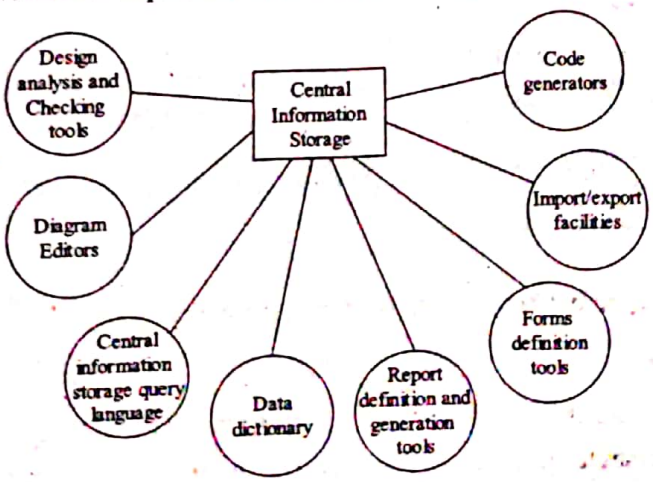


Figure: Components Under the CASE Tools Provided by the Structured Methods

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.

Q28. 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,

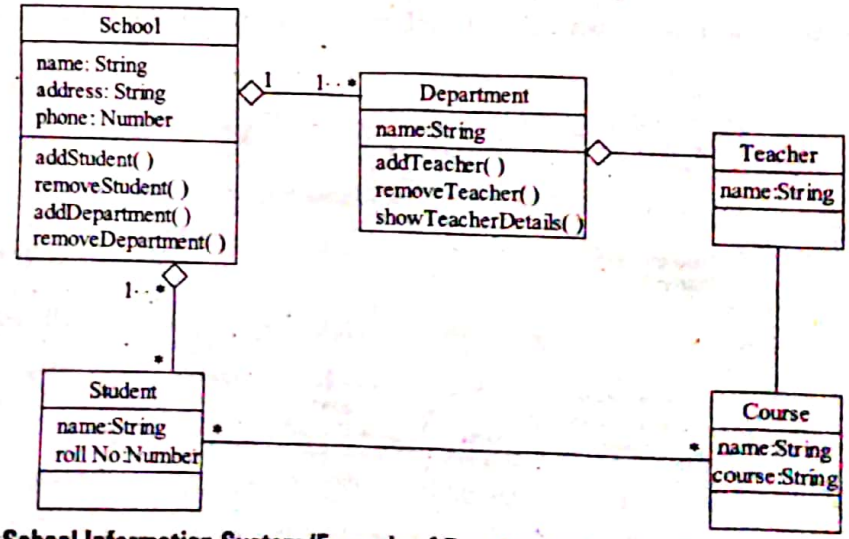


Figure: School Information System (Example of Data Model, Modelling a Database Schema)

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

2.3 REQUIREMENTS ENGINEERING

2.3.1 Abridge to Design and Construction

Q29. Discuss in brig about the bridge to design and construction.

Answer :

The process of designing and developing a software is challenging and creative as well. It is fascinating in the sense that most of the software developers want to jump in before knowing about the requirement. They strongly believe that they can understand more clearly while developing the software. They even argue that the stakeholders will understand the requirements after observing early iterations of software, that requirements engineering is waste of time because it is not same situation always, etc. The arguments seem to be seductive because they are the facts but each of it is flawed and may lead to failed software project. The requirements engineering must be adapted with respect to the needs to process, project and product as well as the people working on software. An abbreviated methods is opted in certain cases. The software team need to adapt the way of approaching RE. The requirements engineering builds a bridge in between the design and construction. According to developers the bridge is assumed to exist in different ways illustrated as follows,

- Starting at feat of project stockholders where business requirements are defined, scenarios of users are defined, functions and features are outlined and project constraints are determined,
- Starting with a wide system definition where system is a part of large system domain.

Without concerning the starting point, the remaining path of bridge contains the following

(i) Software team is allowed to observe the context of software work that is to be performed.

(ii) The requirements addressed by design and construction are represented.

(iii) Priorities are assigned there by guiding the order of tasks to complete the work.

(iv) The required data, functions and behaviors which immensely effect the resultant design are presented.

2.3.2 Requirements Engineering Tasks

Q30. Briefly explain requirements engineering tasks.

Answer : Model Paper-I, Q12(b)

The process of requirements engineering involves the following important tasks.

1. Requirements Elicitation and Analysis

Whenever we are done with the feasibility study stage of the requirements engineering process, the next stage is requirements elicitation and analysis 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.

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

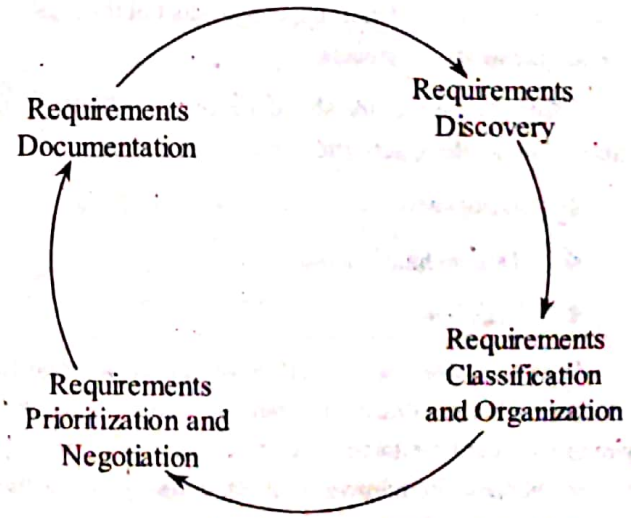


Figure: Various Phases in Requirements Elicitation Analysis Process

Requirements Discovery

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

Requirements Classifications and Organization

As all the requirements are collected and documented, now it has to be organized orderly. For this purpose all the related requirements are clubbed and several groups are framed, often these groups are referred to as clusters.

Requirements Prioritisation and Negotiation

We know that, often human beings are born with different desires and requirements. Hence, whenever the developer consults various stakeholders to gain their requirements, often these requirements conflicts. The developer assigns priorities to these requirements. Therefore, the main purpose of this phase is to provide priorities to these requirements (often referred to as prioritisation) and also to resolve conflicts between them (often referred to as negotiation).

Requirement Documents

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.

2. Requirements Validation

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

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

- ❖ Discovery
- ❖ Documentation and
- ❖ Checking.

However, when we visualize 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.

2.3.3 Initiating Requirements Engineering Process, Eliciting Requirements, Developing Use Cases

Q31. Discuss various steps in requirements engineering. What are the work products of engineering the requirements?

Model Paper-II, Q12(b)

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.

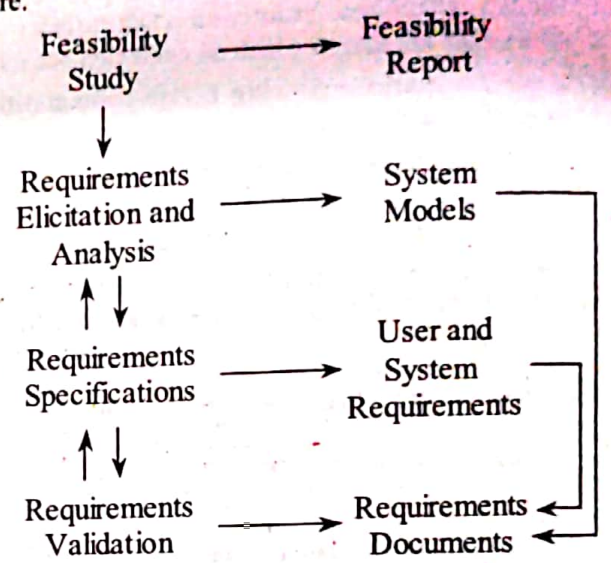


Figure: Various Stages of Requirements Engineering Process

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

1. Feasibility Study

For answer refer Unit-II, Q32.

2. Requirements Elicitation and Analysis

For answer refer Unit-II, Q30, Topic: Requirement elicitation and Analysis.

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

For answer refer Unit-II, Q30, Topic: Requirements Validation.

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.

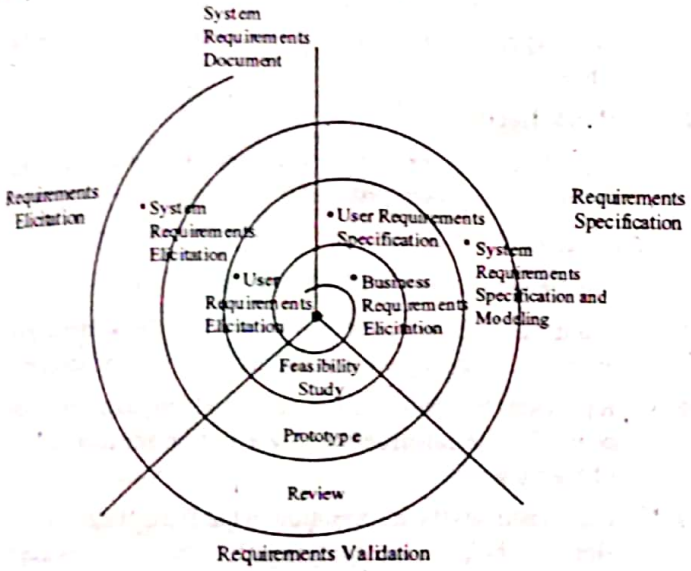


Figure: Spiral Model of Requirements Engineering Processes

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.

Q32. What are the feasibility studies for requirements engineering process?

Answer :

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.

Q33. Elaborate on requirements elicitation and analysis process in detail.

Answer :

Requirements Elicitation and Analysis

For answer refer Unit-II, Q30, Requirements Elicitation and Analysis.

Q34. What is the goal of requirements analysis phase? Give reasons why the requirements analysis phase is difficult one.

Answer :

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.

Q35. What is a view point? Explain its significance. Discuss various types of view points.

Answer :

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 viewpoints. 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.

Q36. Discuss about VORD method in detail.

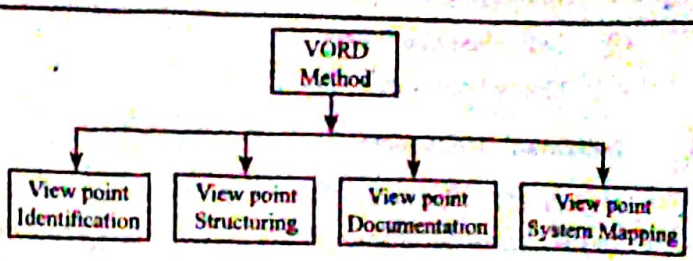
Answer :

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.

Q37. 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.

Q38. 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.

Q39. 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,

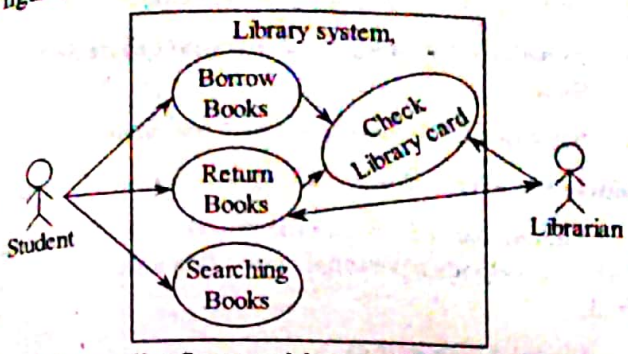


Figure: Use Cases and Actors of Library System

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.

Q40. 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.

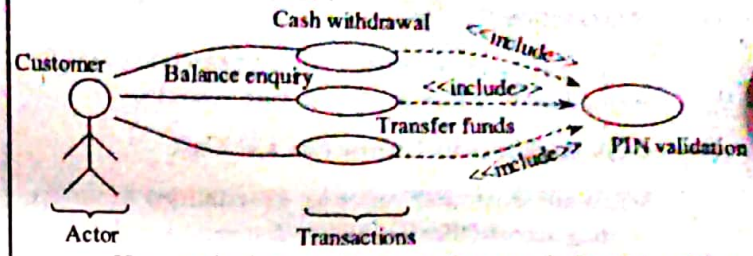
Step6

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

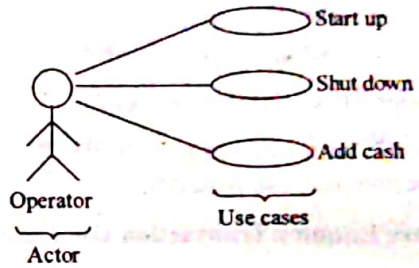
Step7

Customer takes the cash, bank card and the receipt.

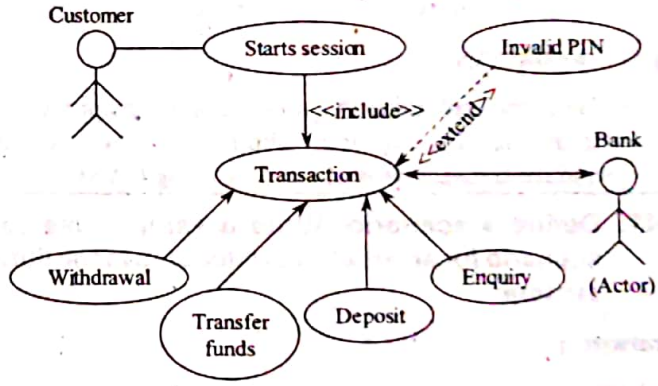
Use Case Model for ATM



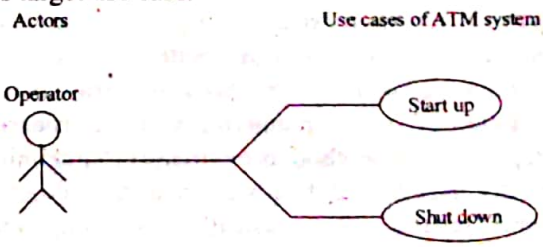
Here <<include>> is dependency relationship which specifies the source use case.



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.

Q41. 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

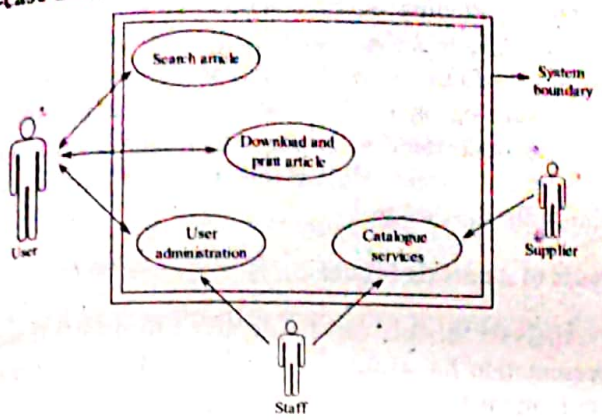


Figure: Use-case Diagram for Article Downloading

Q42. 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.

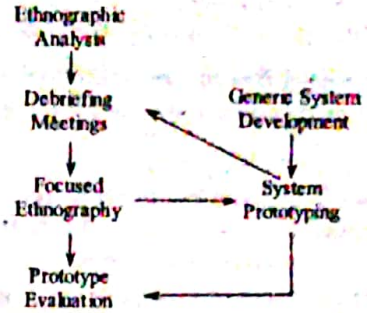


Figure: Different Steps Observed while Combining Ethnography along with Prototype

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.

Q43. Write in detail about interviewing. Explain with examples.

Answer :

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?

2.3.4 Building The Analysis Model, Negotiating Requirements, Validating Requirements

Q44. Explain about building the analysis model.

Answer : Model Paper-III, Q12(b)

The main purpose of analysis model is to provide description about all the required informational, functional and behavioral domains for computer based system. It changes when software engineers learn about the system and when stakeholders understand more about the requirements. Inspite of this there are certain stable elements which provide a solid foundation for the design.

Elements of Analysis Model

Analysis models can be depicted to different modes of representation for which the software team has to consider the requirement from different view points. The generic and common elements of analysis models are as follows,

The use case describes the typical representation of how the end user carrying out multiple roles interacts with the system under certain situations. These representations could be narrative texts, framework of task or interactions, a template-based descriptions or diagrammatic representation. Therefore, a usecase irrespective of its form, represents software or system through end user's perspective.

In the initial step of use case development, it is essential to define set of actors included in the act. Typically, an actor can be specified as a device (or people) that interacts with the system or product with in the framework of function and conduct that is to be expressed. Therefore, the actors depicts the roles that are played by the people when the system operates and each actor can play multiple goals.

End user and actor are not similar to each other. The former one performs a number of different roles whereas the latter one expresses class of external entities that performs only one role in the framework of usecase. On the other hand, during the evolutionary activity, the requirements are elicitated. This entails a problem that is, in the course of first iteration all actors are not determined. Only the primary actors are entitled to get determined. So, as the iterations increases the number of actors also increases.

The chief operation of the primary actors is to interact with the system to obtain necessary system functions and other benefits. They avail this by making direct and frequent conduct with the software. And the system behavior is generally described by the secondary actors. Also, they substantiates the system so as to simplify the work of primary actors.

After determining the actors the use case can be developed based on the following questions,

(i) Identify the primary actors and secondary actors.

(ii) Identify the goals of the actors.

(iii) State the preconditions needed prior to the initialization of the story.

(iv) State the primary jobs and functions that are to be carried out by the actor.

(v) State the necessary exceptions that are to be taken into account as the story is defined.

(vi) State the possible variations that the actors can come across during the interaction.

(vii) State the system information required by the actor so as to produce or change.

(viii) Is the actor entitled to acknowledge the system regarding the changes in the external environment?

(ix) State the necessary information that the actor requests from the system.

(x) Specify the unexpected changes that might occur during the process.

Example: For instance, consider the example of safelocker requirements. This scenario requires four actors namely,

(i) Locker owner

(ii) Setup manager

(iii) Sensors

(iv) Monitoring and response subsystem.

1. The locker owner are usually the users.
2. Setup manager behaves same as locker-owner but plays various roles.
3. Sensors are the devices that are connected to the system.
4. Monitoring and response subsystem serves as the central station for monitoring the safelocker security function.

Now consider the locker owner actor that communicates with home security function in various ways through alarm control or a PC. The various interactions include,

(i) To gain access for all the interactions, the locker owner actor enters the password.

(ii) The locker owner obtains the information about status of a security zone.

(iii) The locker owner also obtains the information about status of a sensor.

(iv) In case of emergency the actor is provided with panic button.

(v) The locker owner can activate and deactivate the security system.

Assume a scenario wherein the lockerowner makes use of screen key panel. And its usecase is as follows,

(i) At first, the lockerowner checks the screenkey panel so as to ensure whether or not the system is ready for the input. If the system is ready, the lockerowner proceeds with the work. However, if the system is not ready, it displays a not ready message on the LCD screen. Upon observing this, the lockerowner needs to close the windows or doors of the panel inorder to remove not ready message from the display.

(ii) In the next step, the lockerowner accesses the safelocker by typing four-digit password into the screen key panel. After keying the password, it is then compared with the valid password stored in the system. If the password is correct, the actor is navigated to the next screen but if the password is incorrect the system raises beep sound and reset itself to allow actor to type the password again.

(iii) In case if the password is correct, the lockerowner is provided with various options to activate the system. If the actor keys 'stay' option, it activates perimeter sensors. And if actor keys 'away' option, it activates all sensors.

(iv) Next, at the time of activation the home owner observes the red alarm light.

Basically, the use case specifies a high-level scenario stating the interation between actor and system. In addition to this, the use cases also facilitate the detail description about the interaction. The following are the given templates for detailed description of use cases with respect to locker scenario.

Use case	:	Startmonitoring
Primary actor	:	Lockerowner
Goal	:	Setting the system to monitor the sensors at the time when lockerowner takes exit from the house or remains inside.
Preconditions	:	Programming the system for a password inorder to identify different sensors.
Trigger	:	Lockerowner 'sets' the system by turning the alarm functions on.

Scenario

(i) The lockerowner checks different options in screenkey panel.

(ii) The lockerowner enters the password inorder to access the locker.

(iii) The lockerowner selects either 'stay' or 'away' option so as to activate just one sensor or all sensors.

(iv) The lockerowner checks the red alarm light to ensure that locker is activated.

Exception

(i) When the screen key panel is not ready to perform the function, the lockerowner ensures that all sensors are closed.

(ii) When the lockerowner enters an in-correct password, the system raises a beep sound indicating the invalid password and the lockerowner has to re-enter the password.

(iii) The lockerowner upon selecting 'stay' option observes a beep sound twice and 'stay' option glows indicating that perimeter sensors are activated.

(iv) The lockerowner upon selecting 'away' option observes a beep sound thrice and 'away' option glows indicating that all sensors has been activated.

Priority : The highest priority use case gets implemented.

Frequency of use : It is used several times a day.

Channel to actor : It is channel via screenkey panel interface.

Secondary actors : It includes support technician, sensors.

Channels to Secondary Actors

Technician support : It includes phone line.

Sensors : It includes hard wired and radio frequency interface.

Impediments

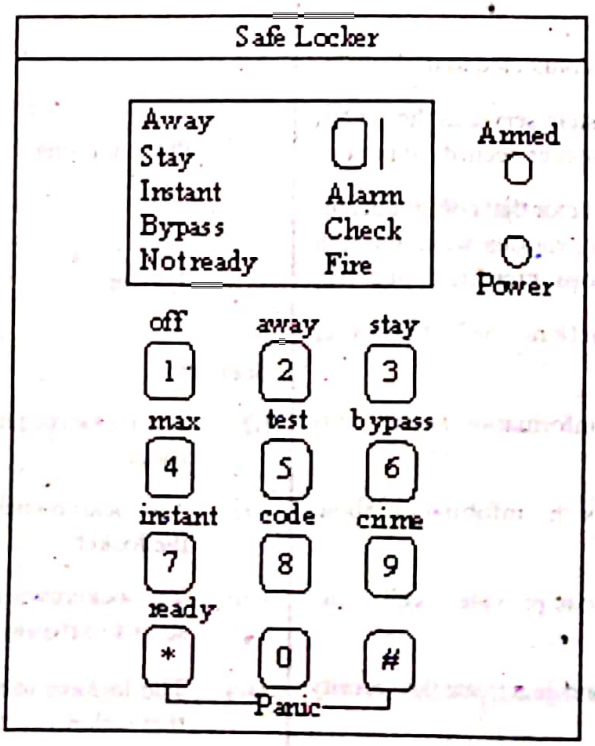
(i) There should be an alternative way to activate the system without the use of password or an abbreviated password can also be used.

(ii) There should be an additional text messages in screen key panel.

(iii) They should specify the time consumed by the lockerowner to enter the password starting form the time when the first key was pressed.

(iv) There should be a method for deactivating the system before it activates.

Each and every use case should be reviewed carefully. Error in any use case interaction entails many problems.



Analysis Pattern

The analysis pattern can be described as a solution to a problem existing within application domain. It can be reused while modeling various applications. The problem could come up while performing requirements engineering on multiple software project and the issues may keep on occurring during the entire projects in a specific application domain.

The use of analysis patterns addresses two benefits,

(i) It increases the development process of abstract analysis models that uses the main requirements inorder to solve potential problems. It facilitates reusable analysis model along with advantages, examples and limitations inorder to ease the problem.

(ii) It provides design patterns and solutions to various common problems that helps in transforming analysis model into a design model.

Moreover, the analysis patterns can be combined into analysis model with respect to pattern name. It can be stored in the repository that helps the requirement engineers to use search facilities and apply them.

Q45. Write short notes on,

(i) Negotiating requirements

(ii) Validating requirements.

Answer :

(i) **Negotiating Requirements:** An ideal requirements engineering framework illustrates that customer requirements can be determined by performing inception, elicitation and elaboration process. Such detail description makes ease in proceeding to the next level in software engineering activities. But such ideas becomes impractical while implementation. In a real time scenario, the user has to undergo a process of negotiation with multiple stakeholders. The act of negotiation entails a proper development of project plan which not just serves the needs of stakeholders but also shows real world constraints (such as people, time, budget) imposed on the software team. Therefore, the stakeholders by performing negotiations tries to balance the functionality, performance, products or system characteristics against cost and time taken for marketing.

An idealistic negotiation produces a 'win-win' situation where in the stake holders obtains maximum amount of product satisfaction that fulfills their majority of needs and a user activities budgets and deadlines.

From Boehm point of view, the negotiation activities can be defined at the beginning of each software process iteration. It includes,

(i) Define system are subsystem key stakeholders.

(ii) Find out the win conditions of stakeholders.

(iii) Determining a standard win conditions for a product associated with software process that satifies the majority needs of all stakeholders.

When all these steps are achieved the stakeholders observes a win-win situation that plays an important role in proceeding to the next software engineering activities.

(ii) **Validating Requirements:** The process of validating requirements initiates after each element of requirements model is created. The element has to underago a process for testing the inconsistency, ommissions and ambiguity. Along the same lines the stakeholders sets up the requirements produced by the model and then combine them with in requirements packages for using a software increments.

The queries that should be addressed here includes,

(i) Identify, the consistency of each requirements with overall objectives concerning with system or product.

(ii) Identify, whether or not all requirements are met at the proper level of abstraction? that is check if any of the requirements is given a technical support which is inappropriate at this stage.

(iii) Identify, whether or not the presence of requirements necessary to the model or it just behaves as an extra features which is unnecessary to the system.

(iv) Identify, if each and every requirement is bounded and error free.

(v) Identify, whether each and every attribute posses the attribution.©

(vi) Identify, whether any of the requirement contradict with other requirements.

(vii) Identify, whether all requirements present in technical environment are obtained where system or product can be placed.

(viii) Identify, whether all requirement are testable after being implemented.

(ix) Identify, whether or not the requirement model follow the information, function and the behavior of the intended system.

(x) Identify, whether the requirements model have been divided in a manner which shows detailed information about the system.

(xi) Identify, whether requirements pattern implies requirements model, are they correctly validated, are the patterns consistent with customer requirements.

UNIT 3

BUILDING THE ANALYSIS MODEL AND DESIGN ENGINEERING

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. How do analysis classes manifest themselves as elements of solution space?

Model Paper-II, Q5

Answer :

Analysis classes manifest themselves as elements solution space in the following ways,

1. They demonstrate themselves as a things such as letters, reports, displays etc., that act as information domain.
2. They demonstrate themselves as certain operations that are part of overall system operation. Explain of such operations is movements of robots to accomplish a task.
3. They demonstrate themselves as third parties including various devices, people etc., that contribute to the system by providing useful information.
4. They demonstrate themselves as organizational units such as team, group etc., associated with the application.
5. They demonstrate themselves as various characters such as managers, engineers etc., who contribute to the system by interaction.
6. They demonstrate themselves various locations such as development floor etc., at which the problem is analysed.
7. They demonstrate themselves as structures such as computers, sensors etc., that are associated with classes and object classes.

Q2. What do I need to know in order to develop an effective use case?

Answer :

After determining the actors the use case can be developed based on the following questions,

(i) Identify the primary actors and secondary actors.
(ii) Identify the goals of the actors.
(iii) State the preconditions needed prior to the initialization of the story.
(iv) State the primary jobs and functions that are to be carried out by the actor.
(v) State the necessary exceptions that are to be taken into account as the story is defined.
(vi) State the possible variations that the actors can come across during the integration.
(vii) State the system information required by the actor so as to produce or change.
(viii) Is the actor entitled to acknowledge the system regarding the changes in the external environment?
(ix) State the necessary information that the actor requests from the system.
(x) Specify the unexpected changes that might occur during the process.

Q3. Write short notes on analysis pattern.

Answer :

Model Paper-I, Q5

The analysis pattern can be described as a solution to a problem existing within application domain. It can be reused while modeling various applications. The problem could come up while performing requirements engineering on multiple software project and the issues may keep on occurring during the entire projects in a specific application domain.

The use of analysis patterns addresses two benefits,

(i) It increases the development process of abstract analysis models that uses the main requirements inorder to solve potential problems. It facilitates reusable analysis model along with advantages, examples and limitations inorder to ease the problem
(ii) It provides design patterns and solutions to various common problems that helps in transforming analysis model into a design model.

Moreover, the analysis patterns can be combined into analysis model with respect to pattern name. It can be stored in the repository that helps the requirement engineers to use search facilities and apply them.

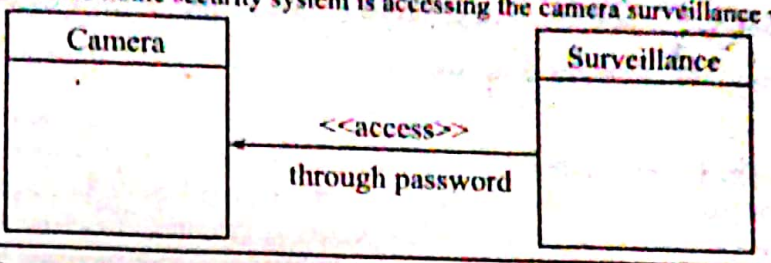
Q4. What is a stereotype?

Answer :

Model Paper-II, Q6

In Unified Modeling Language (UML), a stereotype is additional feature that can be added to an object by customizing its existing features. It is also used to define dependencies. These are included in a separate pair of double angle brackets i.e., <<stereotype>>.

Example of stereotype in a safe home security system is accessing the camera surveillance with use of password.



Q5. What are the conditions to be followed while designing a good software?

Answer :

A good software can be designed by following the given conditions.

(a) Define each requirement of the system while designing a software.

(b) The design must be properly understandable and readable by the tester who test the software, as well as by the programmers who develops the code for the software.

(c) The design must cover all the important point such as data, address, functions etc. A good software is designed for obtaining a good quality. This is possible if the software design satisfies the following conditions.

Q6. Define functional Independence.

Model Paper-III, Q6

Answer :

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 in order 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 such as cohesion and coupling.

Q7. What is the intent of information hiding?

Model Paper-III, Q5

Answer :

The purpose of information hiding is to provide the mechanism for hiding the details or descriptions of the data structures, procedural processing and implementations. The users should not be aware of the information or knowledge of the module details because it is unnecessary information for the users that leads to errors. Information hiding is needed for modifications during the software testing and maintenance because most of the complex data and procedures need to be hidden from different parts of the software system.

Q8. Distinguish between classes and components.

Answer :

Model Paper-I, Q6

	Classes		Components
	Elements Associated		
1.	Each class is associated with operations and attributes.	1.	Components are associated with only operations.
	Internal Structure		
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.
	Representation		
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.).

PART-B

ESSAY QUESTIONS WITH ANSWERS

3.1 BUILDING THE ANALYSIS MODEL

3.1.1 Requirements Analysis Modeling Approaches

Q9. Discuss in detail about requirement analysis.

Model Paper-I, Q13(a)

Answer :

Requirements analysis phase is performed after requirement elicitation phase. It is an activity performed repeatedly at the time of software development cycle. provides specification to software operational characteristics. Due to this it shows the software interface with other system elements and also sets up constraints for the software with requirement analysis. Requirement analysis consist of the activities like classification, organization, prioritization, negotiation and modeling requirements. At the time of analysis, different models are developed with the help of modeling techniques like structured analysis, object oriented analysis, data oriented analysis, rapid prototyping.

The following steps must be considered while carrying out requirements analysis,

1. Customers will not have complete knowledge of technical needs and constraints so they specify the requirements in the order as it comes to their mind. Hence requirements are categorized into distinct functions and arranged properly in meaningful clusters of modules.
2. Prioritization technique is carried out. So that the modules executed in an organized manner without delaying the project and without crossing the estimated cost.
3. Requirements are classified into essential requirements, useful requirements and desirable requirements.
4. Changes carried out at the time of development leads to inflation of cost and time conflicting, requirements are decided only after negotiating between the different stakeholders.
5. Draft requirements are included in the development only after validating the correctness and completeness.
6. Lastly all the requirements are arranged in a systematic approach with the help of modeling techniques. Some of the analysis techniques are as follows,
   (i) Structured analysis
   (ii) Data-oriented analysis
   (iii) Object-oriented analysis
   (iv) Prototyping.

(i) Structured Analysis

Structured analysis is an important aspect of system analysis. It is a technique that makes use of graphical diagrams to develop the requirements specification that can be easily understood by the users. These diagrams illustrate the steps and data that must be included inorder to meet the desired requirements or fucntionalities. Structured system analysis aims at translating the business requirements into software and hardware specifications by focusing on logical systems and functions.

(ii) Data - Oriented Analysis

Data Oriented analysis also called as data oriented modelling represents conceptual view of business requirements. Data model refers to the abstraction of data structures needed by database.

Abstraction shows the real physical environment wherein a user runs its business. Data model is free i.e., software and hardware constraints are not enforced on it, as it is done on database design.

Data models consist of data entities, associations that exist between entities and rules enforced on data for its functioning. Each data model has a functional model attached to it, which specifies how data will be processed in the system. Combination of Data model and Functional model is called as Conceptual database design.

(iii) Object-oriented Analysis

It deals with the problem domain. The focus of the object that come out of analysis is on the problem domain. These objects are referred as "semantic objects", as they often describe few things or concepts in the problem. The task of analysis activity is to find out the classes in the system and their relationship and frequently represented by class diagrams. The models built during object-oriented analysis form the basis for Object-oriented design.

(iv) Prototyping Analysis

Prototyping analysis approach is well suited when all the requirements are not known before starting the software development process. It is adopted usually at times when rapid delivery and user interaction is important. While in case of conventional approaches like process-oriented, data oriented. These approaches are more suitable when requirements are known and user interference is not required.

3.1.2 Data Modeling Concepts

Q10. Explain in detail the different concepts associated with data modeling.

Answer :

The different concepts associated with data modelling are,

1. **Data Objects:** Data objects can be described as representation of composite information. In essence it contains number of different properties or attributes. Thus, width could not be considered as a valid data object while dimensions can be counted as an object.

Examples

(a) A thing such as display, report.

(b) An occurrence such as telephone call.

(c) An external entity such as unspecified object or event that produces or consumer information.

(d) An event such as alarm.

(e) A role such as sales person.

(f) An organizational unit such as accounting department, resources department.

(g) A place such as data warehouse.

(h) A structure such as file.

Data object encapsulates only data and does not provide any reference to any operations which can invoke actions on them. It can be shown as a table which represents all the associated attributes. For instance a car which comprises attributes such as model ID, color, body type. The body shows particular instance of data object. For example chevy cruze is an instance of car data object.

2. **Data Attributes:** Data attributes specifies data object properties. It acquires any one of three given characteristics. They can be employed

(a) To name the instance of data object.

(b) To describe the instance.

(c) To provide a reference to another instance present in another table.

Additionally, there should be more than one attributes acting as an identifier. Identifiers are unique values used for identifying instance of data object. For instance, in a car data object ID number is the identifier.

Attributes for a data objects can be found by abstracting complete understanding of the problem situation. The car attributes such as model, ID number, body type, color will work suitable for motor vehicles department. But they are of no use for automobile manufacturing software department. Thus, attributes relating to different departments are different.

3. **Relationships:** Data objects are linked to other data objects in various ways for instance assume two data objects car and person. These data objects are connected to each other but the ways through which they are related needs to be determined. Thus, their relationships can only be known by thoroughly understanding the roles of person, car corresponding to the context in which the software is to be developed. For instance, Relationship between a person and car is he owns the car and insured to drive it.

3.1.3 Object-Oriented Analysis

Q11. What is object oriented analysis? Discuss the four elements of object model.

Answer :

Object Oriented Analysis (OOA)

It deals with the problem domain. The focus of the object that comes out of analysis is on the problem domain. These objects are referred as "semantic objects", as they often describe few things or concepts in the problem. The task of analysis activity is to find out the classes in the system and their relationship and frequently represented by class diagrams. The models built during object-oriented analysis form the basis for object-oriented design.

Elements of Object Model

There are four major elements of object model, they are as follows,

1. Abstraction
2. Encapsulation
3. Modularity
4. Hierarchy.

1. Abstraction

The abstraction mechanism is a powerful technique to handle the complexity where the obtained input is organized into multiple dimensions and series of chunks thereby overcoming the informational bottle neck. This process is also referred to as chunking. It deals with the generalized, idealized model of the object instead of mastering the complex object. It provides a way to manage the system components and their corresponding interactions without considering the details of building those components.

2. Encapsulation

Encapsulation is the mechanism which binds together the code and the data. This encapsulation prevents the code and data from being accessed by other code defined outside the class. Encapsulation can be defined as wrapping up of data and methods into a single unit known as a class. A class defines the structure and behavior (data and code) that is shared by all its objects. A class consists of data and code, these elements are called members of the class. The data is referred to as instance variables and the code that uses the data is referred to as methods. With encapsulation, it is possible for objects to be treated as "black boxes" with each doing a specific task.

3. Modularity

Modularity denotes to breaking up of something complex into manageable pieces. It is a technique in which complex programs are divided into several modules. By dividing it in modules, classes and objects can be separately defined for each module.

4. Hierarchy

Identifying class and object hierarchies with in the complex software system is another way to increases the semantic content of single chunks of information. The structure of the object plays a significant role as it specifies the different ways the object collaborates with another using patterns of interactions called mechanisms. On the other hand, the structure of the class equally plays significant role as it describes common structure and behavior present in the system. Here, every individual instance of specific kind of object is observed as different but it holds same behavior as other instances of same type of object when the objects are categorized into groups of related abstractions. The user can explicitly distinguish both common and different properties of various objects which masters the inherent complexity. The hierarchy identification in a complex software system is a complicated job because it demands the pattern discovery among various objects where in each one may present complicated behavior. Therefore, once the hierarchies are understood structure of complex system and understanding becomes simple.

Q12. What is top down structured design method? Explain how structured system analysis is done using data flow diagrams.

Answer :

Top Down Structure Design Method

Top down structure design method is defined as the process of breaking an entire unit into small sub units till each subunit can be written using a program instruction. The arrangement of units follows a hierarchical structure i.e., larger unit appears at the top and smaller unit appears at the bottom. The flow of control is passing from top to bottom. Moreover, the organization of components is fixed, so each component can have only one entry point as input and only one exit point as output.

Data Flow Diagrams

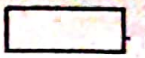
A Data flow diagram is the diagrammatic representation of various functions and the data items which are exchanged among different processes through an information system. It represents the system operations, its implementation and the tasks completed by system.

DFDs help in designing information-processing in an organization. It is also used for performing structured analysis.

Symbols Used in DFD

Data Flow Diagrams (DFDs) use several symbols to explain the inputs, outputs, storage points and routes between every destination. The symbols used in DFD are as follows,

1. *Entity*

Entity is denoted by a rectangle i.e.,  .

It represents the source/destination entity which is external to the system.

2. *Data Flow*

Data flow is denoted by an arrow i.e.,

It represents the flow of data between processes, entities and data stores.

3. *Process*

Process is denoted by a circle i.e.,  .

It represents the process that performs data transform.

4. *Data Store*

Data store is denoted by two parallel lines i.e., 

It represents the storage location of data which can be either temporary or permanent.

Example

Consider an example of Train Ticket Reservation system. Its corresponding data flow diagram using function oriented model is as follows,

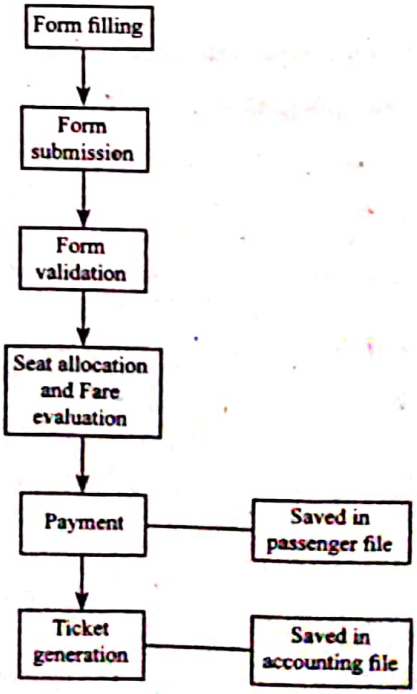


Figure: Flow Diagram of Train Ticket Reservation System

The function oriented model follows top down structure method. The function is broken down into small functions and it is continued till each sub function can be expressed with a simple program instruction. Initially, the coding is started at the bottom level and it is continued for all remaining sub-functions. Finally the code of all sub-functions is integrated and output for the main function is generated. System analysts using function oriented model can identify not only the entities and processes, but, also the data stored in them.

3.1.4 Scenario-Based Modeling

Q13. Discuss in detail the procedure of creating a preliminary use case.

Answer : Model Paper-III, Q13(a)

Creating a Preliminary Use Case: Alistair Cockburn defines usecase as contract for behavior. It can be described as the manner in which an actor makes use of a system to achieve the goal. The use case takes into account the interactions that takes place between the producers and consumers for information and system. Basically usecase explains the use of a particular situation straight forward manner with respect to actors perspective. Following are the primary queries that are frequently encountered in usecases.

1. Specify the required content to be included in the usecase.
2. Specify the quantity to be included in the usecase.
3. Specify the content description in detail.
4. Specify the way to streamline the description.

Inception and elicitation are the two requirement engineering tasks which gives the details regarding the information to be included within the usecases so as to initiate the writing process. Requirement gathering meetings, QFD techniques helps in the following areas,

1. Identifying the stakeholders.
2. Defining the scope of the problem.
3. Defining operational goals.
4. Defining all the functional requirements.
5. Defining priorities.
6. Defining things which can be changed by the system.

An example of set of use cases is as follows, SafeLocker vigilance function determines the functions which are carried out by locker owner (actor).

1. The actor selects a camera for viewing purpose.
2. The actor requests for thumbnails of all the cameras.
3. The actor views all the camera displayed on the computer.
4. The actor goes into the control panel and views a particular camera by zooming it.
5. The actor selects a part of the recording and view in the output display.
6. The actor replays the recording.
7. The actor accesses SafeLocker vigilance using the Internet.

The actor provides narrative of the SafeLocker wherein the locker owner is the usecase.

Usecase: Accessing camera using Internet.

1. The locker owner visits SafeLocker products webesiste.
2. The locker owner logins using user ID.
3. The locker owner now enters two levels of passwords, each containing a minimum of eight characters.
4. Upon entering the password the locker owner is navigated to the next screen wherein the system displays all major function buttons.
5. From all the major function buttons the locker owner selects the surveillance functionality.
6. The locker owner chooses the option "pick a camera".
7. As soon as the locker owner selects this option the system shows the plan of the area where the SafeLocker is kept.
8. Now the lockerowner selects the camera in which he is interested in.
9. The locker owner views the desired camera by selecting the view button.
10. A viewing window is displayed by the system which is found using camera ID.
11. At this point, one frame persecond view is displayed by the system on video output.

Q14. How can a designer examine alternative courses of action while developing a use case?

Answer :

Definition of alternative interactions is necessary so as to obtain thorough understanding of functions as specified by usecase. Thus few course of actions needs to be considered while developing a usecase. They are as follows,

1. At this moment will the actor be able to take some other actions.
2. What are the chances that an actor will come across some error condition.

3. What are the chances that an actor will come across some other behavior like an external event invoked which is out of actors control.

Answer to the above queries leads to the creations of set of secondary scenarios which is a part of original usecase.

(i) Will the actor be able to take some other actions at any moment?

The answer to the above query is "yes". i.e., the actor will be able to view thumbnails of all the cameras concurrently. In this case view thumbnail snaps hots for all cameras becomes the secondary scenario.

(ii) What are the chances that an actor will come across some error conditions?

The answer to the above query is "yes". Because many error conditions might occur in the functioning of a computer based system. For instance, plan of the area where SafeLocker is kept might have not been configured at any time thus selecting the option "pick a camera" leads to error condition. In this case the plan of the where the SafeLocker is kept was never configured becomes secondary scenario.

(iii) What are the chances that an actor will come across some unknown behavior?

The answer to the above query is "yes" because the system will come across an alarm condition with the occurrence of pick a camera option. Hence the system displays a special alarm notification, imparting several options to the actor related to nature of the alarm.

In this case secondary scenario is the alarm condition that occur.

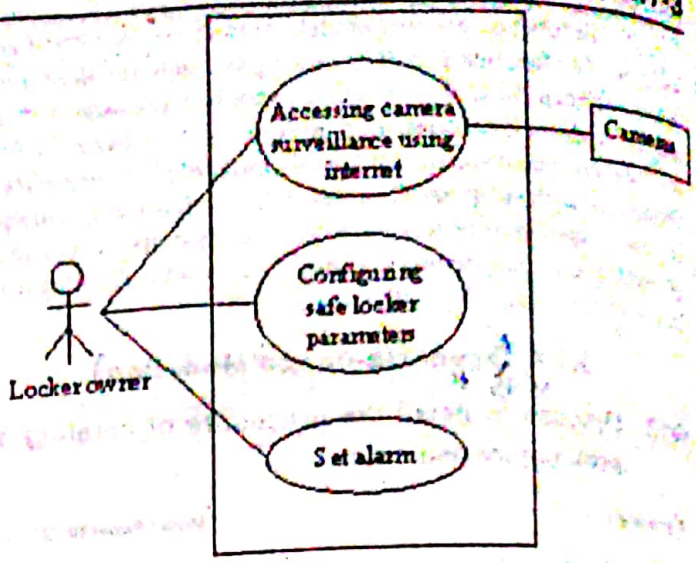
Q15. With an example, present the procedure for writing a formal use case.

Answer :

Informal use cases are at times satisfactory for requirement modeling purposes. But it may become insufficient whenever a use case includes complex activities so as to define a set of steps with many exceptions thus a formal approach needs to be adopted.

Access camera surveillance use case involves more number of formal use cases. The main goal a here is to determine the scope of use case. In use case, trigger specifies that event or condition which needs to be initiated. While precondition specifies the condition which turns to be true before the start of use case. Exceptions determine all the scenarios which are not covered when preliminary use case is refined. Scenarios present in use case describes all the actions need by the actor.

Many times graphical representation of a scenario is not required whereas the diagrammatic representation is necessary as it provides clear understanding of the use cases. Unified modelling language does not provide diagrammatic representation in case of complex scenario provide a preliminary use case consider a pretimary use case for safehome product. The use cases are shown by oval shape.



In UML, all modeling natations has some drawbacks. And use case is one among them. Because poor description of a use case can lead to confusion. It emphasizes more on behavioral and functional requirements and unsuitable for non functional requirements. Thus in case when the requirement model should have precise and detailed information then use case turns out to be inefficient.

3.1.5 Flow-Oriented Modeling

Q16. Discuss in detail about flow-oriented modeling.

Answer : Model Paper-II, Q13(a)

The data flow-oriented modeling is the traditional technique and highly used requirement analysis notation. The DFD is an input-process-output view of the system. In essence, the data objects flowing into the software gets altered at the time of processing the elements and the generated data objects flow out of the software. The representation of the data objects are done using labeled arrows and the alterations are indicated by circles. The data flow is done in hierarchical fashion. In simple terms, initial data flow model also called as level ODFD or context diagram indicates the complete system and the following data flow diagrams review (improves) the context diagram thus increasing the details.

Development of Data Flow Model

The Data Flow Diagrams (DFD) allow the users to create information domain models and functional domain. In essence, the users performs implicit functional decomposition of the system by refining the system into higher levels. Subsequently, the DFD refinement leads to the data refinement.

The following are the guidelines for DFD,

1. The data flow diagram at level-0 must represent the entire software or system as one circle or bubble.
2. The input given initially and the generated output must be noted.
3. The process of refinement must be initiated after separating candidate processes, data objects and data store which are to be presented in next level.

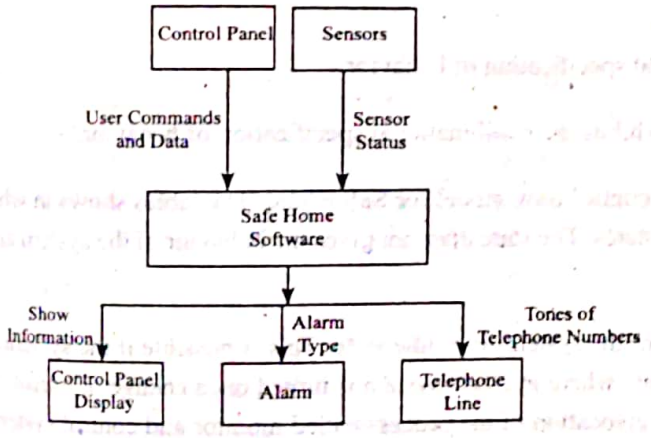
4. The arrows and circles (bubbles) must be identified with meaningful names. The continuity of the information flow should be maintained in every level.

6. The refinement of a single bubble should be done one at a time.

The concept of DFD can be easily understood by considering the DFD model of SafeHome security functions.

Using safe home security function, the home owner will be able to perform configuration upon the security system when it is installed. The function will be able to monitor sensors attached to the security system and also can interact with the home owner using internet or, a PC.

The installation process for SafeHome PC includes the programming and installation of the system. In this step, every individual sensor is allotted with the integer and the type and for arming and disarming the system, a master password is programmed each time a sensor event happens. The telephone numbers are given as input for dialing. An alarm is attached to the system and gets invoked whenever the sensor event occurs. Once the delay time is given by the owner while performing system configuration activities, the software automatically dials telephone number of monitoring service. The telephone number gets re-dialed every 20secs until the connection is made of monitoring service, facilitating, information regarding the location and reporting the type of the event occurred. The owner can check the information through a control panel using an interface. The control panel displays message and system status information.



The above DFD is a level-0 DFD for security function. The boxes are the primary external entities which provide information to be used by the system and takes information as input generated by the system. The labeled arrows in the diagram are data objects or the data object hierarchies.

The level-0 DFD can be extended into level-1 by applying "Grammatical Parsing", In simple terms, a use-case narrative is employed so as to specify the context-level bubble. This implies that all nouns, and verbs are separated in SafeHome processing narrative acquired in the initial requirements gathering. In accordance to Safe Home, the verb signifies SafeHome processes and can be represented as bubbles in the following DFD. On the other hand the nouns can be external entities (boxes), data or control objects (arrows) or also data stores (double lines). Both verbs and nouns can be associated with each other. To move further for refining the next level, a grammatical passing upon the processes for the bubble on any level of DFD is required. Upon obtaining this data, the level DFD is embedded in next level Here, 'Level-1' the context processes are divided into six processes deduced from grammatical passing. And the process of data flow among the processes is given by parse. Thus maintaining the flow of information between level-0 and level-1.

Now the level-1 processes are refined into lower levels such as monitor sensors are refined into level-2 DFD. However, it is important to note that the flow of information should be maintained among the levels. The process of refinement goes on until every single bubble carries out a simple function. In essence, the bubble performs function which could be easily employed as program component.

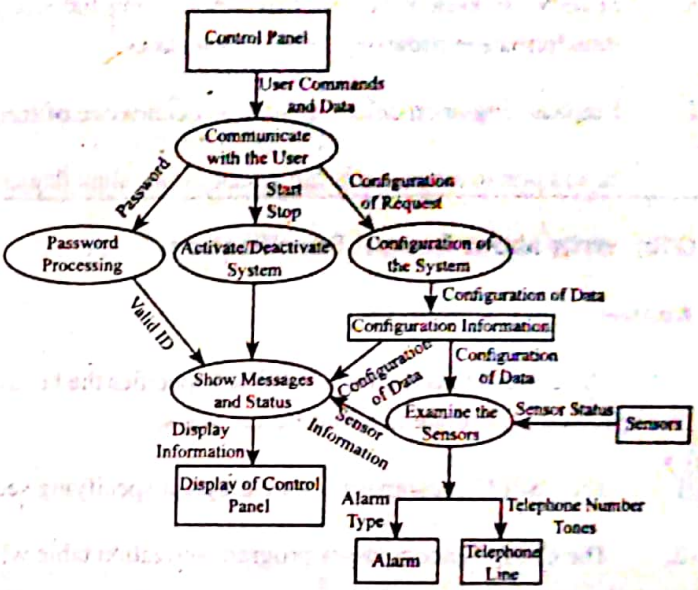


Figure: Working of Sofe Home Security in Level-1 DFD

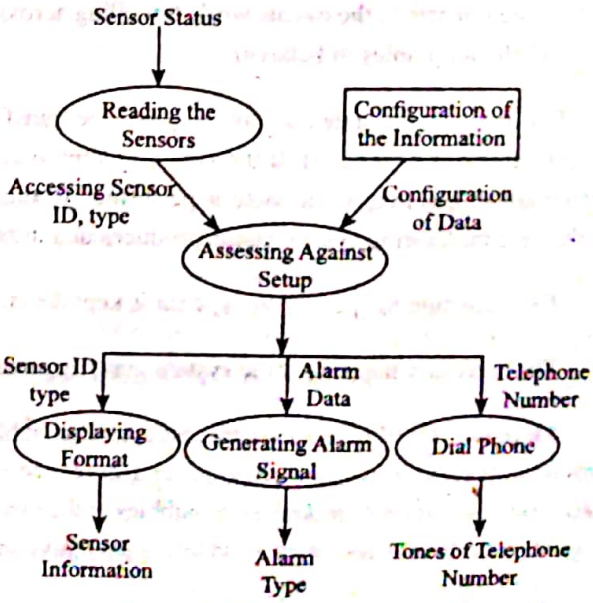


Figure: Refining Level-2 DFD to Examine the Sensor Process

Q17. Write briefly about Development of Control Flow Modeling.

Answer :

The control flow modeling can be used for large class of applications which are driven by events instead of data. It gives control information in place of reports or displays and the processing of the information is done with respect to home and performance. Thus, apart from data flow modeling, the applications also demand control flow modeling. The following guidelines are used in order to choose the correct candidate events.

1. Mention different sensors which are 'read' by the software.
2. Mention different interrupt conditions.
3. Mention different existing 'Switches' activated by operator.
4. Mention different conditions in data.
5. Recollect all different nouns and verbs used for processing narrative, reviewing different 'control items'.
6. Specify the system behavior by determining the states, determining the methods to attain the state and describing the transformation occurring between the states.
7. Emphasizing upon deletions such as occurrence of frequent words which describes the control.

In addition to other events the sensor event, blink flag and start/stop switch is considered as an entity of SafeHome software.

Q18. Write about Control Specification.

Answer :

The Control Specification (CSPEC) specifies the behavior of the system from the level from which it has been referenced. The specification is done in two different ways.

1. The CSPEC encompasses state diagram specifying sequential specification of behavior.
2. The CSPEC encompasses program activation table which exhibits the combinational specification of behavior.

The figure given is the basic state diagram showing level-1 control flow model for SafeHome. The tables shows in what way the system reacts to the events while travelling across the four states. The state diagram gives the behavior of the system and also shows the loop holes in behavior.

For instance, the state diagram suggests the transformation of the system from idle state. This is possible if the system is reset, activated or switched off. If the system is kept in activated state where in alarm system is turned on, a change or transition to the monitoring system status state is performed resulting in the invocation of the process called monitor and control system. with this, the monitoring system status produces dual transition.

1. The transition happens if the system is kept deactivated, thus the system returns to the idle state.
2. The transition happens if the system gets triggered into Acting on Alarm states.

Moreover, CSPEC specifies process activation table (PAT) which is discrete mode of behavioral representation. It specifies information present in the state diagram in accordance with only processes but not states. In essence, the job of the table is to produce the invocation of processes or bubbles at the time of event occurrence in the flow model. Apart from this, the PAT serves as a guide used by the designers for creating executive for managing and controlling the process at the level.

The CSPEC facilitates just the behavior of the system and does not specify the internal working of the processes activated due to this behavior.

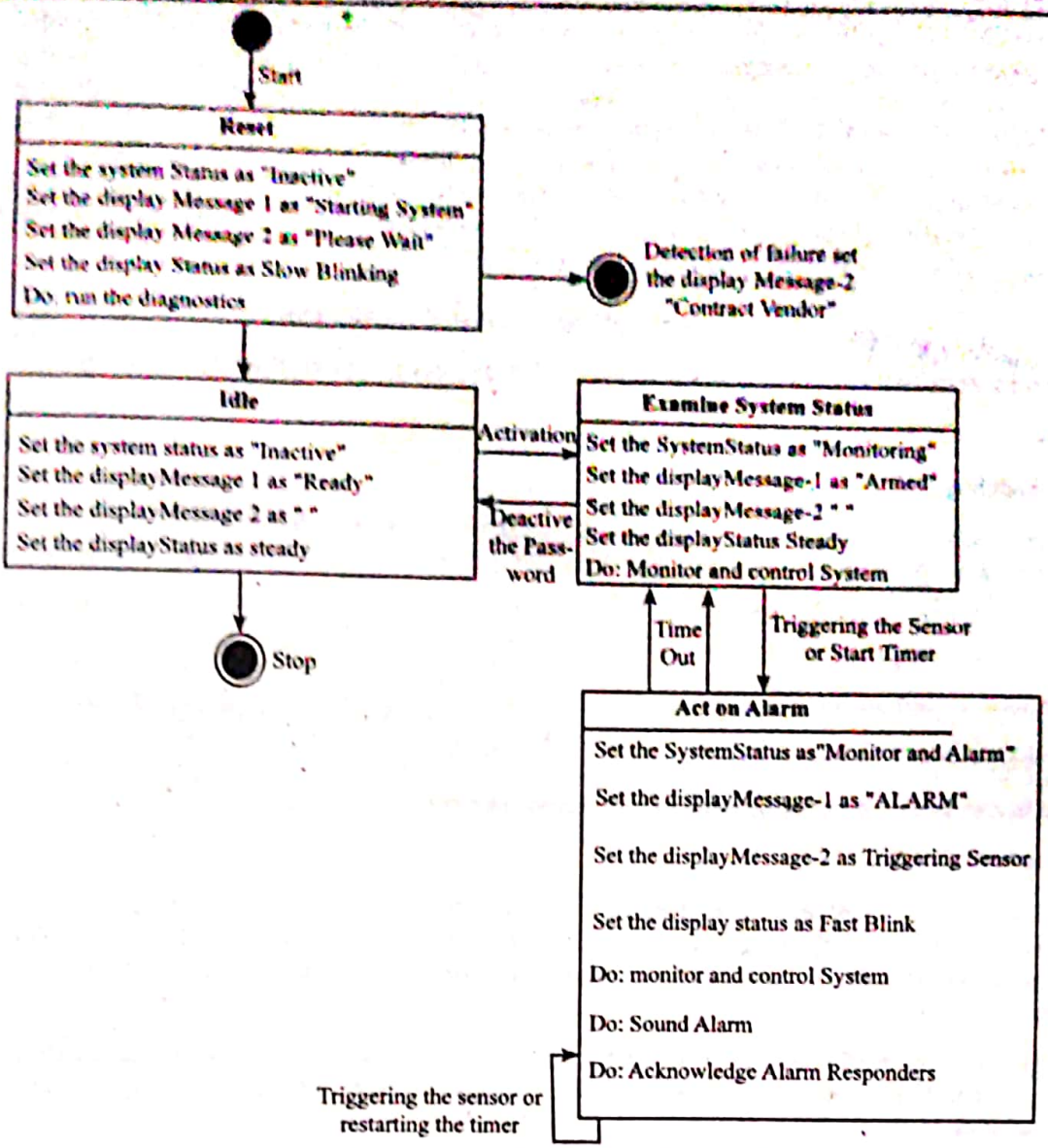


Figure: State Diagram for SafeHome Security Function

Input Events	Output	Activation Process
Sensor event 0 0 0 0 1 0 Flag blinking 0 0 1 1 0 0 Start/Stop Switch 0 1 0 0 0 0 Show complete action status 0 0 0 1 0 0 Progressing 0 0 1 0 0 0 Time Out 0 0 0 0 0 1	Alarm Signal 0 0 0 0 1 0	Examining and controlling the system 0 1 0 0 1 1 Activation or deactivation of system 0 1 0 0 0 0 Show messages and status 1 0 1 1 1 1 Communicate with user 1 0 0 1 0 1

Figure: Activation Process Table for SafeHome Security Function

Q19. Write in short about process specification.

Answer :

The process specification (PSEC) encompasses the different flow model processes used in the last level of refinement. The component of the PSEC is narrative text, program design language (PDL) defining the process algorithm, mathematical equations, tables, UML activity diagrams. A mini-spec is created by when PSPEC occur with every bubble in the flow model. This mini-spec is treated as a guide, used by designers to develop software component to use bubble.

The use of PSPLC can be seen by using process password transformation present in flow model for safeHome. The user initiates by performing password validation of process password at the control panel for safeHome security function. The user function feeds a four digit password into process password. Subsequently, the comparison of the password is made with the master password present in the system. If the match is made then <valid id message true> message is sent to the message and status display function. On the other hand if the match is not made, then the comparison of four digit password is made with the secondary password table. This is given to house guest or workers who want entry in the house. Here if match is found with the tab entry in the, <valid id message true> message is sent to the message and status display function and if no match is found then <valid id message = false> is sent to the message and status display function.

3.1.6 Class – Based Modeling

Q20. (Explain in detail about class-based modeling) Also, discuss on identifying analysis classes.

Answer :

Class-based Modeling

Class-based modeling is a modeling technique that is used for representing makes use of certain elements including classes, objects, their attributes, operations, collaboration diagrams, CRC model (Class-Responsibility-Collaborator) and various packages to represent the following.

1. Objects in the system.
2. Various services carried out with these objects.
3. Relationships among various objects.
4. Classes that can be used in combination to carry out certain operation.

Identifying Analysis Classes

In a software environment, it is a complex task to identify the objects and classes employed in a system. This is because they are not visible as physical objects and need to be identified based on their bahaviour. This can be done by considering the usage scenarios employed at the time of modeling requirements. In addition to this, grammatical parse can also be applied on various use cases.

One way of identifying classes is to create a table which carries all the nouns (or) noun pharses along with heir synonyms included in the system. If the noun is used in preparing the solution, it is included in a solution space and if it is used to provide support to the solution by describing it, then it the noun is used in problem space.

To identify the classes in an efficient way, they need to be named carefully without including a procedural name. This because these types of names are considered for defining operations and it is possible that conflicts may arise with use of such type of names.

Q21. How should a designer determine whether a potential class should, infact, become an analysis class

Answer :

A potential class can be included in analysis model if an only if the class possess the following characteristics.

1. The information related to the potential class must be retained so as to perform system operations effectively.
2. It must have set of needed services (operations) with which the values of the attributes can be changed accordingly.
3. It must contain attributes, which can be used irrespective of the classes.
4. It must contain common attributes which can be applied to every instances of the class.
5. It must contain common operations which can be applied to the every instance of the class.
6. It must contain all the essential requirements related to entities such as devices, people, other systems etc., which required in the requirements model.

For efficient use of potential class in requirements model, it needs to satisfy all the above mentioned conditions. Further the classes are evaluated to include potential classes, it might result in the discarding of objects. For example, consider SafeLock security system. In this system the inclusion and exclusion of potential classes depend on the following list.

	Potential Class	Accepted/rejected	Reason
1.	Owner of the house	Rejected	Characteristics 1 and 2 fail to be applied.
2.	Sensor	Accepted	All the characteristics can be applied.
3.	Primary password	Rejected	Characteristic 3 fails.
4.	Contact number	Rejected	Characteristic 3 fails.
5.	Control panel	Accepted	All the characteristics can be applied.
6.	Monitoring	Rejected	Characteristics 1 and 2 fails.

Q22. Write short notes on,

(i) Specifying attributes

(ii) Defining operations.

Answer :

(i) Specifying Attributes

The characteristic (or) property which describes a class included in requirements model (or) in the problem space is known as an attribute of that class. A single class might carry different attributes in different context. An example of such a class is the data associated with cricket player. The attributes of this class will be different in context of the player's career statistics and pension statistics. The attributes associated with 'cricket players' class in context of career skills include player name, average batting score, wickets taken, catches taken, total number of matches etc., whereas, in context of pension system, the attributes include player name, salary, pension plan, E-mail, contact number etc.,

It is necessary to select attributes carefully by considering each use case to get maximum benefit of these attributes. For instance, consider SafeLocker security system where the owner is responsible for selecting the type of configuration. The system class in this system carries the following attributes.

1. System ID, contact number, status [related to identification]
2. Password, number of attempts allowed, one time password for recovery [related to activation/deactivation].

(ii) Defining Operations

Operations are nothing but the description of how objects behave in the system. They are broadly classified into four categories.

1. Operations that perform manipulation on data such as selection, addition, deletion.
2. Operations that perform computation.
3. Operations that maintains the current state/position of the object.
4. Operations that monitors the objects while the events are being triggered.

It is necessary for an operation to have complete information regarding the nature of class attributes and associations inorder to perform the above functions. Selecting operations is similar to selecting attributes by considering each use case carefully. Moreover, grammatical phrase is also considered for isolating verbs among which few can be directly assigned to a class. For instance, consider the SafeLocker security system in which assign( ) operation directly related to the sensor class because it involves assignment of a number and type. Another example of such operations in the some system is the use of arms( ) and disarm( ) operations that is directly related to the *system* class for alarming and disalarming.

Some operations such as program( ) in the same system might consider certain suboperations such as sensor table creation, configuring alarm etc.

Q23. Explain in detail about CRC modeing. Also discuss in detail the three sections of index card.

Answer : Model Paper-II, Q14

Class

Responsibility-Collaborator modeling is a technique that identifies and organizes the classes which are required while developing a system or a product. According to Ambler, CRC technique comprises set of index cards that are representing those classes which are relevant to system requirements. These cards contains three sections,

Section 1: It contains an area, wherein the name of the class is written.

Section 2: It contains the body of the card that gives the list of class responsibilities. These are the attributes and operations relevant to the class.

Section 3: It contains the information abut the collaborators, which are basically the classes required for a class with information that is required for completing a responsibility.

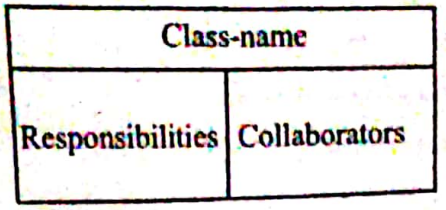


Figure: Index Card

Example

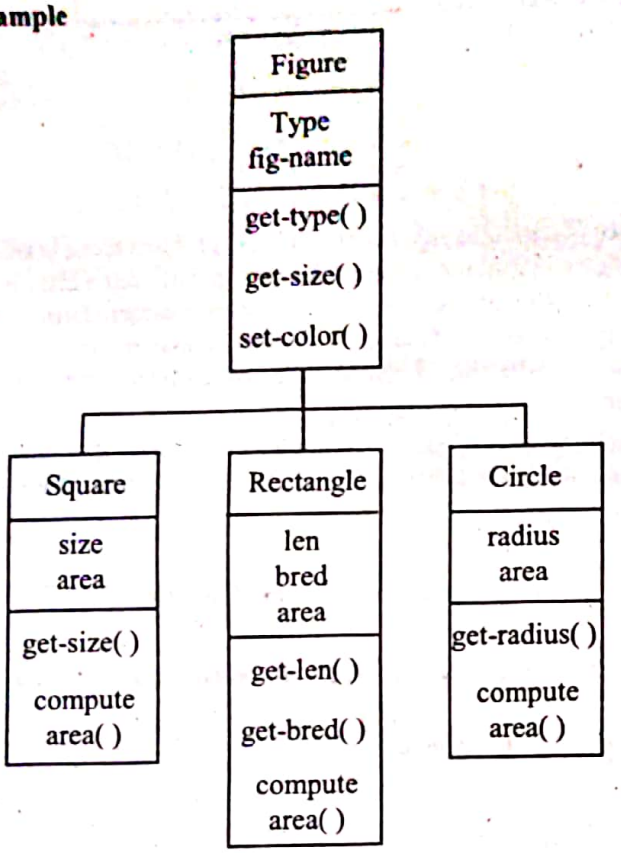


Figure: Class Diagram for Figure

Sections of Index Card

1. **Classes:** The different categories of classes are,

(a) **Entity(Model/Business) Class:** This sort of class represents the objects archived in the database. Such objects exist throughout the application life-time.

(b) **Boundary Class:** This sort of class creates the interface which helps the user in viewing and interacting with the software.

(c) **Controller Class:** This sort of class manages the part of work for starting till the end. In other words, the responsibilities of controllers class is to manage,

- ❖ The creation of entity objects.
- ❖ The instantiation of boundary objects.
- ❖ The communication between the objects.
- ❖ The validation of communicated data.

2. **Responsibilities:** The guidelines that are to be followed while allocating responsibilities to classes are,

(i) **Evenly Distribute the system Intelligence:** Responsibilities of a system is distributed among various dumb and smart classes. Dumb classes carry less number of responsibilities whereas, smart classes carry large number of responsibilities. This system is considered as ineffective even if the dumb classes are used as subclasses of smart classes. This is because the responsibilities are restricted to some of the classes only with which the modification in the system becomes difficult. To overcome this issue, the distribution of intelligence is made in such a way that every class gets an equal part of intelligence. This eliminates the use of dumb and smart classes and provides improved cohesiveness and allowing flexibility in making modification. This equal distribution is made possible by employing CRC cards with which the responsibilities associated with each object can be monitored. If CRC of an object carry large number of responsibilities, actions can be taken to distribute them among various classes,

(ii) **Specify Responsibilities as General:** The general responsibilities should be included in the class hierarchy itself.

(iii) **Maintain Information and Behaviour in Same Class:** The information and behaviour associated with objects should be included in the class to which they belong which provides encapsulation. Moreover, a cohesive unit is made where data and processes are stored.

(iv) **Localize the Information Pertaining to an Object with a Single Class:** Information associated with an objects must be kept in a single class and must not be distributed. The distribution of such information makes the system more difficult to test and maintain.

(v) **Share the Responsibilities Among Related Classes:** The classes that are related to each other because of the usage of same attributes must share the responsibilities among each other.

3. **Collaborations:** According to Wirfs-Brock, collaborations is defined as a request originated from the client and fulfilled by the server. It is generally an agreement/ collaboration between the requesting object(client) and the server. The purpose of this collaboration is to fulfill the responsibility. Collaborations can be identified by examining whether a particular class is capable of fulfilling the responsibility or not. If the class is incapable, then it is required that, the class interact with another class. A part from this, the following three generic relationships helps in identifying the collaborators.

(i) Is-part-of relationship

(ii) Has-knowledge-of relationship

(iii) Depends-upon relationship.

(i) **Is-part-of Relationship:** This type of relationship connects the classes that are part of aggregate class with an aggregate class.

(ii) **Has-knowledge-of Relationship:** This type of relationship acquires information from another class.

(iii) **Depends-upon Relationship:** This type of relationship defines dependency between the pair of classes.

After the completion of CRC model, the following approaches are employed by stakeholders inorder to review the developed model,

- A small portion of index card is allotted to every individual participant, who then isolates the collaborating cards.
- Organize the use-case scenarios into different categories.
- Pass the corresponding class index card to the responsible participant once the name of the object is known.
- Describe the responsibilities specified on the card.
- Modify the card if it is not possible to accommodate the listed responsibilities and collaborations within the use case.

Q24. Discuss in brief about analysis packages.

Answer :

Analysis package is a package that categorizes different elements of analysis model. Each of the package is given a representative name.

Example: Consider the concept of hospital management. The analysis model for this, include the many classes, which are categorized as following packages depending on different perspectives.

- Ambience
- Doctors
- Facility
- Pharmacy
- Machinery
- Tariff plan

Different classes under each package is shown below,

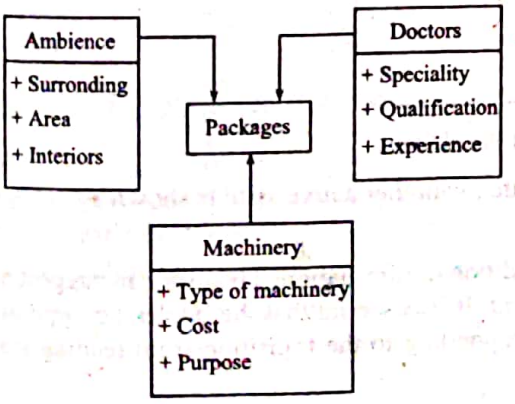


Figure: Packages in Hospital Management

In the figure, the tile indicates the name of the package and '+' sign indicates that the class is visible to public. In case that a class carry '–' sign, it indicates that the class is hidden and a '#' indicates that element can only be accessed by classes of that package only

3.1.7 Creating a Behavioral

Q25. Discuss in detail the behavioral model creation.

Answer :

The behavioral model represents the behavior of the software to the external events. In essence, it shows the way the software reacts to external events or stimuli. The steps required to create the model are:

1. Estimate different use - cases in order to clearly understand the series of communications occurring in the system.
2. Determining the events, managing the sequence of interactions and also examining the ways adopted by the events for associating with specific objects.
3. Developing sequence for every single use-case.
4. Creating the state diagram for the system.
5. Performing review operation to check accuracy and consistency.

Identifying the events with the use case

The use-case involves series of activities which includes actors and the system where as event describes the exchange of information between system and the actor.

According to the studies, the event does not specify the exchanged information instead, it is a fact which shows that the information is exchanged.

The use case is checked at the points of information exchange. The entire concept can be easily understood by referring the use case for a particular part of the Safe Home security function.

The homeowner keys in a four-digit password using the keypad on the control panel. Now, the comparison of the password is made with the valid password stored with in the system. The control panel raises a beep sound and reset itself if the password does not matches, but if the match is found, the control panel is ready for next processing.

In the above scenario, the underlined text represents the events. For each event, the actor gets identified and the exchanged information is recorded. Also, the listing of conditions or constraints must be mentioned clearly. From the underlined use-case phrase "The homeowner keys in a four-digit password using the keypad on the control panel", it can be deduced with respect to requirement model that the object Homeowner is responsible for sending event to the object control panel.

This event is referred to as password entered. The transferred information is nothing but a password in a 4-digit which is considered as non-essential commodity. Interestingly, the emphasis is made on some events having explicit effect upon the control flow of the use-case where as rest of the events does not directly effect the control flow.

For instance, the password entered event doesn't make an explicit transition in the use-case control flow however the outcome of the event i.e., password compared will surely create an explicit effect an not just the information, but also the flow control of the SafeHome software.

After determining all the events, all the assignment of events to the objects takes place. Therefore, the objects generates events such as Homeowner creates the password entered events or identify the already occurred events such as control panel identifies binary results of password compared events.

Q26. Discuss in detail about state representations.

Answer :

There are two different categories of states with respect to behavioral modeling.

1. The state of each and every class at the time when the system performs its functionality.
2. The state of the system determined externally at the time when system performs its functionality.

The state of each and every class includes passive and active characteristics. The former one includes the present status of object attributes where as the latter one also represents present status of the object while performing consistent modifications or processing. For instance, the passive state of class player could be present position, orientation attributes of player and various other features of the player required in the game. On the other hand, active state of the player could be in moving state, at rest, injured, trapped lost. The occurrence of the event often called as trigger, enforces the object to change the state from active state to another state.

The behavioral representations can be of two types:

1. The type one represents the way how every single class changes its states depending upon external events.
2. The type two represents the software behavior based on the time.

State Diagrams for Analysis Classes: The active states in every class and events is represented by UML diagram. For instance, consider state diagram which corresponds to control panel object in SafeHome security function.

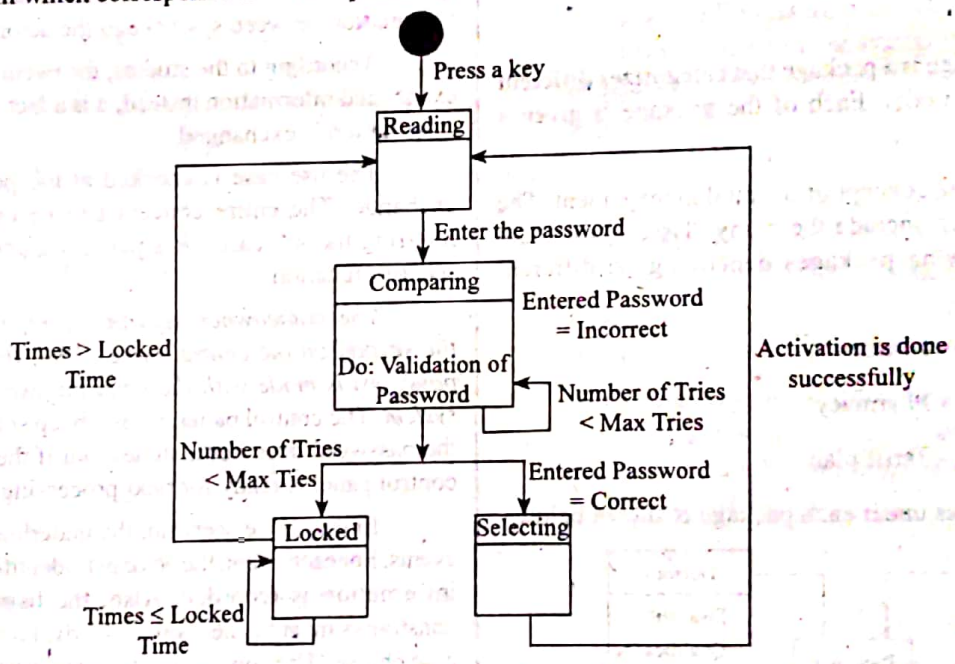


Figure: State Diagram for the Control Panel Class

In the above illustrated state diagram, the transition of one active state to another active state is shown by arrows and the labels conforms to the event triggering the transition.

The behavior of the object can be more understood by providing additional information. The user can suggest a boolean condition which is called as guard and an action apart from specifying event. It is essential that the boolean condition must be satisfied for the occurrence of the transition. For instance, the guard corresponding to the transition from reading state to the comparing state is given as.

if (password input = 4 digits) then compared to the given password.

Basically the guard corresponding to transition include values of multiple attributes of object. Thus, emphasizing upon passive state of object. On the other hand, an action happens in parallel to the state transition or as a resultant of the transition. It usually suggests multiple operations of object.

For instance, the action password entered event is called operation referred as valid password() which uses password object and carries out step-by-step comparison for validation process.

Sequence Diagrams: The sequence diagrams represents the way the event makes the transition from one object to another object. The complete transition is determined by using use-cases and the sequential diagrams can be viewed us a function of time. In simple terms, it is a short hand version of the use cases consisting of key classes and event forcing the behavior flow.

Example: The figure illustrates the sequence diagram for SafeHome security function. Arrows shows events and depicts the behavior of SafeHome objects. The time is indicated by narrow vertical rectangles.

Initial event system 'Ready' is taken from external environment and conveys the behavior to the homeowner object. Once the homeowner types the password, the request lookup event is sent to the system. This event enforces the system to check the password in the database. If the result is obtained, it returns the data to the control panel. A password = correct event is generated to the system if the password is valid. This action activates sensors of the system with the request activation event. At last using activation successful event, the control is transferred to the homeowner.

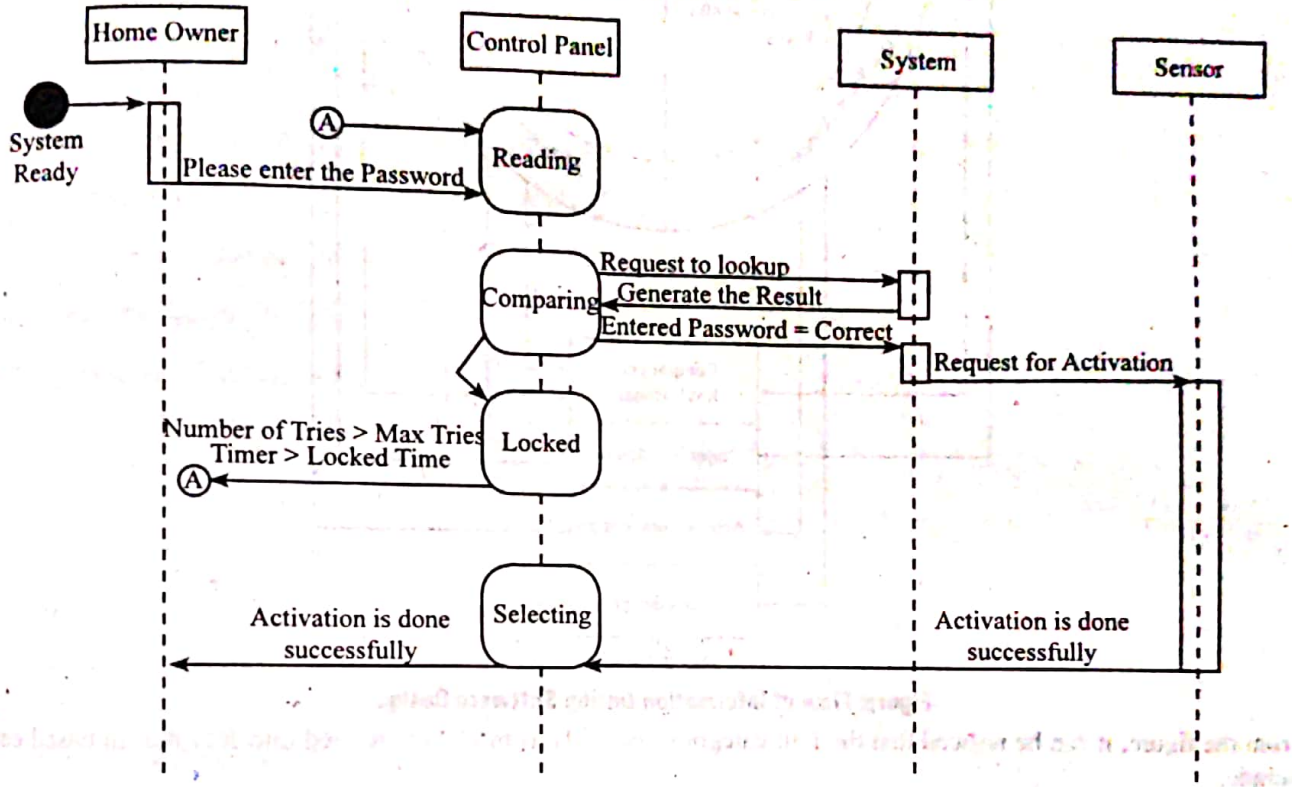


Figure: Sequence Diagram for Safe Home Security Function

3.2 DESIGN ENGINEERING

3.2.1 Design Within the Context of SE

Q27. Draw and explain the flow of information during software design.

Answer : Model Paper-I, Q13(b)

Software design is considered as the final activity in modeling a software after which, the software engineering proceeds towards coding and testing phases. At this stage, the requirements model is refined into design model by categorizing those requirements into four types of elements. They are,

(i) Scenario-based elements

(ii) Class-based elements

(iii) Behavioral elements

(iv) Flow-oriented elements.

The flow of information during this phase can be described in terms of figure as follows,

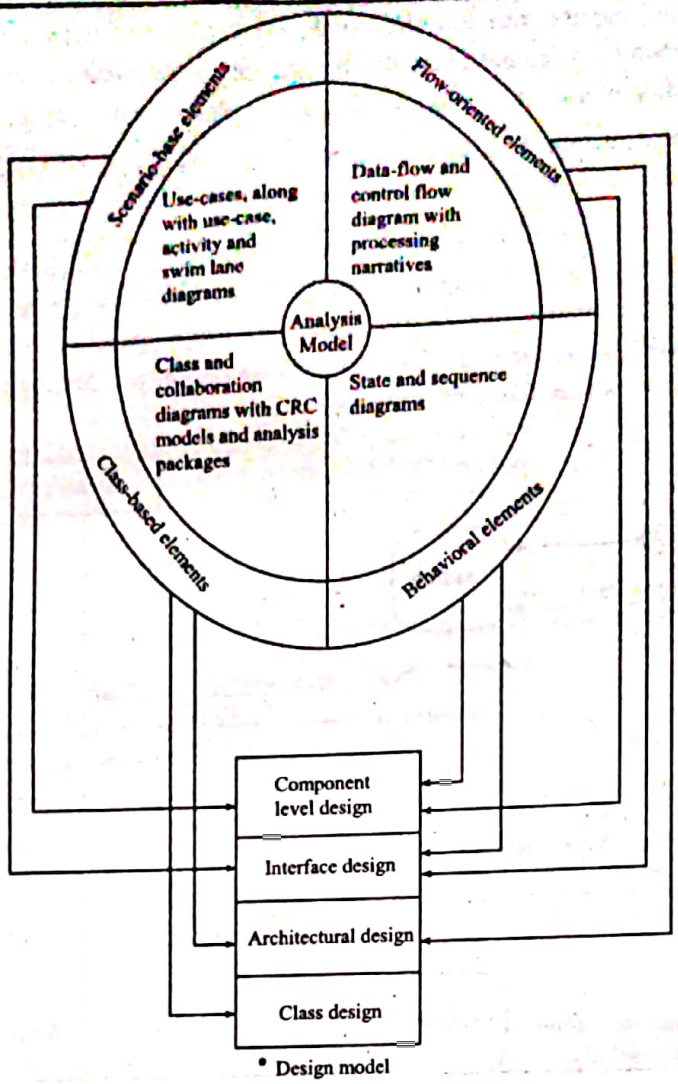


Figure: Flow of Information During Software Design

From the figure, it can be noticed that the four categories of analysis model are refined into four design based categories which include,

(i) Class design
(ii) Architectural design
(iii) Interface design
(iv) Component-level design.

(i) **Class Design:** The class-based elements including the objects and relationships associated with CRC models are transformed into class design. It also include data associated with all the attributes of class-based elements. The depth of design class depends on the design of every component.

(ii) **Architectural Design:** This part of software design model include design patterns and structural elements that results from the requirements model. It also considers the reasons with which the implementation of software design could get restricted. As the architecture depends on the framework, this part is totally related to requirements model.

(iii) **Interface Design:** This part of software design defines interaction of system with software and its users. It considers three categorizes of elements of requirements model. i.e., the flow oriented elements, scenario-based elements and behavioral elements.

(iv) **Component-level Design:** Here, the software elements are transformed into procedural description interms of software. It also considers the same three categories used by interface design.

This transformation for the flow of information is used to improve the quality of the software product. This is because it provides the following information,

(a) Various representations of software.
(b) Transforming requirements into product.
(c) Minimizes the risk by providing complete blueprint of the resultant product.

3.2.2 Design Process and Design Quality

Q28. What is software design? Discuss where exactly design comes in software engineering.

Answer :

Software Design

Model Paper-II, Q13(b)

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.

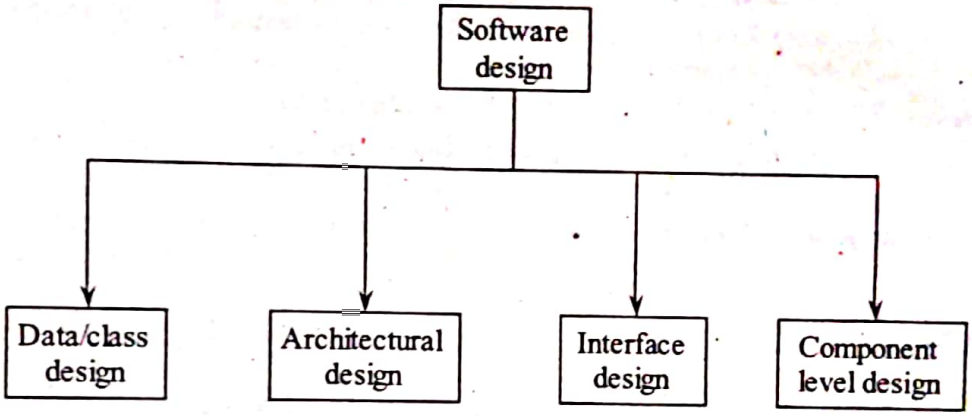
Designing can generally be classified into five types.

(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.

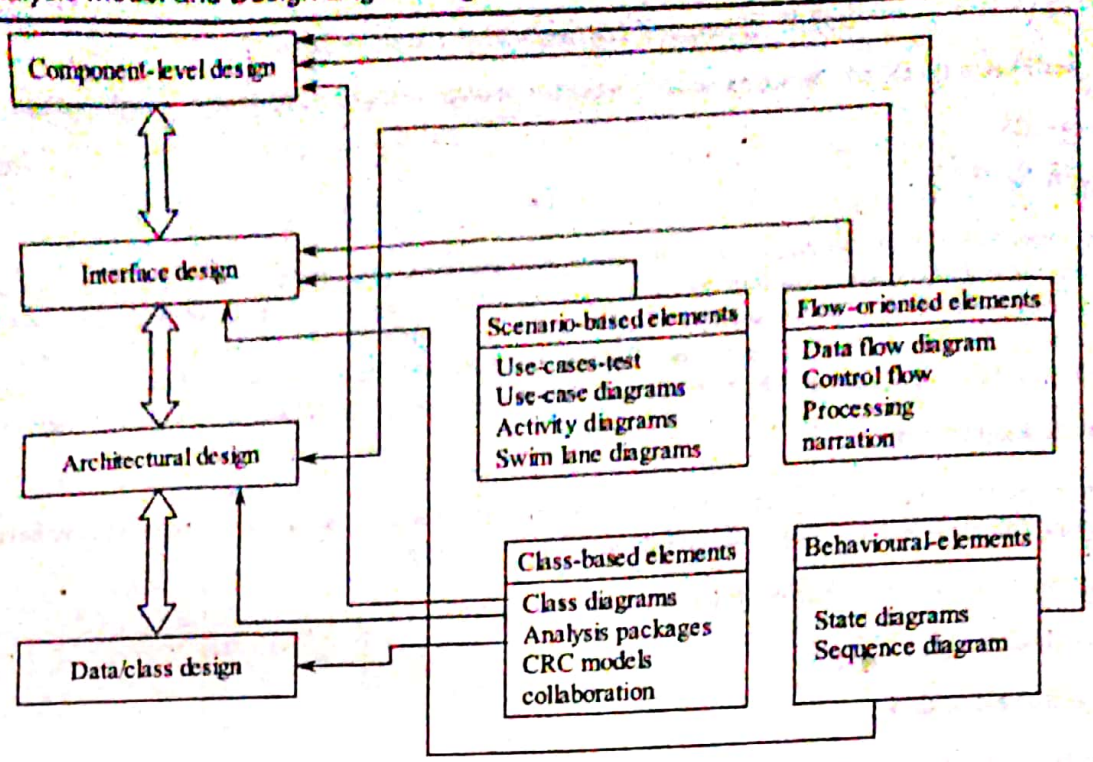


Figure: Transformation of Analysis Model into Design Model

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.

Q29. 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).

Q30. Write about the following,

(a) Design quality

(b) Product-line software.

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.

Q31. Explain various software quality standards and discuss how to assure them.

Answer :

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.

Q32. "Data modeling can be viewed as a subset of OOA." Comment on this statement and justify your comments.

Answer :

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.

Q33. Explain the following five component characteristics:

(a) Standardized
(b) Independent
(c) Composable
(d) Deployable
(e) Documented.

Answer :

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.

3.2.3 Design Concepts

Q34. 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.

Q35. Define modularity. Discuss the ways of achieving effective modularity. Write advantages and disadvantages of modularization.

Answer :

Modularity

For answer refer Unit-III, Q34, 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, Q34, 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.

Q36. Define refactoring. Explain it's intent. Also explain the advantages and disadvantages of it.

Answer :

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.

Q37. 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.

Q38. 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.

Q39. 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.

3.2.4 The Design Model

Q40. How to translate the analysis model into the design model? Explain with an example scenario.

Answer :

Model Paper-III, Q13(b)

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.

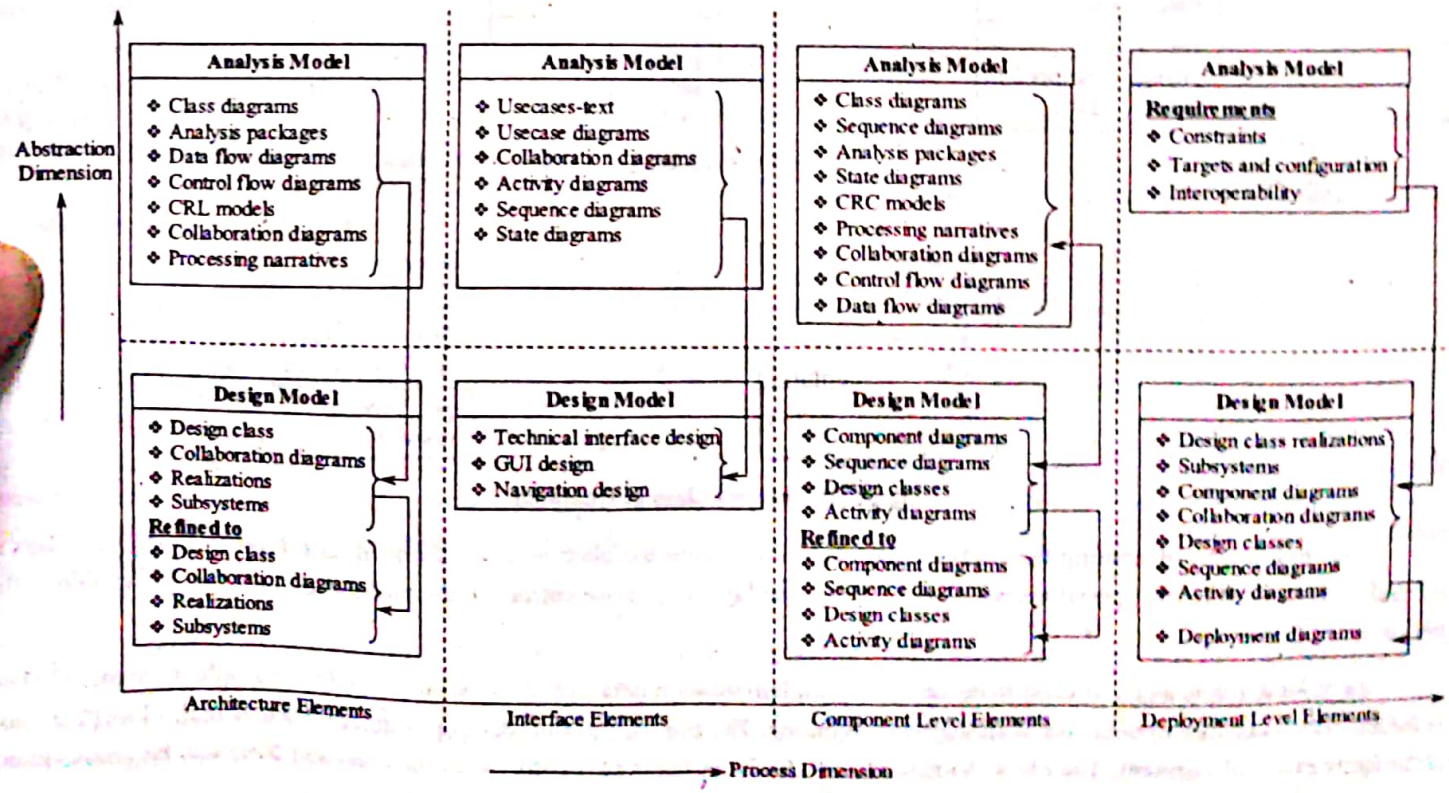


Figure: Design Model

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".

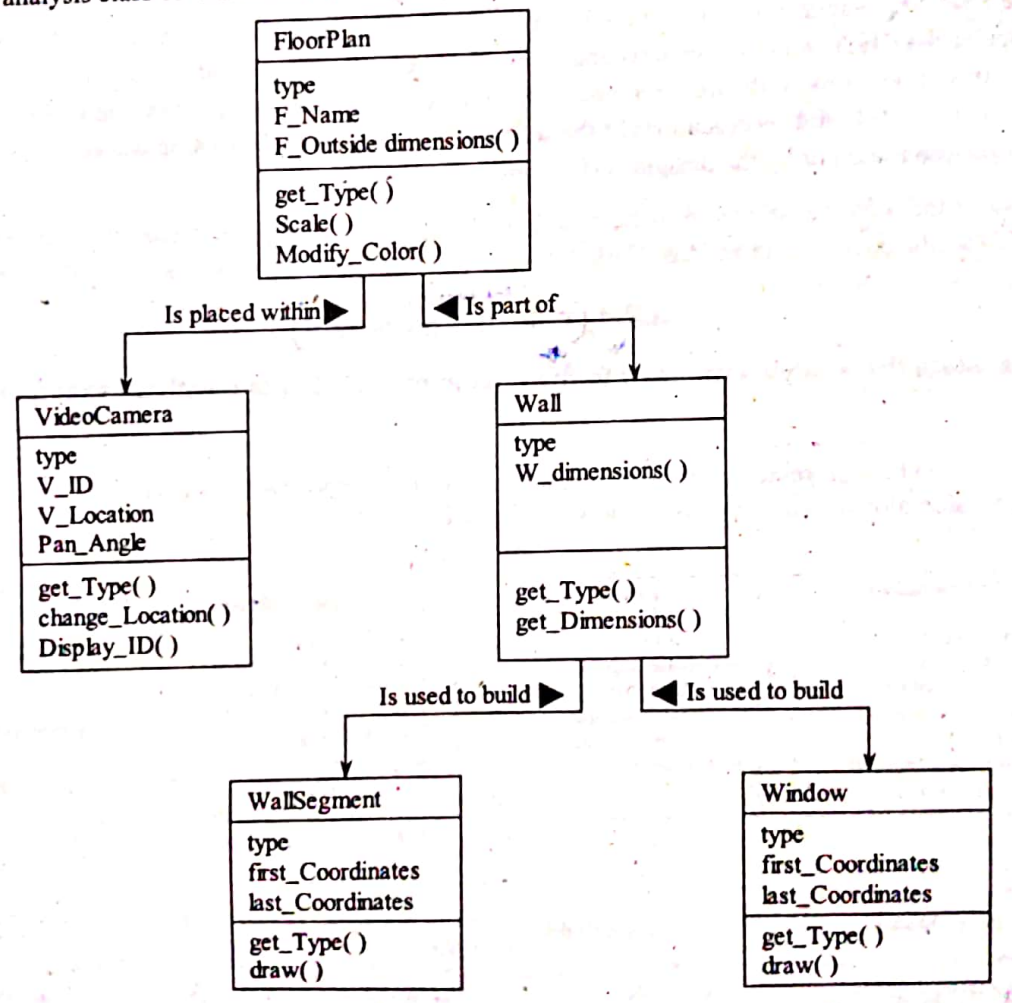


Figure (1): Analysis Class of FloorPlan

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).

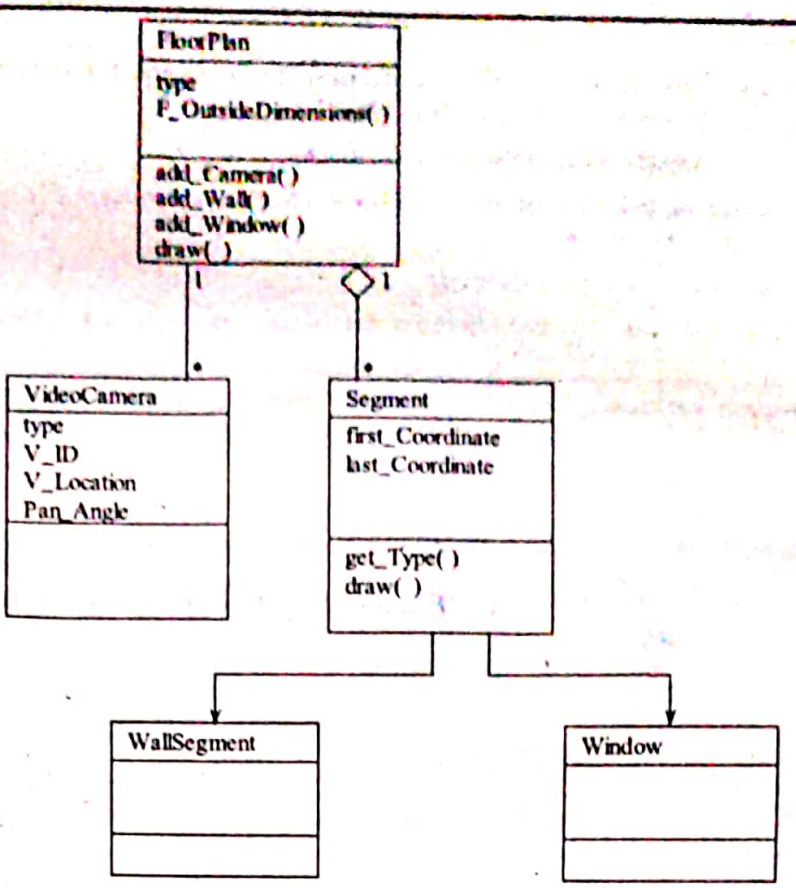


Figure (2): Design Class for FloorPlan

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.

Q41. 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.

Q42. Define Interface. Discuss various types of interfaces. Give examples for each.

Model Paper-II, Q14(a)

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.

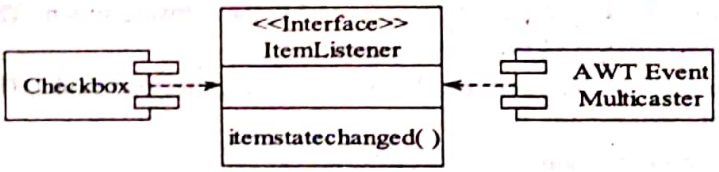


Figure (1): Representation of Interface in terms of UML Notation

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

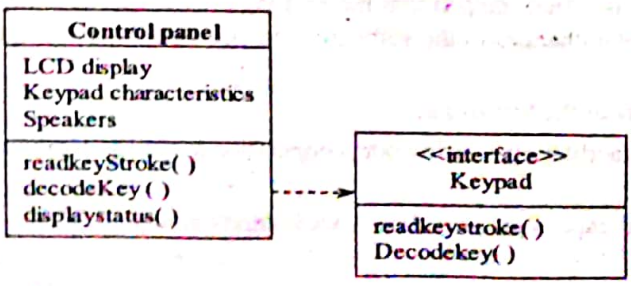


Figure (2): User Interface

Example of External Interface

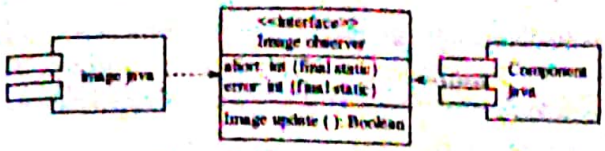


Figure (3):External Interface

Example of Internal Interface

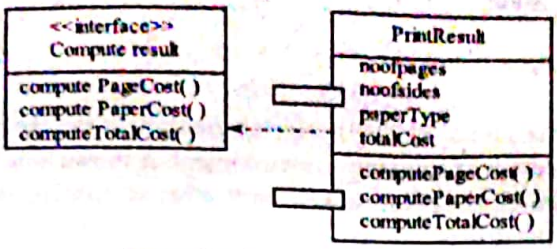


Figure (4): Internal Interface

Q43. Explain the component from traditional view with an example.

Answer :

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.

1. Object-oriented View

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.

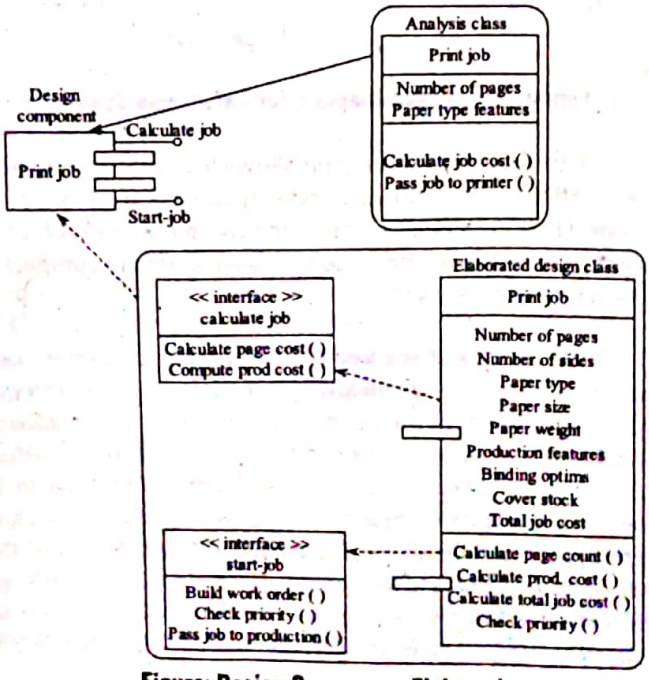


Figure: Design Component Elaboration

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.

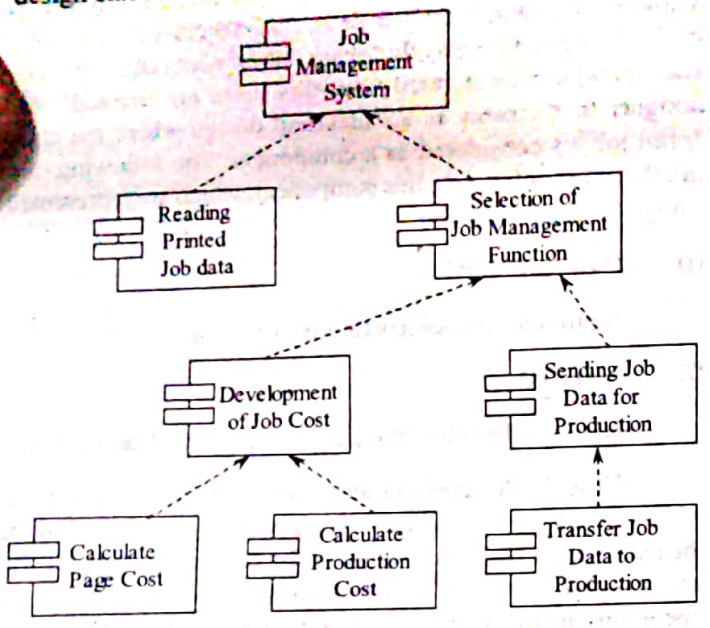


Figure: Structure Chart for a Conventional System

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.

Q44. 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

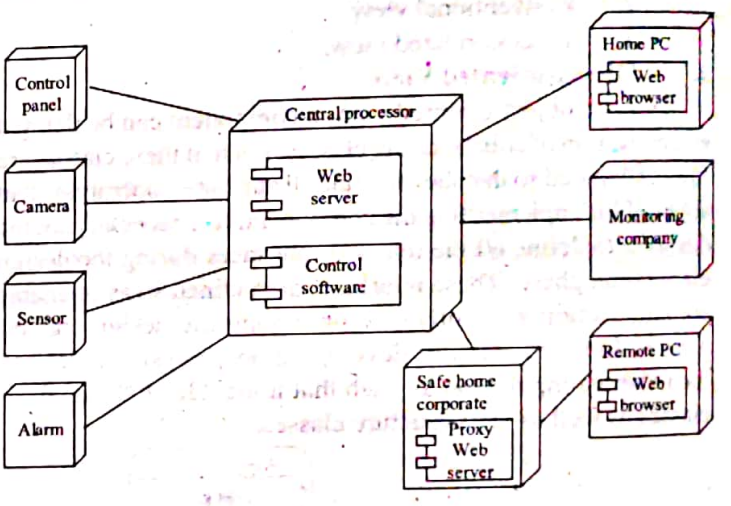


Figure: Deployment Diagram for Safe Home System

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.

3.2.5 Pattern-Based Software Design

Q45. Discuss about pattern based software design in detail.

Answer : Model Paper-II, Q14(b)

Design Pattern

A pattern in general refers to the particular layout which helps to perform certain tasks with ease. Technically, patterns can be considered as devices which help the programs in sharing the knowledge of their design.

The term design pattern is used in object-oriented terminology to perform the tasks such as defining the objects, class interfaces, inheritance hierarchies and factoring them into classes with relationships. Once all these steps are considered as a pattern they can be reused by applying them to several common problems.

A design pattern can be considered as 'reusable solution for commonly occurring problems in software design'. A design pattern is not a finished design, rather it is a template to solve the problems in many different situations.

The concept of design patterns was introduced by an architect named Christopher Alexander during 1970. Though, he mentioned the concept with respect to architectural patterns of buildings, it's also applicable to object-oriented design patterns.

The object-oriented design patterns represent relationships between classes and objects. These design patterns enable to reuse successful designs thereby avoiding to design them again and again. It also improves the speed of designer. The reusable object-oriented design pattern identifies classes and instances with their roles in much better way.

The most influential work on design pattern was done by Gamma, Helm, Johnson and Vlissides commonly known as Group of Four (GOF).

According to GOF, for the creation of a reusable object-oriented design, the design pattern names, abstracts as well as identifies the important aspects of a common design structure. The GOF design patterns can be described as communication among classes and objects which are customized for solving a design problem in a specified context.

Advantages of Design Patterns

The two major benefits are,

(i) Reusability of Solutions

By using proven solutions, we can solve software development issues which enables us to develop highly cohesive modules with minimal coupling.

(ii) Establishment of Common Terminology

- Enables efficient communication between designers.
- The pattern name is enough to solve particular issue.
- Improves code readability for coders.
- Reduces the time to find solutions by reusing tested and proven development paradigms.
- Introduces additional levels of indirection to achieve flexibility.
- Reduces the time for understanding the design.
- Promotes individual learning and team development.
- Readily tackle the changes as they are time tested solutions and hence makes software more modifiable.
- Enables creation of designs for complex problems that don't require large inheritance hierarchies.
- Helps to understand the basics of object-oriented design more easily and quickly.
- Motivates the junior team towards the senior team to learn the concepts of design pattern.
- Promotes communication among developers by providing a common language.
- Avoids those alternatives which avoid reusability thereby reducing variations.
- Improves design understandability and documentation.

Importance is given to codify the design patterns in particular domains by using domain specific patterns such as information, visualization, web design and user interface design patterns.

GOF has defined four essential elements of design pattern in the following way,

1. Pattern Name

It is a unique name which describes a design problem. Pattern name being concise and meaningful, improves communication among developers and helps to design at higher levels of abstraction.

2. Problem

It states the problem and its context such as how to represent algorithms as objects. It ensures when the pattern is applicable and what conditions should be met before the pattern is used.

3. Solution

It provides description of elements that make design pattern such as their responsibilities, relationships and collaborations. Pattern solution can't describe a particular implementation because patterns are like templates that are applicable to several similar situations. The solution gives a generic arrangement of classes and object to solve a problem.

4. **Consequences**

As the name implies these highlight the results, pros and cons of applying the pattern etc. It describes the impacts on system's portability, extensibility. They determine costs and evaluate design alternatives of pattern, language and implementation issues.

Design Pattern Template

Design pattern template is a consistent format proposed by GOF in order to understand the design pattern. In this format, each pattern is divided into sections according to a particular template. The template of the design pattern is as follows,

1. **Pattern Name and Classification**

All patterns must be associated with unique and descriptive name to be identified and referred. A good name is an advantage to design vocabulary.

2. **Intent**

It describes the purpose or goal of the design pattern with its reason. It is a short statement that specifies the pattern's rationale and the problem it addresses.

3. **Also known As**

This implies the association of other names to the pattern. It is an optional field.

4. **Motivation**

Here a scenario is considered that depict certain design problem and their solution using class and object structures. This will help to understand the pattern.

5. **Applicability**

It describes the situations when the design pattern is applicable to address the design problems and the context of the pattern.

6. **Structure**

It features graphical representation of pattern with class diagrams and interaction diagrams representing. The sequences of requests and collaborations between the objects. The graphical notation used are based on object modeling technique. It says, how the pattern solves the problem.

7. **Participant's**

It specifies the roles of classes, objects in design pattern.

8. **Collaborations**

It describes the way classes and objects interact to perform their roles.

9. **Consequences**

As the name implies, it describes the results, trade-offs and side effects of the design pattern.

10. **Implementation**

How the pattern can be implemented i.e., application of techniques and hints to implement the pattern. It also deals with language-specific issues.

11. **Sample Code**

Provides an illustration of how to use the pattern in programming languages such as C++, Java, Smalltalk etc.

12. **Known Uses**

Provides examples of pattern with its real usages.

13. **Related Patterns**

Relationship of the pattern with other patterns, their points of differences and how the pattern can be substituted with other patterns.

Q46. What type of design patterns are available for the software engineer?

Answer :

The different types of design patterns available for the software engineer are as follows,

1. Architectural patterns
2. Design patterns
3. Idioms.

1. Architectural Patterns

Architectural patterns are the design patterns that provides complete structural description about the software. The description specifies the following,

(i) Different relationships existing between the software components and subsystem.

(ii) Rules which defines relationship between the different architectural elements such as classes, packages, components and subsystems.

2. Design Patterns

Design pattern are the patterns that perform several tasks such as defining the objects, class interfaces, inheritance hierarchies and factors them into classes with relationships once all these steps are considered as a pattern they can be reused for solving commonly occuring problems in software design. A design pattern is not a finished design rather it is a template to solve the problems in many different situations.

3. Idioms

Idioms are the language-specific pattern, which imple-ments following,

(i) An algorithmic element of a component

(ii) An particular interface

(iii) A protocol

(iv) An approach to establish interaction between the components.

UNIT 4

CREATING AN ARCHITECTURAL DESIGN, MODELING COMPONENT LEVEL DESIGN, PERFORMING USER INTERFACE DESIGN

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. Differentiate between control and stamp coupling.

Answer :

	Control Coupling		Stamp Coupling
1.	Control coupling is said to occur when one module passes a variable to another module which control the internal logic of the other.	1.	Stamp coupling is said to occur when two modules pass data through a parameter which is a structure (or record).
2.	It is a moderate form of coupling.	2.	It is a loose form of coupling and simplifies interfaces between modules.
3.	The weakness of control coupling requires the calling program which receives the flags to have the knowledge of the internal logic.	3.	The weakness of stamp coupling is it encourages the creation of artificial data structures (i.e., binding un-related data elements into a structure).
4.	In control coupling, as rule of thumb one should use descriptive flags instead of control flags to describe the situation.	4.	In stamp coupling, as rule of thumb, one should pass a data structure containing required number of fields instead of passing a data structure with large number of fields.
	Example of control coupling is, print report		Example of stamp coupling is, calc_order_amt (what-to-print-flag) (i.e., the amount of the order to be returned).

Q2. Write short notes on architectural patterns.

Answer :

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.

Q3. Elaborate on "REP" in detail.

Answer : Model Paper-III, Q7

Release/Reuse Equivalency Principle (REP): The Release/Reuse Equivalency Principle(REP) suggests that the classes must be packed into the packages such that they should be convenient for other users to reuse. The packages consisting of the collection of classes must be tracked and released.

The author of the class/package must take some care about the packages that he/she develop. The author must notifies the reusers in advance about any changes in the packages that he/she wants to perform. The author has to maintain and support the older versions of the entities when the users are upgrading towards the newer version of the entities. This can be achieved by establishing a release contract system.

The REP states that the developer should not simply claim the reusability of the components but also provide the tracking system, notifications, support and safety to the potential reusers.

The REP not only focuses on reusability of the components but also it focuses on the reusers who use these components i.e., it should allow the same users to reuse all the classes of compatible component(package) but it should not allow a user to reuse the classes that are incompatible.

Q4. What is software component? Differentiate between hardware and software components.

Answer : Model Paper-I, Q7

Software Component: A software unit that is independent and that can be composed with other components in order to create software system is called a software component. It is a modular building block for computer software. These components are modular, deployable can be replaced and are able to encapsulate the implementation and expose the interfaces.

Difference between Software Components and Hardware Components

	Software Components		Hardware Components
1.	A software component consists of a group of instructions that tells the computer hardware how to perform a task.	1.	A hardware component is a physical device that are installed, connected to a power supply and interconnected to a power supply and interconnected with network cables.
2.	These components are custom built.	2.	These components are assembled according to the circuit design.
3.	The software component can run regardless of hardware.	3.	The hardware component cannot run without the corresponding software.
4.	It supports changing of requirements.	4.	Changing of requirement is cost-effective but it is not suitable to change.
5.	It cannot be replaced.	5.	It can be replaced if it wears out.
6.	Example: Interfaces, operating systems etc.	6.	Examples: Systems, I/O devices, people etc.

Q5. How should a user assess an architectural style that has been derived?

Answer : Model Paper-II, Q7

The organization and refinement of an architectural style can be determined using two ways,

(i) Control

(ii) Data.

The questions in these two ways provide insight into an architectural style.

Control

(i) How is the control inside an architecture managed?

(ii) Is there any existence of separate control hierarchy? If yes, then what functions are performed by the components of this control hierarchy?

(iii) How is the control transferred inside the system by using components?

(iv) How can a control be divided among the components.

(v) What is control topology?

(vi) Can control be synchronized?

(vii) Are the components operated asynchronously?

Data

(i) How is communication carried out between components?

(ii) How is the data flow carried out? Is it continuous or irregular?

(iii) Is the data transfer carried out from one component to the other component?

(iv) What is the role of data components?

(v) How does the data components interact (actively or passively) with other components?

(vi) How does control and data communicate inside the system?

Q6. Give few characteristics of good design.

Answer :

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.

Q7. 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.

Q8. Write about software documentation.

Answer :

Model Paper-I, Q8

Software documentation is the detailed information about the software provided by the developer of the software to its users. It includes documents like all the executable files, source code, manuals, design document, test document, software requirements specification, installation manual etc. These documents server an important role in understanding the software.

A good document reduces the effort of maintaining the software and helps the users to efficiently use their system. Incase of employee turnover, it (good document) helps the new employee to go through the documentation and easily adapt the software soon. It also helps the project manager to track the status of the project and accordingly produce and review the accomplished work.

Q9. What is user interface analysis?

Answer :

Model Paper-II, Q9

It refers to the process of examining the requirements of interface to develop a good user interface. It is a central principle of all the software engineering process models that before attempting to find the solution to a problem, it is better to analyze and understand it problem includes understanding,

(i) The end users

(ii) The tasks of end users

(iii) The content of the interface and

(vi) The environment in which the task will be performed.

Q10. What are the essential steps involved in implementing the interface design?

Answer :

Model Paper-II, Q8

Following are the steps essential while implementing the interface design,

- Initially, identify the objects and its associated operations. This is usually done by traversing through the use case description. Later, these are sufficiently elaborated and refined.
- Now, categorize these objects, under the following types i.e., source, application and target objects respectively.
- When all the possible objects are identified and their relative operations are defined, then start creating the screen layout.
- To make the layout attractive, several text, graphic images, icons etc., are placed at suitable locations. To make the layout to look more lively, any real world entity which is satisfying the situation (while making the layout) can be added.

PART-B

ESSAY QUESTIONS WITH ANSWERS

4.1 CREATING AN ARCHITECTURAL DESIGN

4.1.1 Software Architecture

Q11. What is software architecture? Why is it important?

Answer :

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.

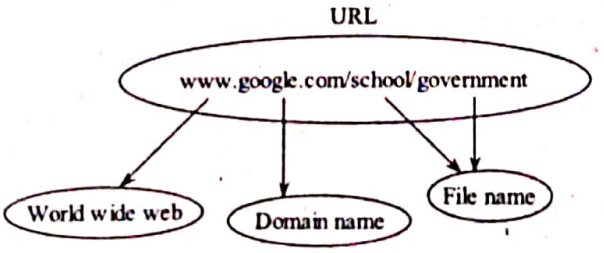


Figure: Sample of Software Architecture

Q12. Discuss why software architecture plays an important role during development and discuss various architectural terminology.

Answer : Model Paper-I, Q15(a)

Software Architecture

Software architecture plays crucial role during the development of software.

For remaining answer refer Unit-IV, Q11, 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.

Q13. 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 :

Software Architecture and its Importance

For answer refer Unit-IV, Q12.

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.

4.1.2 Data Design

Q14. What is meant by data design? Explain with an example.

Model Paper-II, Q15(a)

Answer :

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-IV, Q15.

Q15. 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.

4.1.3 Architectural Styles and Patterns

Q16. What is architectural style? Discuss various categories of it.

Answer : Model Paper-II, Q15(b)

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.

Various Categories of Architectural Styles

(a) Data Centered Architecture

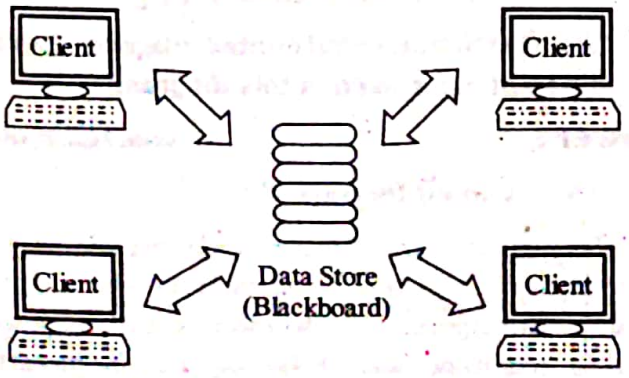


Figure: Data Centered Architecture

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.

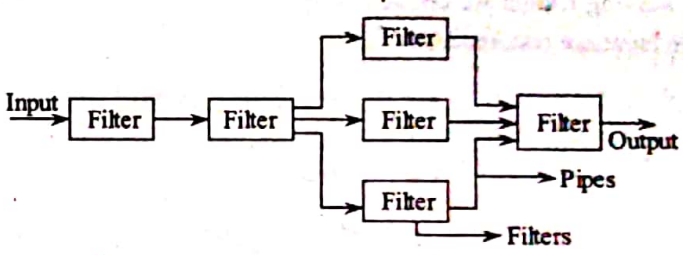


Figure: Data Flow Architecture

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.

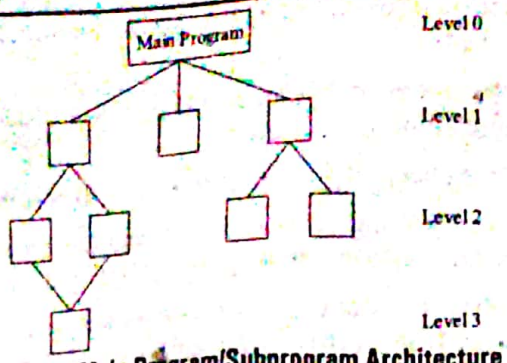


Figure: Main Program/Subprogram Architecture

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.

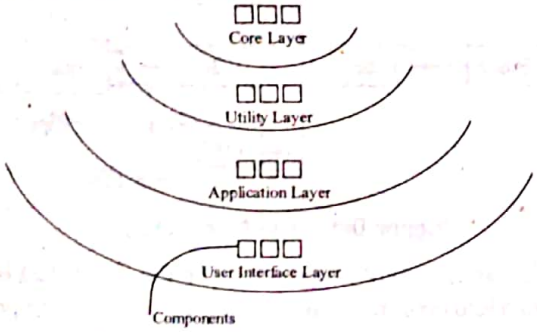


Figure: Layered Architecture

Q17. 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.

4.1.4 Architectural Design

Q18. Draw the architectural context diagram. Explain different parts used in this diagram.

Answer : Model Paper-III, Q14(b)

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.

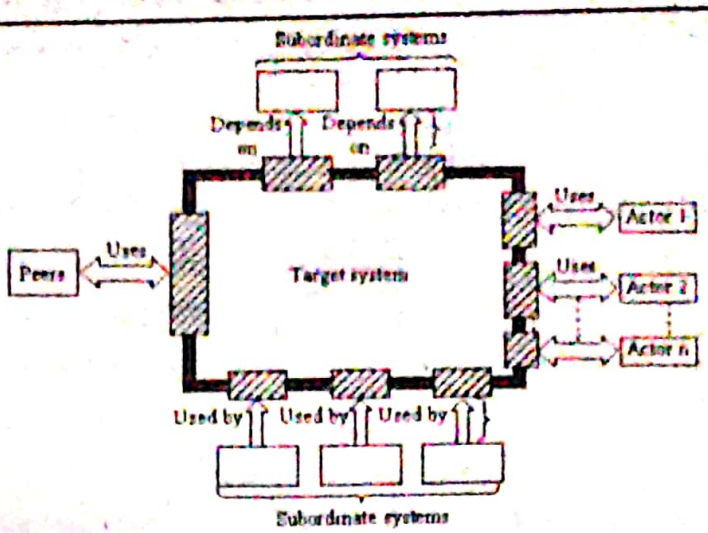


Figure: Architectural Context Diagram

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).

Q19. 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.

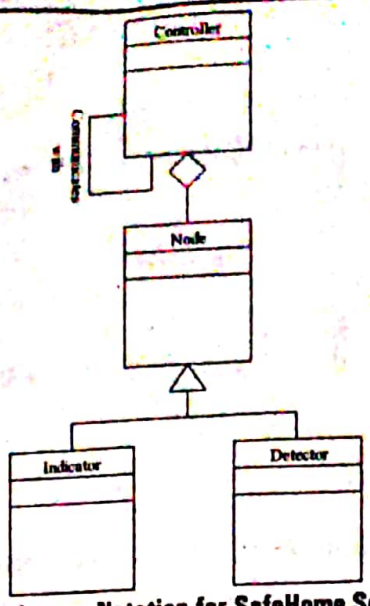


Figure: UML Archetype Notation for SafeHome Security System

As the above represented archetypes represents only abstractions, they can be refined into further components by refining these abstractions.

Q20. 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.

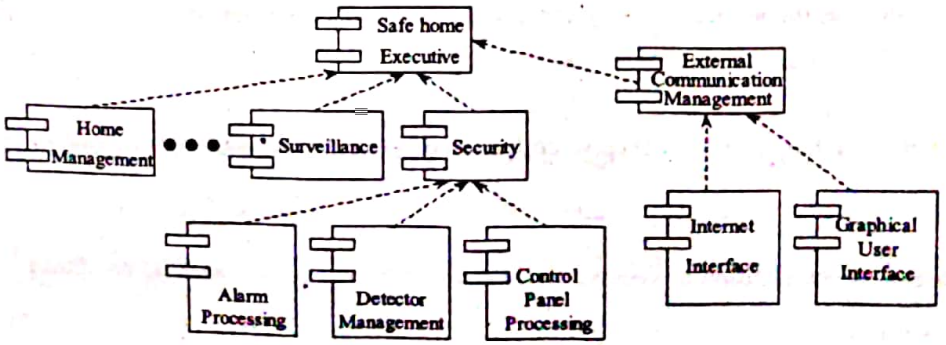


Figure: Refining of Architecture into Components

Q21. What is instantiation? How would you describe instantiations of the systems?

Answer :

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.

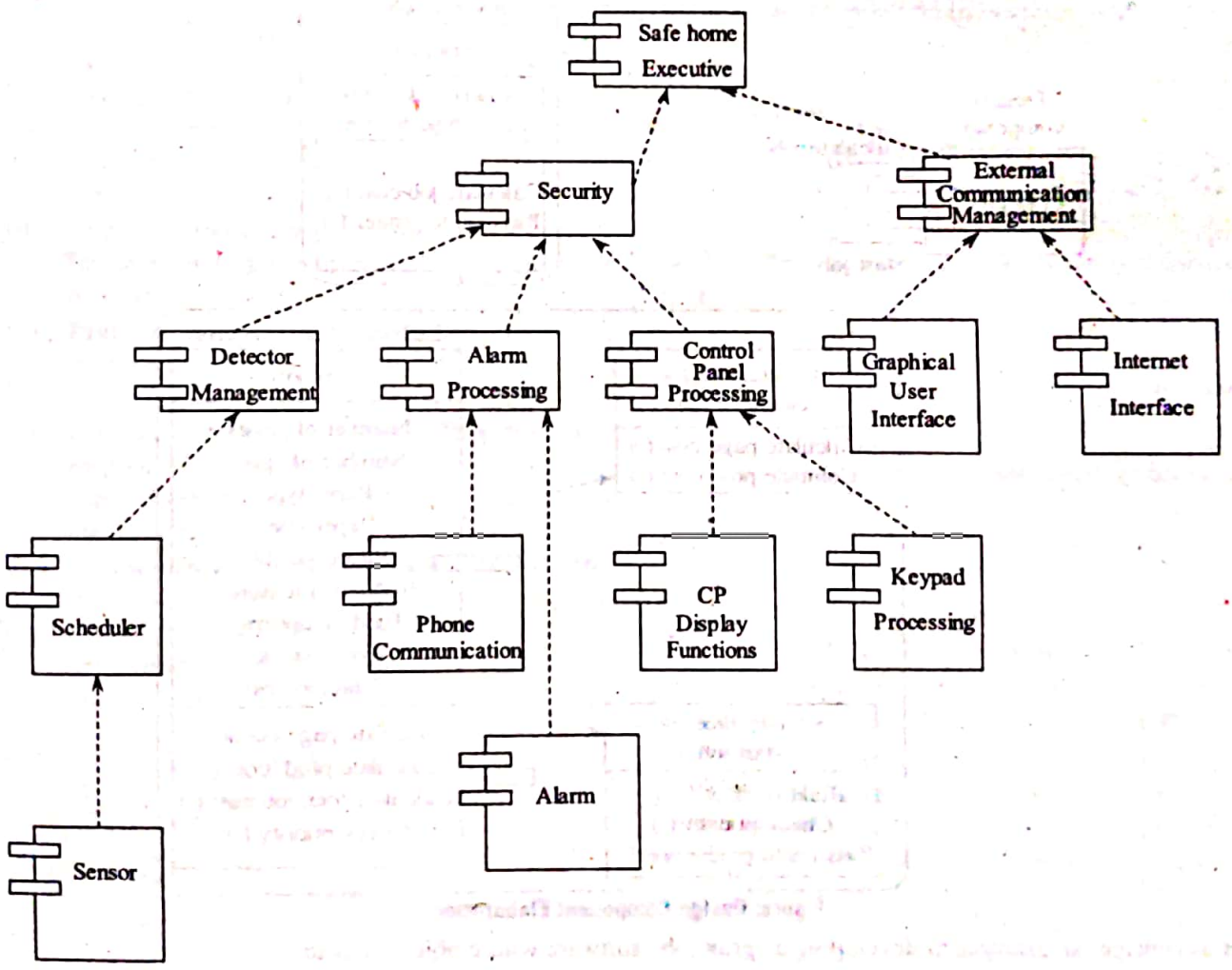


Figure: Instantiation Process

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.

4.2 MODELING COMPONENT-LEVEL DESIGN

4.2.1 Definition of Component

Q22. Explain the component from traditional view with an example.

Model Paper-II, Q18(a)

Answer :

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.

1. **Object-oriented View:** 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.

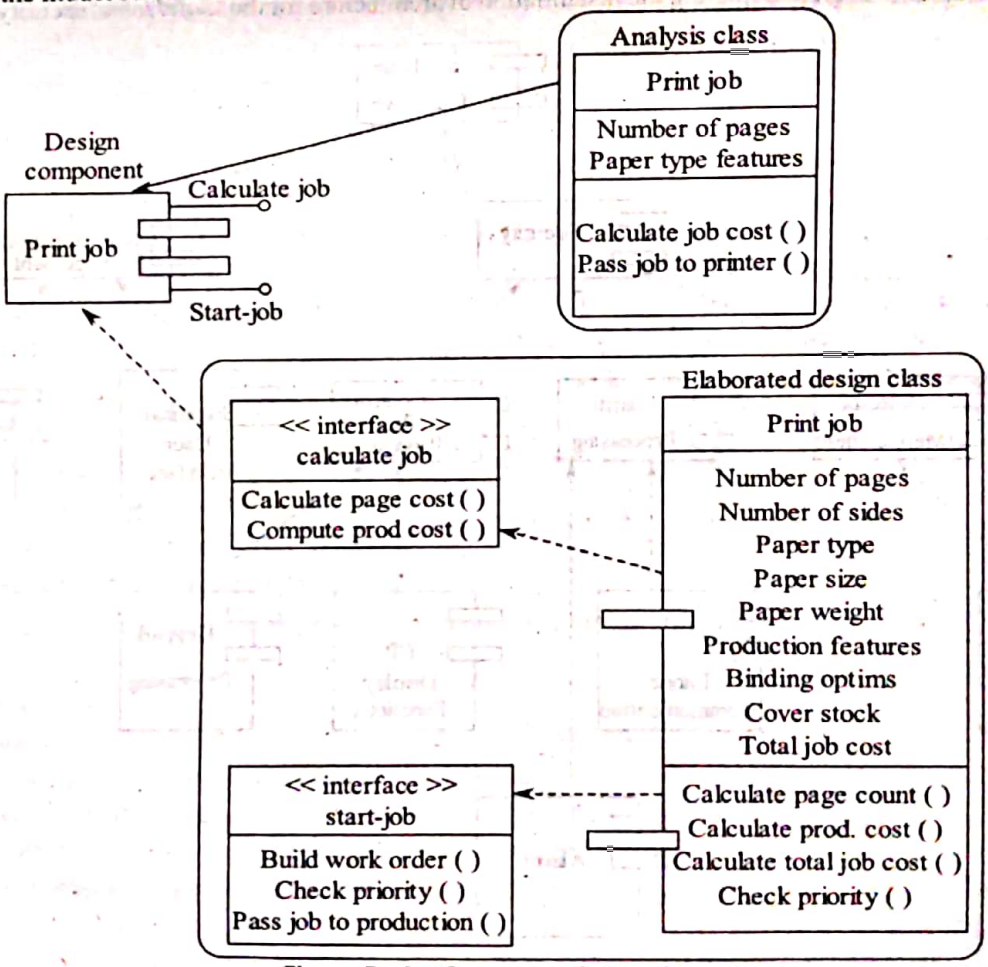


Figure: Design Component Elaboration

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.

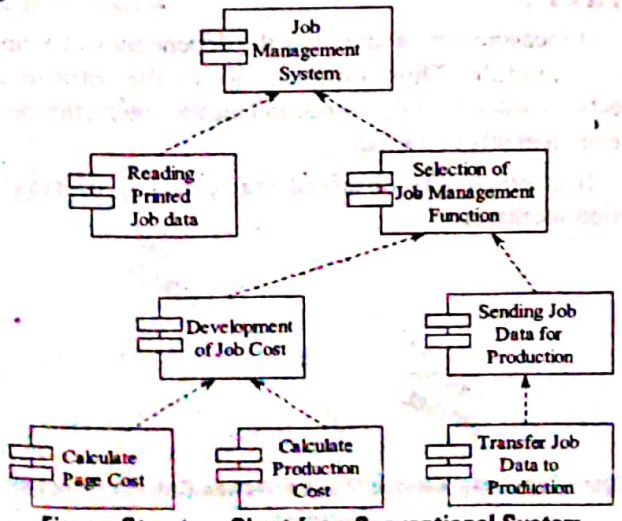


Figure: Structure Chart for a Conventional System

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 modulers 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.

4.2.2 Designing Class – Based Components

Q23. Explain the principles of class-based component design in detail.

Answer : Model Paper-III, Q14(a)

Design Principles

There are basically seven design principles which are applicable to component level-design. They are,

1. Open-Closed Principle (OCP)0
2. Liskov Substitution Principle (LSP)
3. Dependency Inversion Principle (DIP)
4. Interface Seggregation Principle (ISP)
5. Reuse Release Equivalency Principle (RREP)
6. Common Closure Principle (CCP)
7. Common Reuse Principle (CRP).

1. **Open-closed Principle (OCP):** Open-closed principle states that "a module should be opened for extension but closed for modification". Despite being contradictory, this principle is considered as one of the most essential characteristics of a good component level design. According to this principle, the designer must design the component in such a way that even if it is extended there are no internal modifications performed to the components. In other words, it can be said that the component must be capable of being extended but there must not be any requirement of carrying out internal modifications. The designer achieves this by creating abstraction that acts as a buffer between the functionality and the design class.

2. **Liskov Substitution Principle (LSP):** If a component works properly when some base class object is provided to it, then according to LSP, it should continue to work properly even if it is given a subclass object of that base class. In other words, subclass objects should be substitutable in place of base class objects.

3. **Dependency Inversion Principle (DIP):** The higher level modules and lower level modules must be abstraction dependent i.e., the modules of higher level should not rely on the modules of lower level. The abstraction should be detail independent but detail must be abstraction dependent.

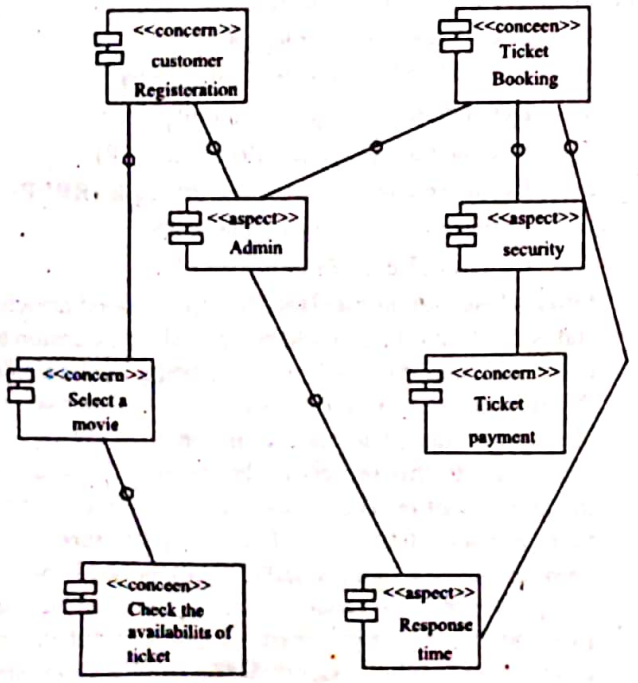
4. **Interface Seggregation Principle (ISP):** The dependency relationship that exists between the classes must rely on the smallest possible interface. The designer should also provide a general-purpose interface rather than many client specific interfaces.

5. **Reuse Release Equivalence Principle (RREP):** The reuse granule is nothing but the release granule. The components released by a monitoring system are the one's that have the capability of being reused effectively.

6. **Common Closure Principle (CCP):** The property of common closure can be shared by those classes that are internal to the released component i.e., changes made to one class result in changing all the other classes.

7. **Common Reuse Principle (CRP):** The property of common reuse can be applied to the classes that are defined within a package i.e., if a class defined in a package is reused then all the other classes within the same package must also be reused.

Online Movie Ticket Management System



Figure

Q24. List the different component level design guidelines.

Answer :

Component level design can be efficiently performed if the designer follow certain guidelines which are applicable to components, their interfaces and the dependencies and inheritance.

Guidelines Applicable to Components

1. The components must be provided with their naming conventions, which are specified while designing and later refined as well as elaborated during the design of component level model.
2. The component names in architectural model must be selected from the problem domain.
3. The component names must have a meaning so that they can be easily understood by the stake holders.

Guidelines Applicable to Interfaces: Interfaces plays a vital role, as they provide the information regarding communication and collaboration of different component. The guidelines stated by ambler include,

1. If an interface is represented using a lollipop approach then the approach must be used in conjunction with formal UML box and dashed arrow approach.
2. If an interface needs to be consistent, then the flow of interface must be from the left hand side of the component box.
3. If an interface is relevant to a component being designed then only that interface needs to be displayed irrespective of other available interfaces.

Guidelines Applicable to Dependencies and Inheritance

1. The dependencies must be moduled from left to right and the inheritance must be moduled from bottom to top in order to achieve better readability.
2. The dependencies between the components must be represented using interfaces.

Q25. Explain in detail the concept of cohesion.

Answer : Model Paper-III, Q15(a)

A measure which identifies the dependency of elements within a module. Thus, coupling gives the intra-module connections. Intra-module connection means, interaction needed between elements of a module.

It is practically observed that coupling decreases as cohesion increases.

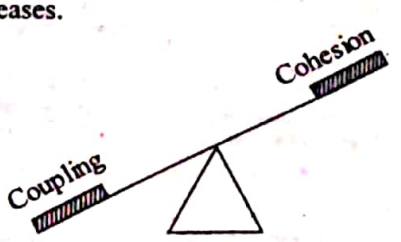


Figure: Inversely Related Modularization Criteria (Coupling, Cohesion)

This, does not mean that coupling becomes 'zero' or 'nill' for the highest cohesion

Levels of Cohesion: The amount of cohesion between elements gives rise to the levels of cohesion. There are seven levels of cohesion.

1. Functionally cohesive
2. Sequentially cohesive
3. Communicationally cohesive
4. Procedurally cohesive
5. Temporally cohesive
6. Logically cohesive
7. Coincidentally cohesive.

1. **Functionally Cohesive:** Two elements are said to be functionally cohesive if they perform the same type of function. Here, function is not necessarily a mathematical function instead it represents a single goal. For example, "sorting of an array" and "computing square root" are functionally cohesive modules. This is the strongest form of cohesion that exist between the modules.

Test for Functionally Cohesive Modules: To identify whether a module is functionally cohesive or not "write a sentence describing the purpose of the module". This sentence should describe the module completed accurately no matter how long that it is. Then, try to match the written sentence with the following types of sentences.

Type1: "If the sentence has a comma or contains more than one verb", then the module is definitely performing more than one task (function). Hence, the module is not functionally cohesive, but is either sequentially or communicationally cohesive.

Type2: If the sentence uses words like 'when', 'first', 'last', 'before', 'after', etc.,", which are actually related to time, then the module is not functionally cohesive but is either sequentially or temporally cohesive.

Type3: If the predicate of the sentence does not have even a single object after the verb, then the module is logically cohesive.

Type4: If the sentence has words like "initialization and cleaning". The module is not functionally cohesive but is temporally cohesive.

If the written sentence matches (in terms of the syntax of the sentence). Syntactically with any of the above types, then the module is not functionally cohesive. As we said that, if a comma comes in a sentence (that means a sentence is a compound sentence) then the module is not functionally cohesive. This is not completely true, it means that it is "likely" that the module is not functionally cohesive.

2. **Sequentially Cohesive:** Elements in a module are sequentially cohesive when output of one element acts as an input to another element. Sequential cohesion do not give any rules to combine these type of elements in modules. The elements that are sequentially cohesive can be combined in a single module, or divided into a number of parts etc. Thus, a sequentially cohesive module as a main system which is to be designed.

3. **Communicationally Cohesive:** Two elements are said to be communicationally cohesive if they need the same input or output data. An example for a communicationally cohesive module is "printing and punching of record". Thus, if the elements in a modules are communicationally cohesive then they can't be functionally cohesive.

4. **Procedurally Cohesive:** Elements that belong to the same procedural unit are said to be procedurally cohesive. Procedural cohesion occurs because of the division of a module into sub-modules. For example., the sub-module can be a loop or a block of conditional statements. If the structure of a module is defined by a flow chart then the elements in a module are related by procedural cohesion.

5. **Temporally Cohesive:** If two elements are related in time and are executed together, then the elements are temporally cohesive. For example, a module may perform initialization, termination and clean-up of all elements. Execution of the elements is done at almost the same time.

6. **Logically Cohesive:** Elements that perform the same logical function are said to be logically cohesive. For example, a module may perform all inputs or all outputs i.e., to input or output a particular record of information, a status flag is used. The flag is used to indicate the statements to be executed in the module. Using "hybrid the logical cohesion information flow". form of coupling of a module is made more complicated. Thus, logical cohesion should be avoided to the possible extent.

Elements that are temporally cohesive are also logically cohesive. But, temporal cohesion overrules logical cohesion because temporal cohesion do not use status flags.

7. **Coincidentally Cohesive:** When decomposing a program into several pieces of code called modules, it is possible that the elements of one module is put into another module for any reason. If the elements do not have any kind of relationship, then the elements are said to be coincidentally cohesive.

For example, a module is decomposed from another module to store the duplicate code, the elements that comprises this module are not related to each other. Thus, modification of such a module will result in "incorrect" functioning and also cause unwanted modifications to elements. Hence, decomposition should be done carefully to avoid such situations.

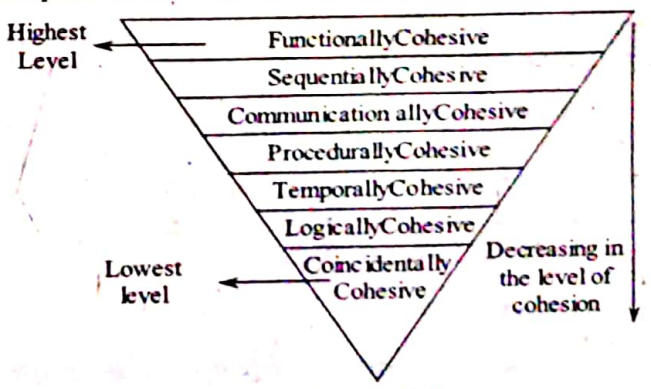


Figure: Levels of Cohesion

Functional cohesion is the strongest of all cohesions and coincidental is the lowest. Functional cohesion gives an exact resemblance to the main system. But, the most commonly acceptable cohesion level is communicational.

Q26. Explain about layer cohesion in detail.

Answer :

Layer Cohesion

To achieve layer cohesion, the resources responsible for providing related services to the user or to higher-level layers must be maintained together excluding other unrelative services. A hierarchical layered structure forms a proper layer cohesion lower layers are not applicable to access higher layers but higher layer can access services provided by lower layers. This can be show from the example below,

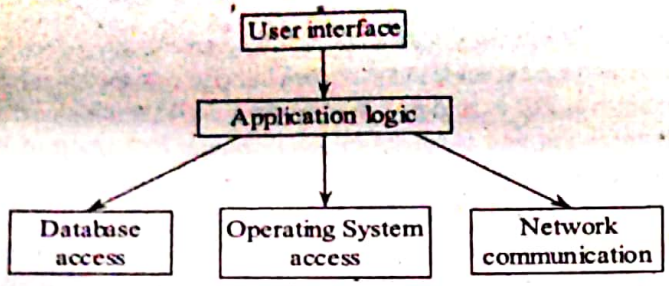


Figure: Layer in Application Program

The related services that can form a layer are as follows,

1. Data storage services
2. Computational services
3. Data transmission services
4. Security management services
5. User interaction services
6. Access of operating system services
7. Hardware interaction services.

Example: To interface a system with a sound card, it is necessary to create a module that can interact with that card. The module should only interact with this card rather than doing other additional services. Moreover, this module should comprise the code that accesses the card directly.

The layer provides its services with the help of set of procedures or methods known as Application Programming Interface(API). The specification of API must define the semantics of services. The protocols for higher-level layers to access the API and its side-effects.

Benefits of Layer Cohesion

1. The top-level layers can be replaced without affecting the lower-level layers.
2. Lower-level layers can be replaced by its equivalent lower-level layer. However, this requires the aspects of API to be replicated, so as to allow the top layers to continue their work in the same way.

Q27. What is coupling? Explain various types of coupling in detail.

Answer :

Coupling

Coupling is defined as a qualitative measure that identifies the dependency of which one module have on another module. A module is said to be tightly coupled, if they are strongly connected (highly dependent) and are said to be loosely coupled, if they are loosely connected (less dependent). On the other hand, if the modules are independent of one another then no coupling exist between these modules.

Types of Coupling

The different categories of coupling include,

1. **Content Coupling:** Content coupling is said to occur when one component secretly modifies the internal data of another component.
2. **Common Coupling:** Common coupling is said to occur when two or more components uses a global variable. This sort of coupling results in many errors, which are sometimes difficult to be corrected.
3. **Control Coupling:** Control coupling is said to occur when one operation OP1( ) invokes another operation OP2( ), such that, the invocation results in passing the control flag to OP2( ). This control flag then directs logical flow within the operation OP2( ). The control coupling results in changing the meaning of control flag if certain unrelated changes are made in OP2( ). If these changes are bypassed, then errors may be generated.
4. **Stamp Coupling:** Stamp coupling is said to occur when an operation of a class (i.e., class 1) consists another class (i.e., class 2) as its argument. In this type of coupling, modification of the system becomes highly difficult as class 2 is part of class 1's definition.
5. **Data Coupling:** Data coupling is said to occur when operations communicate with each other via long strings of data arguments. This type of coupling requires huge amount of bandwidth for enabling communication between the classes and components, which inturn results in increasing the complexity for the interfaces.
6. **Routine Call Coupling:** Routine call coupling is said to occur when one operation is invoked by another operation.
7. **Type Use Coupling:** Type use coupling is said to occur when one component uses a data type which is defined in another component. If the data type defined in a component changes, then all the components which are using this data type must also change.
8. **Inclusion or Import Coupling:** Inclusion or import coupling is said to occur when one component imports the package or the content of another component.
9. **External Coupling:** External coupling is said to occur when one component communicates with other infrastructure component, such as operating system functions, database capability, telecommunication functions.

4.2.3 Conducting Component – Level Design

Q28. Explain the steps involved in conducting component-level design.

Answer :

Steps in Conducting Component-level Design

The following steps are conducted for component-level design when it is applied for an object oriented system,

Step 1: Identifying the Design Classes that are Relative to the Problem Domain Component

In this step, every analysis class and architectural component that corresponds to problem domain is elaborated using analysis and architectural models respectively. This elaboration is done in a similar way performed in object-oriented view.

Step 2: Identifying All Design Classes that are Relative to the Infrastructure Domain: In this step, all the classes that correspond to the infrastructure domain such as GUI components operating system components, object and data management components are identified.

Step 3: Elaborating the Design Class that are not Reusable Components: In this step, all the interfaces, attributes, operations that are required for implementing a class are elaborated. It is necessary to consider the design heuristics during this elaboration. Such elaboration can be done by,

(i) Providing message details during collaboration of classes or components.

(ii) Identifying the relevant interfaces to the components

(iii) Elaborating attributes and defining datatypes and data structures.

(iv) Describing the flow of processing logic.

(i) **Providing Message Details during Collaboration of Classes or Components:** In this sub-step, the procedure of collaboration between the analysis classes is described by making use of collaboration diagram. The details related to collaborations is provided by specifying the structure of the message, which is being passed among the objects that are internal to a system. After the completion of component-level design, the messages are elaborated by expanding its syntax in the following manner,

[guard condition] sequence expression (return value): = name-of-message (list-of-arguments)

Here, "guard condition", denotes the conditions that are to be satisfied prior to sending a message. "Sequence expression", denotes an integer value that specifies the order in which a message is sent. When a message executes an operation "name-of-message" denotes the operation that is to be executed.

"List-of-arguments" denotes the attributes that are defined in an operation.

(ii) **Identifying the Relevant Interfaces for the Component:** In this sub-step, all the interfaces that are applicable to the components are identified. An interface is defined as a set of externally visible operations that does not have any internal structure, attributes, relationships. In a formal way, an interface is considered similar to an abstract class, which is responsible for interconnecting the design classes in a controlled manner. The operations that are associated with a design class can be categorized into numerous abstract classes. Every individual operation which is defined in an interface must perform processing that invokes only a single function or sub-function their by making the interface cohesive.

(iii) **Elaborating Attributes and Defining Data Types, Data Structures for Implementation:** In this sub-step, all the attributes, datatypes and internal data structures that are required for implementing the components are elaborated. The datatypes and data structure that provides the description of attributes are defined using programming language which is used for carrying out implementation. The data-type of an attribute is defined using UML notation in the following manner,

att-name : data-type-exp = initial-val {property-string}

In the above syntax, 'att-name' specifies the name of the attribute, 'data-type-exp' specifies the datatype for the attribute, 'initial-val' specifies the value assigned to an attribute when an object is created, 'property-string' specifies the characteristics of a defined attribute.

(iv) **Describing the Flow of Processing Within Each Operation:** In this substep, the flow of processing is described using any one of the following approach. (i) UML activity diagram (ii) programming language based pseudoca. The elaboration of every software component is done iterately. Initially in the first iteration, every operation is defined as part of design class and in the next iteration, the operation is elaborated. In every iteration, it should be ensured that there exist high cohesion in the operation so that it perform processing which focuses on a limited function or sub-function.

Step 4: Describing Persistent Data Sources and Identifying the Classes for Managing: In this step, the persistent data sources such as databases and files are described. Basically, these data sources are defined as a part of the architectural design, but when the design is elaborated it is necessary to provide the additional information about their structure as well as their organization.

Step 5: Developing and Elaborating the Behavioural Representations for a Class or Component: In this step, the behavioural representation of components and design class is developed and elaborated at both architectural and component level. In architectural level UML state diagrams are used in order to represent the behavior of analysis classes whereas in component level the state diagram are used for representing the behavior of design class. There are two factors that affects the dynamic behavior of an object,

(i) Event

(ii) Current state.

It is necessary for the designer to analyze all the use cases (relevant to the design classes) which provides the information that assists the designer in understanding the events affecting the object and the states. The transition from one state to another state is represented using the following syntax.

Event-name [param-list][guard-condition]/action-exp where "event-name" specifies name of event, "param-list" specifies data regarding the event, "guard-condition" specifies a condition which must be satisfied prior to the triggering of event and "action-exp" specifies the action to be performed when transition occurs.

Step 6: Elaborating Deployment Diagrams for Additional Implementation: In this step, the deployment diagrams are elaborated in component-level design so as to provide additional information about the location of key packages that are applicable to the components. Generally, each component is not represented in the component diagram because of diagrammatic complexity. However, there are certain situations wherein deployment diagram are elaborated so that the diagram provides the information about the operating system environment and location of component packages.

Step 7: Factoring every Component Level Design Representation and Considering Alternatives: In this step, the component design is refactored at every iteration so as to create complete consistent and accurate component -level model.

However, while refactoring the designer must not encounter the issue of tunnel vision. Therefore, it is necessary to provide the designer with all the alternative design solutions so that, the designer may consider these solutions prior to the development of final design model.

4.2.4 Object Constraint Language

Q29. Explain briefly about object constraint language.

Answer :

Object Constraint Language (OCL)

OCL is an expression language for UML that provides description about the constraints specified on object oriented language and other modelling artifacts. The use of OCL enhances (or supplements) UML by enabling a software engineer to utilize a formal grammer and syntax while constructing unambiguous statements regarding the different design components. The construction of an OCL language statement involves the following four parts,

1. **Context**

   It define certain situation that consists of valid OCL statement.

2. **Property**

   It represents certain characteristics regarding the context.

3. **Operation**

   It performs the actions of manipulating a property. Some of the operations include arithmetic, set oriented.

4. **Key Word**

   It specifies conditional expressions. Some of the keywords include if, then, else, and, or, not, implies.

Example

Consider the following guard condition "An authorization occurs only if a particular customer is authorized to approve the cost of the job". The syntactical representation of this expression in OCL is,

```
Cust
    self.check_authorization = 'True'.
```

Here, the self.check_authorization is a boolean attribute that belongs to class 'Cust'. The value of the attribute must be set to 'true' when the guard condition is to be satisfied.

After the creation of design model, it is necessary to specify pre or post conditions that are to be satisfied before executing an action specified by the design. These conditions are specified using powerful tools of OCL. In addition to these conditions, another condition called invariant condition is to be specified which must be satisfied before the pre-condition and after the post condition.

Example

Consider the following guard condition.

A customer provides the "max_cost_bound" for print-job and "delivery_data" when the characteristics of other print-jobs are specified. Here, if the cost and delivery date is above the specified bound, then the job is not submitted and a notification is send to the customer. The syntactical representation of this condition in OCL is,

```
Context  Print_job : : Check (max_cost_bound : integer,
cust_delivery_date : integer)
Pre : max_cost_bound > 0 and cust delivery_date < 0
and self_job authorization = "false"
Post : if self.total_job_cost < = max_cost_bound and
self.delivery_date < = cust delivery date
then
    self.jobauthorization = "True"
end if.
```

Here, the pre-condition denotes that the cost and delivery date must be given by the customer and the authorization must be set to value "false". Once the cost and delivery bounds are known, the post condition is executed. The expression self. jobauthorization is initially set to value "false" and later to "true" when the operation completes its execution.

4.2.5 Designing Conventional Component

Q30. Explain in detail the following design notations.

(a) Graphical design notation

(b) Tabular design notation.

Answer : Model Paper-II, Q16(b)

(a) Graphical Design Notation

Graphical Design Notation uses flow charts for designing the components. The purpose of using flow chart is that they depict the procedural details in a simple pictorial representation.

A flow chart uses the following boxes of various shapes to represent different operations,

(i) **An Oval Shaped Box (⬭)**

It denotes the beginning of a process

(ii) **Rectangular Box (▭)**

It denotes a process to be carried out

(iii) **Diamond Box (◇)**

It denotes a decision to be made on logical conditions.

In addition to these boxes, a flow chart uses an arrow (→), to denote the direction of flow in the process.

Flow Chart Constructs

The different structure constructs can be represented using flow charts are,

1. Sequence
2. Condition (if_then_else)
3. Selection
4. Repetition.

1. Sequence Construct

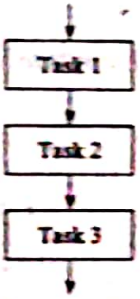


Figure: Sequence Construct

A sequence construct is denoted by processing rectangular boxes connected together by an arrow that shows the flow control.

2. Condition Construct

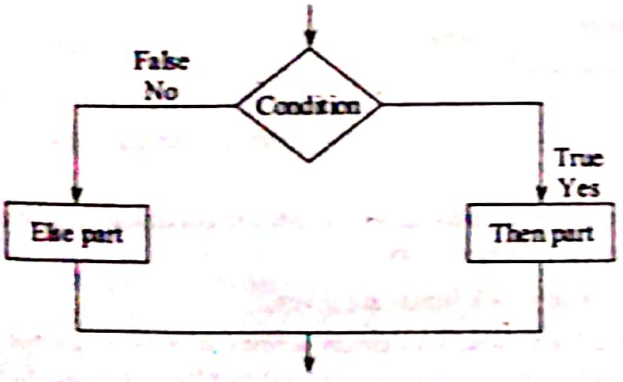


Figure: Condition Construct

A condition construct also known as if_then_else is denoted by a decision diamond box that contains the logical condition. If the condition is true, then the logic in the "then part" gets executed and if the condition evaluates to false then the "else part" gets executed.

3. Selection Construct

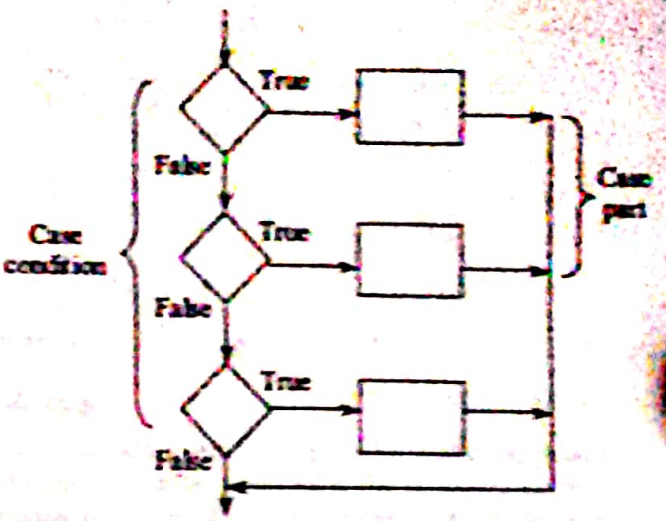


Figure: Selection Construct

A selection construct is an extension of condition construct, where a given value is evaluated until a true condition is obtained. And upon the occurrence of the true condition the "case part" gets executed.

4. Repetition Construct

A repetition construct can be represented in two different forms,

(i) Do while

(ii) Repeat until.

(i) Do-while

A do-while repetition construct evaluates a condition and executes the loop consecutively until the condition becomes false.

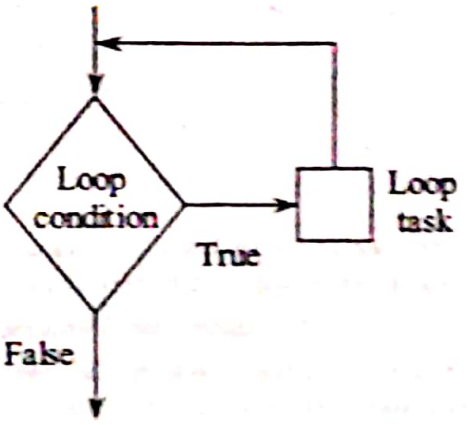


Figure: Do-while Condition

(ii) Repeat-until (while)

A repeat until construct initially process the task in a loop and then evaluates the condition. This evaluation is done repeatedly until the condition holds true.

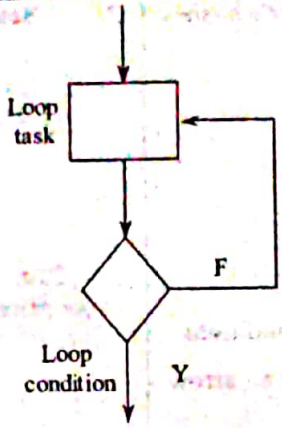


Figure: Repeat-until (while)

The disadvantage of using these structured constructs is that they might result in inefficiency when there is a need of eliminating the set of nested loops. Moreover, such elimination in turn increases the probability of errors and may result in having negative impact on readability and maintainability.

Therefore, to overcome this drawback the designers can use any one of the following alternative,

(i) Redesign the procedural representation, so as to eliminate the need of "escape branch" in nested loop.

(ii) Violate the structured constructs in a controlled manner.

(b) Tabular Design Notation

The tabular design notation uses decision tables in designing the procedures to test complicated pairs of conditions and take appropriate action depending on those conditions.

A decision table is a tabular form that consists of a set of conditions and their respective actions. A decision table consists of four parts. They are as follows,

1. Condition/cause stub
2. Condition/cause entry
3. Action/effect stub
4. Action/effect entry.

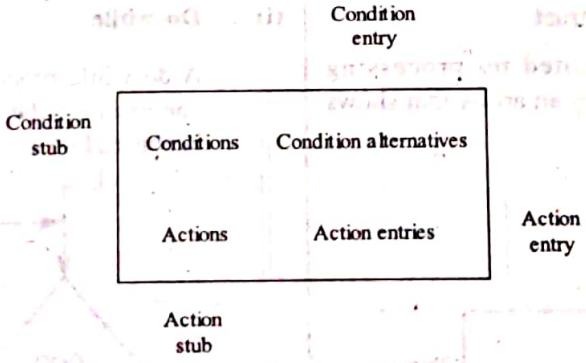


Table: Decision Table

Each condition corresponds to either a variable or predicate whose possible truth-values are listed in condition alternatives. Each action is a procedure that is to be executed based on the values present in action entry.

Condition stub and condition entry forms the upper half of the table in which inputs are given.

Action entry and action stub forms the lower half of the decision table in which necessary actions to be taken are not decided. Every column represents a test case (rule) which is the combination of causes and its corresponding effects are specified in action stub. Limited-entry decision table is the simplest of all decision tables because here the condition alternatives are simple boolean values which are either Yes/No or True/False. The fulfillment of each condition and action is noted with either "Yes1" or "No". It is mandatory that there should be a truth value for every condition and its action.

Condition Stub

It consists of a list of causes. For a rule to be satisfied, conditions are marked with Yes/No. "Yes" indicates that the condition is satisfied and "No" specifies that the condition is not satisfied. The variable I specifies the immaterial test case which means that a particular condition does not have any role in that rule.

Action Stub

It describes the possible actions that are to be executed when a particular rule is satisfied. "Yes" in action entry specifies that an action needs to be executed and "No" specifies that an action need not be executed.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Condition 1	Yes	No	Yes	No	No	Yes
Condition 2	No	Yes	I	Yes	No	I
Condition 3	I	Yes	No	I	No	Yes
Condition 4	Yes	I	I	No	Yes	No
Action 1	Yes	No	Yes	No	No	Yes
Action 2	Yes	Yes	No	Yes	No	No
Action 3	No	Yes	No	Yes	No	Yes
Action 4	Yes	No	Yes	No	Yes	Yes
Action 5	No	Yes	No	Yes	No	No

Table: Action Stub

The above decision table is converted as follows,

1. Action 1 is performed if Conditions 1, 4 are true (Yes), in Rule 1 and if Condition 2 is false (No), if Condition 1 is true in Rule 3 or if Conditions 1, 3 are true in Rule 6.
2. Action 2 is performed if Conditions 1, 4 are true under Rule 1, if Conditions 2, 3 are true under Rule 2 or if Condition 2 is true under Rule 4.
3. Action 3 is performed if Conditions 2, 3 are true under Rule 2, if Condition 2 is true under Rule 4 or if Conditions, 1, 3 are true under Rule 6.
4. Action 4 is performed if Conditions 1, 4 are true under Rule 1, if Condition 1 is true under Rule 3, if Condition 4 is true under Rule 5 or if Condition 1, 3 are true under Rule 6.
5. Action 5 is performed if Conditions 2, 3 are true under Rule 2 and if Condition 2 is true under Rule 4.

Steps for Building a Decision Table

The following steps are performed while developing a decision table,

Step1

All the actions that are related to a particular module are listed.

Step2

All the conditions for the processing of the module are listed.

Step3

Ignoring unobtainable pairs of conditions the remaining set of conditions are associated with their corresponding actions.

Step4

The rules specifying the action to be performed when a set of conditions are met is defined.

Q31. Define PDL. What is the advantage of using PDL? Explain with suitable example how it helps in expressing the design in different levels of detail.

Answer :

PDL

PDL can be defined as a a language with the complete external syntax of a structured programming language and a vocabulary of a natural language, english. Thus, PDL may be regarded as "structured english". The automatic processing can also be done on the design, due to formal structural constructs. The difference between PDL and programming language is the way of using a narrative text, which is embedded explicitly within the PDL statements. Since it is not possible to compile the PDL, certain tools are used that are responsible for translating the PDL into a programming language. In addition to this, the tools are capable of generating information related to nesting maps, cross-reference table and different types of other information.

The syntax of PDL consists of constructs for the following,

(i) Component definition construct

(ii) Repetition construct

(iii) I/O construct

(iv) Condition construct

(v) Data declaration

(vi) Interface description.

It is possible to expand the basic syntax of PDL by including keywords associated with multitasking, interrupt handling, interprocess synchronization.

Advantages of PDL

The advantages of PDL are as follows,

1. PDL is not a programming language due to which the designer can freely use PDL without worrying about syntactical errors.
2. PDL allows detailed explanation of the design, that is appropriate for the problem.
3. PDL provides a successive refinement technique by initially generating a rough outline of the entire solution at a given stage of detail and upgrading the design only when the design is agreed on at this stage.
4. PDL saves cost by finding the design errors early during the design phase.
5. PDL develops error-free designs by providing design verification by phases.
6. The designer can express a design at different levels as per requirement.

Example

Consider an alarm management component. Using a PDL, the following rough outline of the code is obtained.

Component AlarmManagement;

The alarm management component manages the switches on the control panel and inputs from sensors by type. Moreover, it acts on all alarm conditions.

```
Set the data items and system status to default values.
Initialize the system ports and reset the hardware.
Check controlPanelSwitches(cs).
    if cs = "alarmOff" then invoke alarm set to "off"
    ⋮
    default for cs = none
  reset the switches and signalValues
  do for all sensors
      invoke checkSensor procedure returning ValueOf Signal
        if ValueOfSignal > bound[TypeOfAlarm]
          then phone.msg = msg[TypeOfAlarm]
            set alarmBell to "on" for t
            set system status = "condition
                                for Alarm"
            parbegin
              invoke alarm procedure "on", t;
              invoke phone procedure set to
```

```
                typeOfAlarm, phoneNumber
              parend
            else skip
          endif
        enddofor
      end alarmManagement
```

The code for the alarm management component is a rough outline of the design code. It consists of more English sentences than a programming code. However, the sentences are clear enough for the designer to replace them with code as soon as the design is approved. As a result of this, the designer can include only the required amount of detail and enhance the amount of detail as per requirement.

4.3 PERFORMING USER INTERFACE DESIGN

4.3.1 The Golden Rules

Q32. Discuss the Mandel's design principles that allow the user to maintain control.

Answer :

Model Paper-I, Q14

"Place the User in Control"

The above statement refers to one of the three golden rules supplied by Theo Mandel in order to achieve a good interface design. A deep illustration on above mentioned statement is given below.

People who deal with the first phase of software development i.e., information gathering phase to prepare the customer requirement specification, often get astonished with certain peculiar requirements of the users. For example, if windows based graphical interface is a resultant system to be developed and if say certain customer specifies that he requires the system (to be developed) to be well formed such that, the software implementing the system should be capable enough to judge the user requirement even before he attempts for it. Though this statement looks quite contradicting but by analyzing it deeply we may come out with many conclusions like,

- Every user intends comfort while he is dealing with any of the graphical interface.
- Moreover, the extent of comfort should be of high value that it should completely satisfy the user as well as the environment, the system should be understandable by normal human mentality i.e., the system should not resort any complex interfaces creating an environment hard to understand.

Hence, with above mentioned specifications, it can be realized that various routines, procedures or certain limitations implemented by the designer of a system are meant to favour the user. But these specifications some times are implemented by the designers to make the implementation easy but at the same time they turnout to be complex for the normal users operating the system. Hence, by considering these consequences few principles have been deduced favouring the users in order to maintain control. These are as follows,

Principle: "Allow user interaction to be interruptible and undoable"

Explanation: To analyze above mentioned principle, consider a situation where a user is working with multiple tasks. Hence, the system should be so compatible to allow this user to switch between these tasks with the cost of not losing the data. Also, the user should be capable enough to terminate or undo the executing tasks etc. Switching between different tasks is referred as the process of interrupting the current task and moving ahead with the next task.

Principle: "Hide technical internals from the casual user"

Explanation: In this principle, stress is laid to keep away the user from the technical details of the system. The user should view only the outputs of the program. He should not be made aware of the implementation details of the program (or) the operations of various entities of the system, which are running collaborately to display the output.

For example, air application should not prompt user to type any operating system command.

Principle: "Design for direct interaction with objects that appear on the screen"

Explanation: The interface should allow a meaningful display of objects, such that they should resemble the physical real world entities. Hence, the user, just by viewing once should be able to identify the purpose of a given object. For example, we often find an icon  displayed by the operating system, to refer an object which is responsible to control volume of the system.

Principle: "Provide for flexible interaction"

Explanation: The above principle specifies, that the user interface should agree with the priority of interaction on the feasibility of the user. For example, if a person is blind, then the system should supply feasibility by allowing him to supply commands by means of using keyboards or voice commands etc. At the same time, it should be remembered that the provision of priority should not be an alternative to the mode of interaction which suits best for a given application i.e. For example , it often turnout to be critical to use a keyboard to accomplish sophisticated diagrams using an "Auto CADD" software.

Principle: "Define interaction modes in a way that does not force the user into unnecessary or undesired action"

Explanation: In an interface design, the designing of interaction mode is of prime focus. The above principle lays stress on this aspect, since the user gets frustrated if he/she remains too long or too short in it. Hence, the user should remain in this mode for a moderate period of time. For example , assume that the user is using the paint software and he needs to draw a square on the monitor. This he can do it easily by selecting the square from the list of shapes provided. The list of shapes should get deactivated as soon as he selects and pastes the appropriate item on the monitor so that he can switch on to another mode easily, without worrying much of shapes mode.

Principle: "Streamline interaction as skills levels advance and allow the interaction to be customized"

Explanation: The given principle favours to improve the design of the interface, where the user intends to use a given series of actions multiple times. In this regard the concept of say, "Macros" can be used effectively.

Q33. Discuss the design principles that reduce user's memory in user interface.

Answer :

It is estimated that the best user interface design is one, which makes the user to apply very little of his memory. Hence, the above mentioned principle is referred as one of the golden rules suggested by Mr. Theo Mandel in favour of designing the user interface. According to him, the interface should be so interactive that, it should be capable of storing certain valuable information which is necessary and sufficient enough to make the user to recall the procedures of interaction, thus allowing the user to apply very little of his mental ability. Following are the certain principles suggested by Mr. Mandel in this aspect,

Principle: "Disclose information in a progressive fashion"

Explanation: This principle favours disclosing of information, belonging to a given instance of an interface in a step by step manner. This means that a large volume of information related to any instance of an interface should not be directly provided to the user. Rather, it should be initiated with a less volume of information in first step, followed by increasing it next higher steps.

Principle: "Define shortcuts that are intuitive"

Explanation: While designing a user interface, short cuts should be preferred to wider extent. This makes interaction very easy and in a less time, since the user need not worry about the entire procedure, rather, just remembering certain shortcut values, he can get the work done. For example, just by remembering the shortcuts, "ALT + f + s", he can store a given file easily instead of halting the given session by clicking the escape button, using the mouse, clicking the file drop down list and then selecting the "store" menu item, which is a cumbersome and time consuming process.

Principle: "Establish meaningful defaults"

Explanation: This is a direct principle, which specifies a default value to be included in every application of the user interface. This is an important aspect, since a given interface can be used by large number of users (normal to professional). Hence, a means should be provided to select from a set of values, which are alternative to defaults and a "reset" option should also be included which can switch to a default value after remaining on the other alternative values.

Principle: "Reduce demand on short-term memory"

Explanation: The demand for short term memory rises only when a user is dealing with certain complex computations. If the interface includes certain mechanisms to remember the previous interactions, this itself will reduce certain amount of memorizing. This task can be accomplished just displaying certain "Visual Cues", which itself can act as remainder to the user with which he himself can recall his past interactions.

Principle: "The visual layout of the interface should be based on a real world metaphor"

Explanation: The above principle lays stress on the layout structure of the interface. It says that the layout of an interface should resemble the real world system terminology, referring which the interface has been built. For example, if the interface represents the restaurant cash system, then it should include the layout structure, which is frequently observed at normal restaurant's cash counter i.e., the systems should include fields like total amount, amount paid, amount to be returned, etc.

Q34. State and explain Golden Rules of UI. How these rules affect on UI analysis and design?

Answer :

Three golden rules given by Theo mandel for a good interface design are as follows,

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent.

1. **Place the User in Control**

For answer refer Unit-IV, Q32.

2. **Reduce the User's Memory Load**

For answer refer Unit-IV, Q33.

3. **Make the Interface Consistent:** This is one of the golden rules suggested by Mr. Mandel to achieve best user interface design.

Following are the principles suggested by Mr. Mandel in favour of making the interface design consistent.

Principle: "Maintain consistency across a family of applications"

Explanation: Whenever we deal with a specific kind of task, the given set of applications (associated with that task), should be implemented by following certain prescribed rules. These rules should be equally adhered to all these applications. This is important to maintain consistency.

Principle: "Allow the user to put the current task into a meaningful context"

Explanation: In general, a given interface is the outcome of combination of hundreds of interactive modules. Here, each module or layer provides interactive entities (through which interaction takes place) such as buttons, checkboxes, images etc. It is prescribed that, selection of interactive entity to be made depending on the context. When switching from one module to the other, a path should be provided, so that it remains an indicator to determine the location of the user (as users can get deep into several modules or layers. Hence, indicators remain effective in getting deep into or coming out of these layers).

Principle: "If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so"

Explanation: In this principle, producing the interface in favour of user's interest is of prime focus. In general, if a given user gets habituated using several key's for a definite task such as ALT + f + s to save current file, it is prescribed that, such common usage should not be altered to some other keys. Sometimes these alterations can make users frustrate.

4.3.2 User Interface Analysis And Design

Q35. State and explain the different models that come into play when a user interface is to be analyzed and designed.

Answer :

Whenever we analyze the user interface design process deeply, we encounter four different models, with each model having it's own priority and importance. A brief illustration on these models, is as follows,

(I) **The User Model:** Whenever we design a user interface, the user model plays a major role. It is developed by software engineers, which includes a track of user profile such as his age, sex, educational background and various other considerations. This is an important criteria since, it is the end user who is going to deal with the product for ever developed by a given software development team. According to this model, the end user can be one of the following.

He may be "Novice"

Novices refers to a person who may not be aware of the process of interaction of the system which is developed, but possessing only little knowledge of usage of computers or the given product. They have little semantic knowledge but no syntactic knowledge.

(or)

"He may be knowledgeable and intermittent person"

In this category, the end users though might have a domain knowledge to certain extent, but at the same time they lags in general system usage which remains important enough to effectively deal with the application developed. They have good semantic knowledge but low syntactic knowledge.

(or)

"He may be knowledgeable and regular user of the system"

In this category the users may remain proficient in both the fields i.e., he may be sound in application domain as well as can effectively manage the system. These kind of persons often develop skills and resort to frequent shortcuts to mature the interactions. They have good semantic as well as syntactic knowledge.

(ii) The Design Model: In the user model, we deal with the users specifications. In the Design Model we deal with the software specifications which is again summarized by the software engineers. These specifications include the data in its lowest terms followed by architectural, interface and various procedures respectively.

(iii) The End-user Specific Mental Model (or) System Perception Model: As the name suggests, the mental model refers to the end user visualization of the system or the application to be developed. This brings out the familiarity of the end user. In this aspect, usually the user who had worked with the same domain at least once may be capable enough to describe the working of the application effectively. This model may be helpful to a large extent in generating the application.

(iv) The Final and the Most Important, the Implementation Model: This is the final model, where the given developed application is matched against all the essential specifications of the system, if the application is satisfying these specifications, then it can be assumed that the application can provide comfort to the users. It is basically a combination of all the above specified models and to ensure success of this model, the design model should satisfy the guidelines present in the user model. And finally the implement action model should be constrained enough to satisfy the technical aspects of the software as well as the system on which it is going to be implemented.

The implementation model is the most important since, this model is to be formed by satisfying all the three models specified above, though there may exist large differences between them. Hence, decreasing these differences and satisfying all the above mentioned analogies, will then lead to success of the application.

Q36. What are the goals of the user interface design?

Answer : In the above session, we had dealt with the user interface analysis and design models. Now, consider the design process. The design process is a sequential procedure which is divided into four phases. These phases are represented using a spiral model and each phase can be repeated more than once before the completion of entire design process. The four phases and the spiral model is shown below.

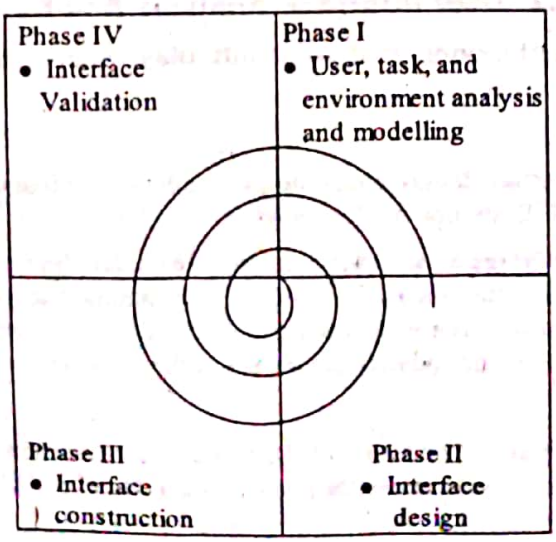


Figure: Spiral Model Representing the User Interface Design Process

Phase-I (User, Task, and Environment Analysis) : In this phase, the user requirements are acquired depending on the user type and its category. Once the users are categorized, now, it all depends on the software professionals to gain the user level perception of the system to be developed. Also depending on the user category, their requirements are separated. Once the sufficient user requirement information is gathered, a deep task analysis is performed, where different ways through which a user operates the system to accomplish his objectives are determined by traversing through all the four phases of the spiral model. Later details on the environment where the system is going to be implemented is gathered. This information comprises of details like,

- Does the system hardware is exposed to certain calamities like light or large noise, heat etc.
- The location of interface.
- Is the interface surrounded by other co-systems which can be operated by the user, without disturbing the current system etc.

Hence, all the above specified information is gathered and an analysis model is formed.

Phase-II (Interface Design): Interface Designing deals with the designing of the system. Here, different objects, which are active part of the system are determined and implemented. Their operation details are also specified in this phase.

The following are the major steps to be followed during interface design,

(i) Defining interface objects and operations by using the information collected during interface analysis.

(ii) Modelling the behavior of the user actions (events) which may change the user interface.

(iii) Prototype each interface as it will be given to the end-user.

(iv) Provide a means to the user through which it can determine the system state using the user interface.

Phase-III (Implementation): This is also known as the construction phase. Here, a prototype is initially constructed, which is later brought up into its originality by using the interface development tools.

Phase-IV (Interface Validation): It is Just like testing phase where the correctness of the system is tested against the user requirement. Following are the issues which are frequently considered in this phase.

- Is the system completely satisfying the user?
- Does the system possesses easy to be learned and adopted features?
- Is the system satisfying the variations and the details specified by the user in user requirement gathering phase etc.

4.3.3 Interface Analysis

Q37. Explain about user interface analysis and user analysis.

Answer : **Model Paper-I, Q15(b)**

User Interface Analysis

It refers to the process of examining the requirements of interface to develop a good user interface. It is a central principle of all the software engineering process models that before attempting to find the solution to a problem, it is better to analyze and understand it problem includes understanding,

1. The end users
2. The tasks of end users
3. The content of the interface and
4. The environment in which the task will be performed.

1. **User Analysis:** In order to meet the requirements of the end user, it is necessary to analyze the mental image developed by the end user. The mental image of the users may differ from the design model of the software engineers. Hence, the only way to determine the convergence of the mental image and the design is to understand the users and their requirements and collect their views. This can be done by taking the information from a broad array of sources.

Interviewing the User: The direct way of analyzing the user is conducting interviews with users to understand their needs, motivation, work culture etc. This interview involves one-to-one meetings between the software team representative and the end user.

Sales Input: Here, sales people will gather all the information of customers and users by meeting them regularly. This information helps in categorizing the end users and understating their requirements.

Marketing Input: It also provides the information about the end-users, and how differently each user uses the software.

Support Input: The inputs such as user's likes and dislikes, the features that work or not etc., will be provided by the support staff. Support staff will seek this information by talking with the users on daily basis.

2. **Task Analysis and Modelling:** The goal of task analysis lies in the underlying solutions to the following questions,

(i) What sort of work must be performed by a user in specific circumstances?

(ii) What will be the work-flow?

(iii) Which tasks and subtasks need to be performed?

(iv) What should be the hierarchy for tasks?

(v) Which problem specific domain objects will be manipulated by the user?

Various analysis techniques are used by the software engineer to find solutions to the above problems. All these techniques are applied to the user interface as follows,

For remaining answer refer Unit-IV, Q38, Q39.

3. **Display Content Analysis**

For answer refer Unit-IV, Q40.

4. **Analysis of the Work Environment in User Interface Design**

For answer refer Unit-IV, Q41.

Q38. Write short notes on,

(a) Use-cases

(b) Task elaboration

(c) Object elaboration with respect to user interface design.

Answer :

(a) Use-cases

Usage of use-cases differs with respect to the context (i.e., the situation where it is applied). For example, use cases can be applied in generating models which describe the user interactions with a system (or) when they are considered in task analysis phase, each of these use-cases depicts certain tasks implemented by a given user. The above illustrations describe only the modelling aspects of a given system. But while dealing with use cases, it is often important to be known that, in many occasions, use-cases, are also written i.e., use cases can also be adhered in the form of paragraphs describing a particular situation. Assume that we require a house to be constructed, for which we approached an architect. When the architect asks "How would you like your house to be", then our answer may be follows,

"I want my house to be sharp and straight forward. As my family is large, hence, it should consists of many rooms with good spacing and ventilations. I am extremely concerned with the number of toilets and bathrooms. There should be some spacing between the boundary walls and the actual construction, so that the spacing can be accommodated with beautiful lawns. With an additional spacing for garage".

The above illustration is nothing but a use-case. This use-case becomes essential for the architect while designing the blue print of the house on the system. He initially picks up the required objects and other various essential tasks. Apart from this, to attain gratitudes from us, the architect can also adorn the design with several live carvings etc.

A use-case can also accept several variants i.e., a use-case can also form a part of the other use-case and can extend the behaviour of a concrete use-case. Finally, use-cases can also be applied to whole or part of a system.

(b) Task Elaboration

It is a process of refining the processing task while performing task analysis, usually an elaborative approach is considered. This is important in understanding the activities performed by user, which are to be now adhered to the user interface.

In task elaboration, the foremost the fact to be considered is identification and classification of tasks. To analyze the task elaboration process in detail, consider the following example.

Assume, that a software is needed to be developed for a construction company which is capable of producing drawing related to buildings, bridges, towers etc. Assume that basic tasks that a computer programmer performs in the construction company is designing the outlay of buildings, positioning of flats at appropriate positions, lift systems supporting all the layers of the building etc. Now these concrete tasks can be further elaborated like positioning of flats at appropriate positions reflects tasks like placement of doors, windows, ventilations etc., to each flat, and the lift systems supporting all the layers can be further elaborated into, the dimensions of the lift, locality of lift operation, starting point and ending point of the lift etc. It has to be noted that, these subtasks can be elaborated.

(c) **Object Elaboration:** When Object elaboration process is considered, it is the software engineer who is playing the key role in this entire process. He initially analyzes the use case data and also concentrates on other valuable pieces of information provided by the user. Apart from this information, the software engineer also looks for the blue print designed by the architect (as discussed in the use cases) in order to extract every possible object. Now, these objects are translated to form a class. In the next instance various attributes of this class are defined and the operations which are intended to be performed by these objects are ascertained as the operations of this class. This is the process of object elaboration. Finally, one has to remember that, the operations which are defined are not implemented at the same instant, rather the implementation of these operations is observed only after the designing process progresses further.

Q39. Write short notes on,

(a) Workflow analysis

(b) Hierarchical representation.

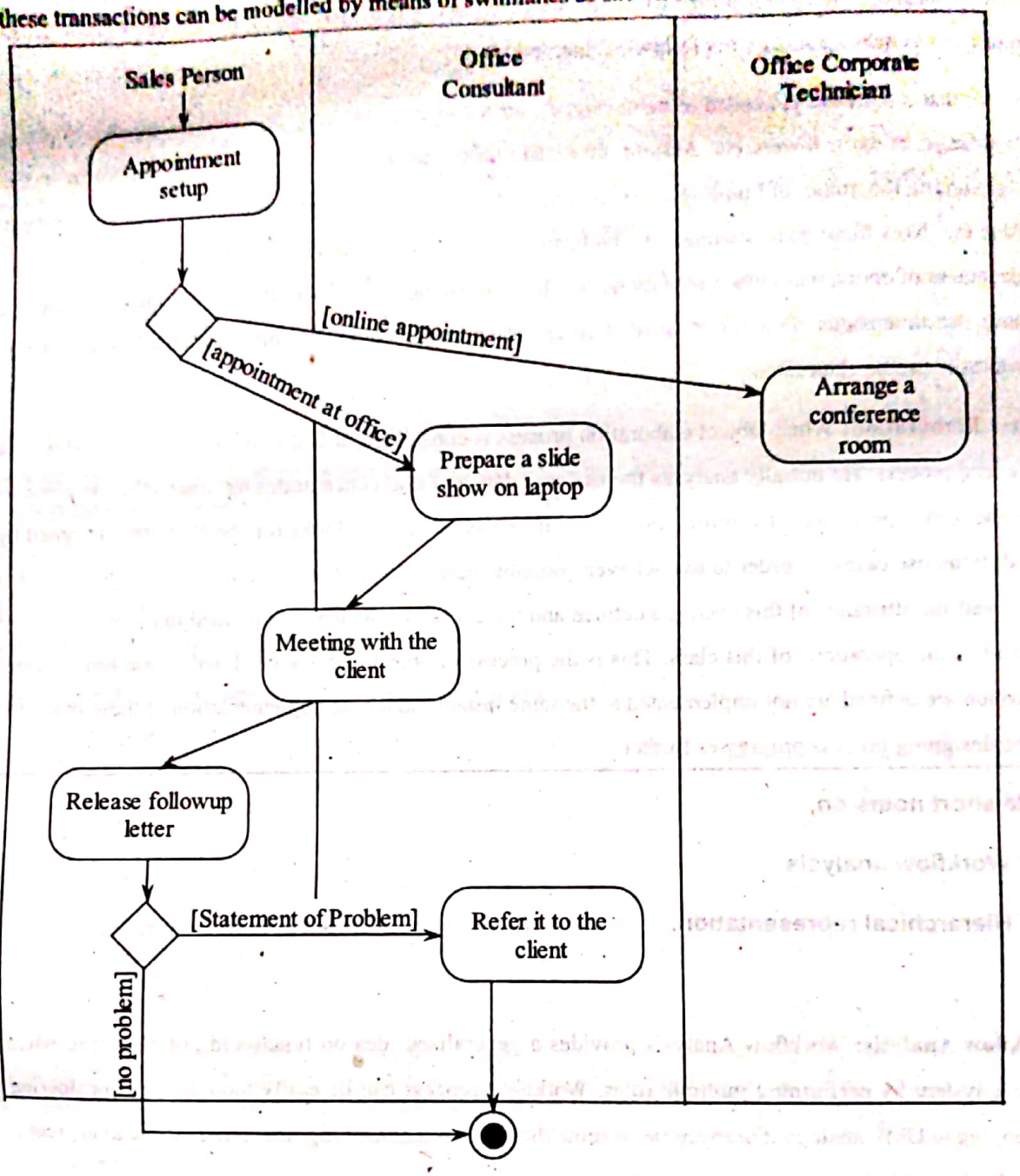
Answer :

(a) **Workflow Analysis:** Workflow Analysis provides a generalized idea on functioning of a system, when multiple users interact with a system by performing multiple roles. Workflow process can be easily modelled by considering the swimlanes concept belonging to UML analogy. For example, assume that there is a consulting firm and they intend to greet a new client. For this situation following is the set of activities,

- An appointment is initially fixed with the client by a sales person.
- Here, there are two possibilities i.e.,
  - The appointment can be either on-line i.e., via internet (or)
  - At the office.
- If the appointment is on-line, then arrange a conference room for on-line interaction or if the appointment is at the office premises, then prepare slides show on the laptops which can be displayed to the client.
- Now, the consultant, salesman and the given client greet depending on their selected place and time.

- Here, the salesman usually follows up with a letter and finally, if certain statement of problem exists is intimated by the consultant to the client through some means.

Now, these transactions can be modelled by means of swimlanes as shown below.



As each user is designated differently with respect to other users, hence, all these characteristics differs from one another. Finally, each activity mapped in above diagram can be easily elaborated without much loss of information.

(b) Hierarchical Representation: In the workflow analysis, we had observed the modelling of various transactions or activities performed by each user. Now, each of these activities can be further elaborated sequentially or stepwise. This elaboration of activities is often referred as Hierarchical representations. For example, consider the hierarchical representation of the above workflow model. Here, the activity appointment setup can be described in detail as follows,

Appointment setup

Client verification

- client name
- client address
- client's ID

Appointment details

- appointment details
- appointment time
- appointment place etc.

Q40. How do we determine the format and aesthetics of content displayed as part of the user interface?

Answer : User interface is a mechanism that helps the user to enter into a dialog in order to operate the system and obtain the desired results. Once data is processed and analyzed, its presentation can be arranged in such a way that it meets the needs of user. The user tasks are helpful to present a variety of different types of content. The content of the data can be classified in two forms i.e., in a graphical forms like picture, animations etc., and other form of data is non-graphical presentation such as spread sheets, pages of MS-word etc. This technique is used for knowing presentation the output form of data which is developed by the user. Sources such as,

- Components
- Database
- External systems.

Therefore, the user will have several options of viewing the system output information. The format and aesthetics of the display content are involved at the interface analysis level. There are different questions to be answered while presenting the display content. They are,

1. Will system resolution be scaled to fit the display content on the screen?
2. What are the several kinds of data assigned to screen locations?
3. Will the user be able to customize the screen location for displaying the content?
4. Will on-screen a unique identifier number is assigned to all display contents?
5. How will colour be adjusted for better understanding?
6. How the error messages and warnings will be displayed to the user?
7. How the large documentation will be separated for better understanding?
8. Are methods available for large collections of data?

The content presentation may be a direct representation of the input information or it may present the information graphically after answering all of the above questions.

Q41. Discuss the importance of analysis of the work environment in user interface design.

Answer :

Analysis of the Work Environment in user Interface Design

The designing of user interface needs to complete the interface analysis such as user analysis, task analysis, content analysis and work environment analysis. Each of these interface analysis are essential for the user interface design. The importance of work environment analysis in user interface is explained by Hackos and Redish. According to their studies, work environment plays a vital role in designing the products because users always dream to work in a user friendly location. Users do not perform their work in isolation. While working in an environment, they may be influenced by the various activities such as,

- The physical characteristics of the workplace.
- The kind of instrument used at the workplace.
- The interaction among the users at the workplace.

If the designing of products do not fit into the environment, then it may be difficult to design an efficient product. User-interface can be easily maintained in certain conditions which provide better facilities for accessing each component of the system.

Whereas in others, it may be quite difficult because of the different factors such as noise, unavailable instrument and small display unit etc. For example, the aeroplane workplace. In this analysis, the physical work environment will also be studied then the work analysis identify the following tasks. They are,

(i) Location of the user interface.

(ii) The execution of user interface along with other tasks.

(iii) Special human factors such as physical and cultural activities.

Apart from the physical environmental factors, the culture of workplace also play an important role such as support provided to the users, understanding among the users and system interaction can be determined for accuracy and efficiency of user interface. These are the factors which help to design the user interface.

4.3.4 Interface Design Steps

Q42. Elaborate on various interface design steps.

Model Paper-II, Q8

Answer :

End of Interface analysis marks the beginning of interface design issues. Interface design is a step by step process in which each step may get repeated multiple times and the new step generated contains upgraded qualities than previous steps. Following are few important steps effective in developing the interface design,

Step1: Initially, consider the useful information obtained from the interface analysis phase. Extract interface objects from it and define their (objects) operations.

Step2: Model the events, that may cause the change of state of the user interface.

Step3: Prototype each interface state, exactly as it will be given to end user.

Step4: Consider the information obtained from the interface. Using this information mark the way the user interprets the state of information.

Irrespective of above mentioned steps, a given interface designer may initiate the process of designing by neatly sketching the required interface states and later defining the essential objects and its related information (i.e., attributes, and operations).

It is always prescribed to the designer to traverse through the golden rules, produce models depicting the implementation of the interfaces and lastly define the platform that will be taken into consideration.

Implementation of Interface Design Steps: Following are the steps essential while implementing the interface design,

- Initially, identify the objects and its associated operations. This is usually done by traversing through the use case description. Later, these are sufficiently elaborated and refined.
- Now, categorize these objects, under the following types i.e., source, application and target objects respectively.
- When all the possible objects are identified and their relative operations are defined, then start creating the screen layout.
- To make the layout attractive, several text, graphic images, icons etc., are placed at suitable locations. To make the layout to look more lively, any real world entity which is satisfying the situation (while making the layout) can be added.

Developing User Interface Design Patterns: In general, the design pattern refers to an abstract representation, which can be treated as a solution to a well formed problem. As there are many highly developed graphical user interfaces available today and hence, any of them can be utilized effectively to obtain a wide variety of design patterns.

"Various design issues associated with the user interface design"

Generally, there are four major design issues which should be addressed at the beginning of the design process so as to lower the constraints (such as delays in the project etc.) which may dissatisfy the customer. Details of these issues are illustrated below.

- **Error Handling:** While operating computers, we often encounter certain error messages displaying an unknown information i.e., the messages of the form "My computer not responding due to error at 1736 and requires to terminate immediately. Such messages often frustrate the users since, it neither carries any valuable information nor it describes any measures to escape such errors". Hence, while designing user interface, following are few effective guidelines to be followed, for making error handling mechanism easy,

1. The messages should be understandable i.e., it should be displayed in the language preferred by the user.
2. The messages should carry valuable information to escape errors.

3. Actual cause of the occurrence of error should also be included in the error messages.

4. While displaying an error message, certain graphics should be associated with message i.e., a strong beep sound or flashing or colour changing periodically, etc.

5. Finally, the information embedded on the message should not comment or blame the operators for their erratic operation.

Inclusion of Help Topics (or) Options: Almost every software includes help topics which may guide the users for the proper usage of the software. Hence, while including help topics following are few essential facts to be considered effectively.

1. In order to provide query, sufficient space such as text fields or certain options should be provided.

2. While responding to a given query a separate window should be displayed narrating the suggestions to the operator's query or should specify the exact path to obtain the solution or should narrate the ways through which the operation can gain valuable information (often diagrammatic explanation is useful)

3. Once the operator is satisfied, there should be sufficient ways through which the user may resume to normal operations. These ways can include a return button, a function key etc.

4. While displaying a window, the data presented on it should follow a flat layout. The text should include certain important key or can contain a hierarchical tree structure or can include certain hyperlinks which may take the user from one document to other.

5. The help a topic or options should be included to only few functions. But help data should contain information to all the applications specific to a given function.

Inclusion of Menu and Command Labelling: The inclusion of menus and command labelling should be such that, it must satisfy the following questions.

1. Is there availability of a command with almost every menu item?

2. What will be the form or representation of command? For example, function keys (F1 etc) or key sequences like ctrl + s or typed words.

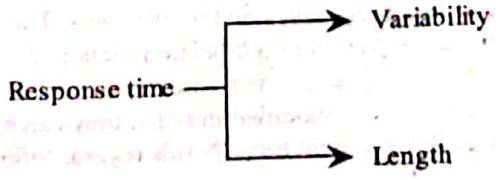
3. Does all the menus represent the purpose of their inclusion? Are they easy to learn and remember?

4. Does the submenus (belonging to a concrete menu) satisfy the condition for which it is referred by an item of its concrete menu.

5. Are there any facilities with which a given operator can easily abbreviate or create short forms for the commands.

6. Are there any provision for the operator if he/she forgets the command? Is the command too complex to be remembered.

The System Response Time: It refers to the time taken by the system to respond to a user activity. It is usually measured from time of generation of an event by the user till the response made by the system. The system response time has two important characteristics i.e., variability and length.



Length refers to the time between the event generated and the response made by the system. Variability is the deviation in the available average response time. As the variability gets decreased, it enriches the user during system. As its negative aspect, if the variability increases it irritates the user and he blames himself, thinking that some error might have crept during his operation.

4.3.5 Design Evaluation

Q43. With the help of a diagram explain in detail the interface design evaluation cycle.

Answer : Model Paper-III, Q15(b)

Interface Design Evaluation Cycle

After the completion of designing phase, a user interface prototype is developed which is later evaluated. Here, evaluation is important so as to affirm the granularity of the interface i.e., to what extent the given interface satisfies the user. Here, the user feedback is considered, also the given prototype is lent to various users and their comments are taken based on which statistical model is developed. These statistical models are helpful in determining the aspects where the given interface lags and are to be modified and also those aspects which completely satisfy the user. Hence, in short we can say that, complete analysis of working of given interface is made. Various stages which remain crucial during evaluation of user interface is diagrammatically represented below.

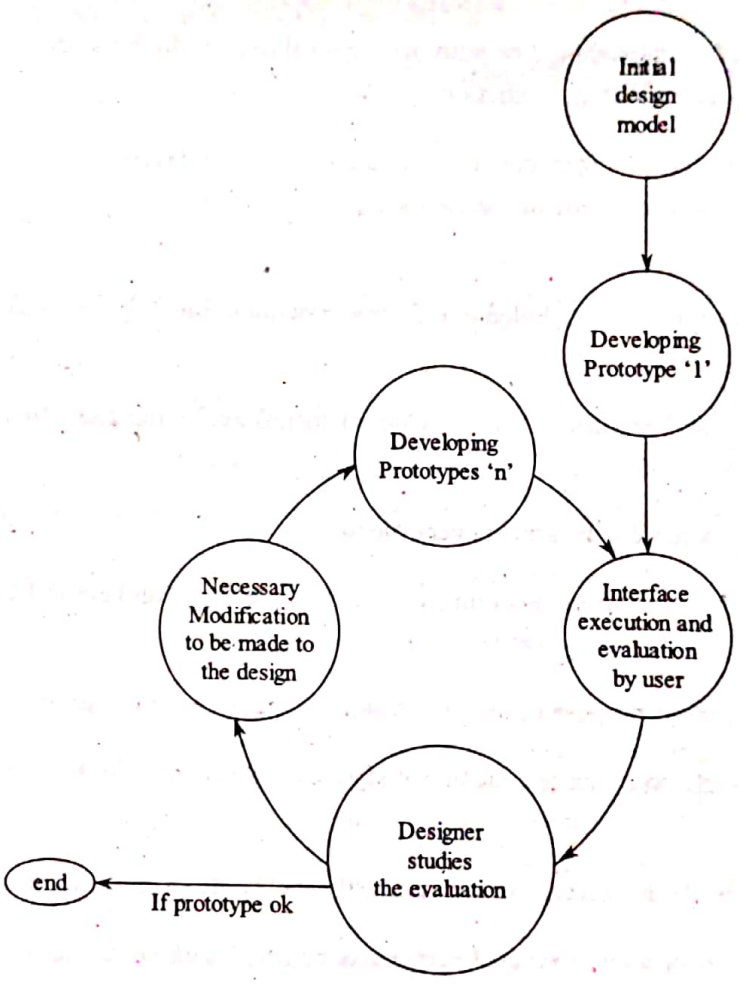


Figure: The Cycle Depicting the Process of Interface Evaluation

Consider the above cycle. Once the initial design model is developed in the next step a user interface prototype is built. This prototype is later transmitted to the user for execution. After execution, the designer is provided with the feedbacks from the user. These feedbacks may include direct comments or reports developed by the users. The designer deeply analyzes these reports and accordingly modifies the prototype. The new prototype (after modification) obtained are again transmitted to the user for further analysis. This cycle continues until the user is completely satisfied, causing no further modifications to be made to the prototype. However, if certain genuine problems are sorted and are modified initially, they can reduce the number of rotations of the cycle at the same time reducing the overall software development time. In this regard, following are few design criteria to be applied in initial stages of the evaluation process.

- Scripts can be built which may include various system as well as interface specification.

These scripts remain essential to the users of the interface in determining requirement of learning of the system.

- In order to determine the system interaction and efficiency, a specification can be made which includes the count of number of tasks as well as the equivalent actions which are required to be performed per task.
- The design model can include specifications like the number of tasks, its equivalent actions and also various system states. These specifications help to find the amount of mental ability a user is applying while using the system.
- The complexity and the user acceptance level of a given interface can be directly judged by interface design issues like error handling, the help menus provided and the interface style respectively.

Hence, we can say that, after the development of prototype, the designer can collect certain useful qualitative information by allowing the users of the system to answer simple questions like their agreement or disagreement or collect their scaled response, Like scales, subjective response etc or if the designer wishes to acquire certain quantitative information then, he can effectively observe a given end user by testing his frequency and sequence of actions, count of tasks completing in a given span of time, types of errors committed and its recovery time, time spent while stairing at the display etc. Hence, basing on this data, rectifications can be easily made to the existing prototype.

UNIT 5

TESTING STRATEGIES DEBUGGING, PRODUCT METRICS AND SOFTWARE QUALITY

PART-A

SHORT QUESTIONS WITH ANSWERS

Q1. What is meant by software testing? And list its characteristics.

Model Paper-I, Q9

Answer :

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.

Answer :

"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. Define Testing.

Answer :

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.

Q4. 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.

Q5. Define unit testing and top-down integration testing.

Answer : Model Paper-II, Q9

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.

Q6. Define black box testing strategy.

Answer : Model Paper-III, Q9

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.

Q7. Define metrics.

Answer : Model Paper-II, Q10

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.

Q8. Explain the merits and demerits of use-case-oriented metrics.

Answer :

Use-Case-oriented Metrics

Merits

- 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

- 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.

Q9. What guidelines should be applied when we collect software metrics?

Answer :

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.

Q10. Discuss the importance of quality assurance.

Model Paper-III, Q10

Answer :

Quality Assurance (QA) is the activity where certain auditing and reporting functions are performed in order to assess the completeness and effectiveness of quality control activities. The quality assurance is important because,

(i) It helps in defining ways in which software quality can be achieved.

(ii) It helps development organizations to examine whether the required quality level has been achieved or not.

(iii) It helps to reduce the amount of rework that will result to lower costs.

(iv) QA professionals look software from customer's point of view and hence act as their representatives. As a result the desired product is built keeping in mind the customer's ease.

Q11. Explain the relationship between quality and security of software.

Answer :

The importance of security of a software is increasing along with the growth in criticality of web-based system and applications. A low quality software can easily be hacked. As a result the security risk of it increases with its problems and costs.

Basically, there are two types of problems that occur in software,

1. Occurence of bugs that are the problems occurred during the implementation.
2. Flaws in software due to improper architectural plans.

The security completely relates to the quality. People aware of the above problems should focus on early phases of the life cycle stuff rather than in later phases.

The bugs are paid much attention than the flaws elimination of architectural flaws makes the software hacking difficult. Therefore, people should focus on quality at design phase itself in order to build a secure system.

Q12. Define quality of conformance.

Answer :

Model Paper-I, Q10

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 which, were 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 satisfactor = A complaint product + High quality + Delivery of product within the estimated schedule and time

Q13. What are the attributes of a good test?

Answer :

The attributes of a good test are as follows,

1. **It Must Quickly Detect Errors:** The probability of detecting an error quickly is possible if the tester maintains complete knowledge/details of the software and is prepared to overcome from failures in advance.
2. **It Must be Best Among its Set:** Sometimes only a subset of tests among a group of similar tests may be used for revealing errors. These tests has the probability of revealing the errors from the entire class.
3. **It Must Not be Redundant:** Each test must have a different purpose. Although they have limited testing time and resources there is no significance in performing a test with similar purpose as that of any other test.
4. **It Must Neither be Simple nor Difficult:** It is possible to combine successive steps with a test case. But the resulting side effects of this act may cover errors. Hence, each test must be executed separately.

PART-B

ESSAY QUESTIONS WITH ANSWERS

5.1 TESTING STRATEGIES

5.1.1 A Strategic Approach to Conventional Software Testing

Q14. What is software Testing? Explain test Characteristics.

Answer :

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.

Q15. What is the need of software testing? What are its main objectives and principles?

Answer :

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.

Q16. What are the main objectives of software verification and validation? Briefly explain different V and V techniques.

Answer :

"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.

Q17. 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.

Q18. What are the strategic approaches to software testing?

Answer : **Model Paper-III, Q16(a)**

The overall strategy for software testing can be diagrammatically represented using the following spiral model.

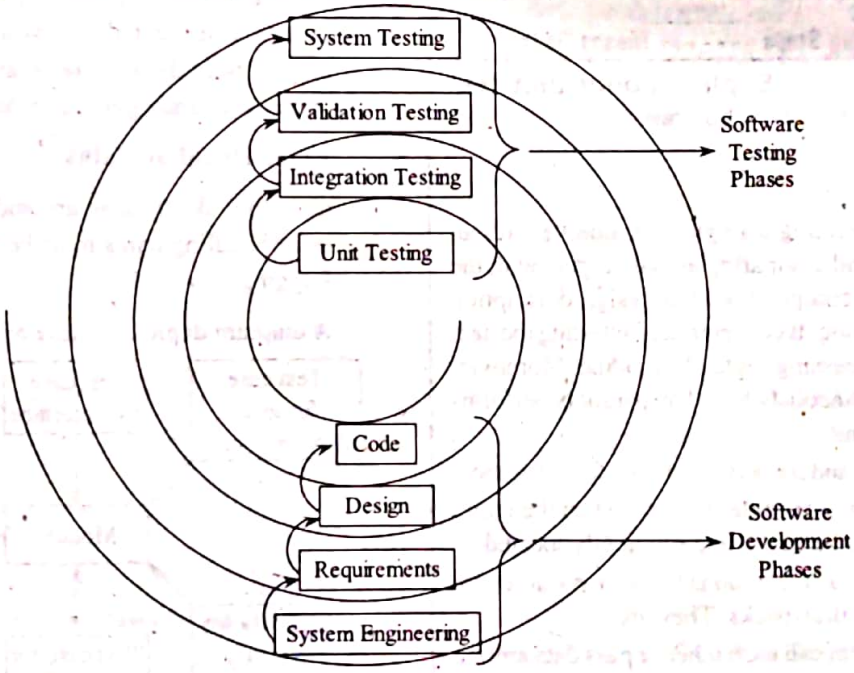


Figure: Testing Strategy Represented Using Spiral Model

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.

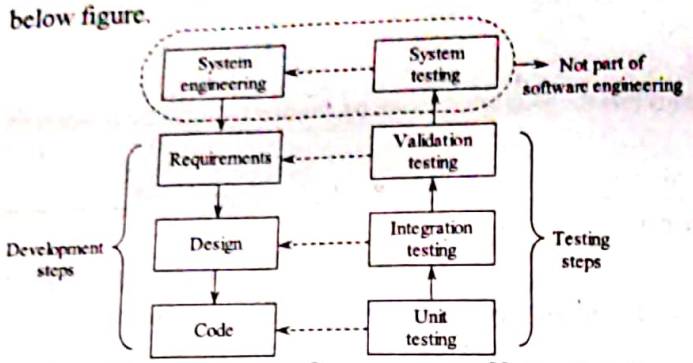


Figure: Software Testing Steps -----➤ Means "is for"

Q19. Define unit testing. Explain about unit test considerations and procedures.

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.

(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.

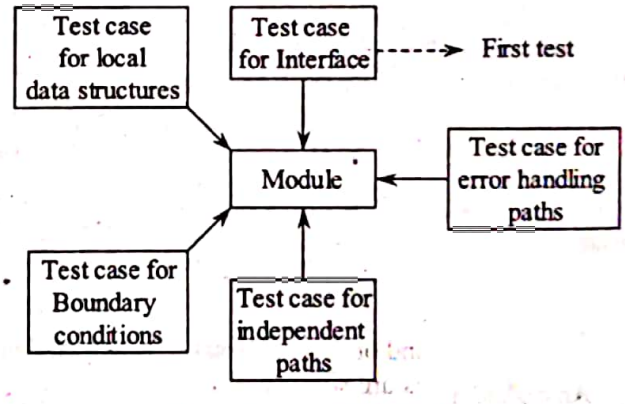


Figure: Tests in Unit Testing

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.

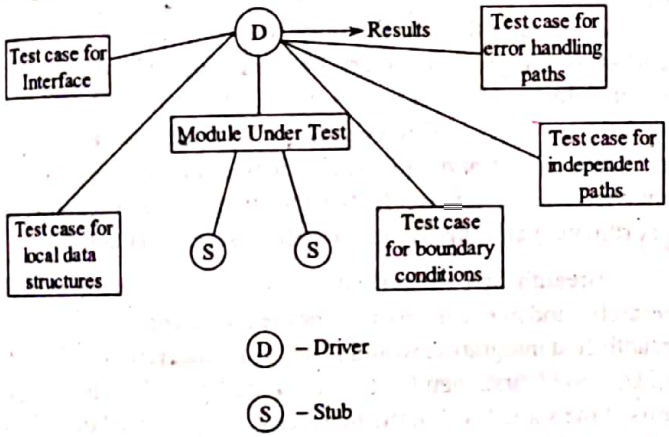


Figure: Unit Test Environment Using Driver and Stubs

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.

Q20. Discuss in detail about integration testing. What are its types? Explain the big bang approach.

Answer : Model Paper-I, Q18(a)

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,

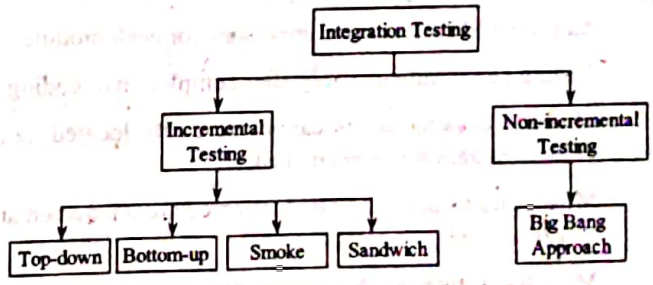


Figure: Types of Integration Testing

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.

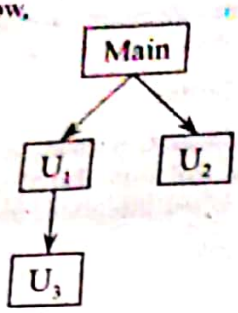


Figure: Example of Program Structure

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:

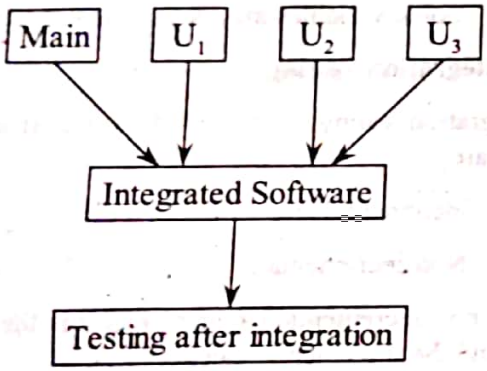


Figure: Example of Non-incremental Testing

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.

Q21. Explain about top-down integration in detail.

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.

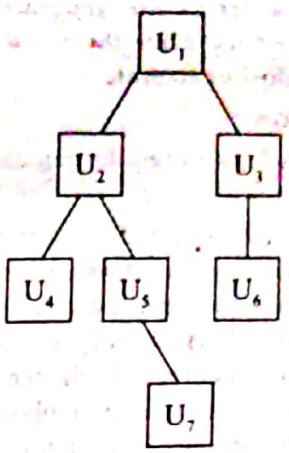


Figure: Example of Control Hierarchy

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.

Q22. What is meant by bottom-up integration test? Explain how it is implemented.

Answer :

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,

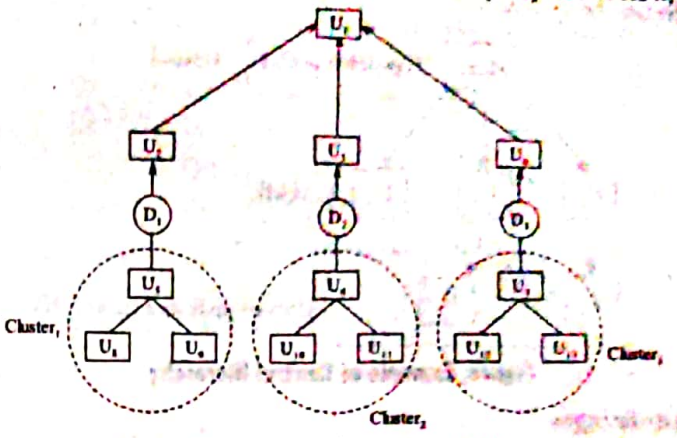


Figure: Bottom-up Approach with Drivers

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.

Q23. 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.

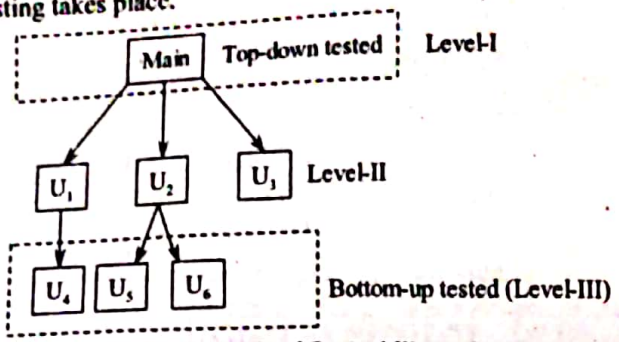


Figure: Example of Control Hierarchy

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.

Q24. Explain about "regression testing" in detail. Explain the importance of it.

Answer : Model Paper-III, Q16(b)

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.

Q25. 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.

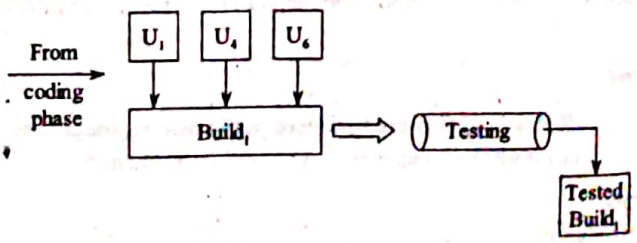


Figure (i): First Build

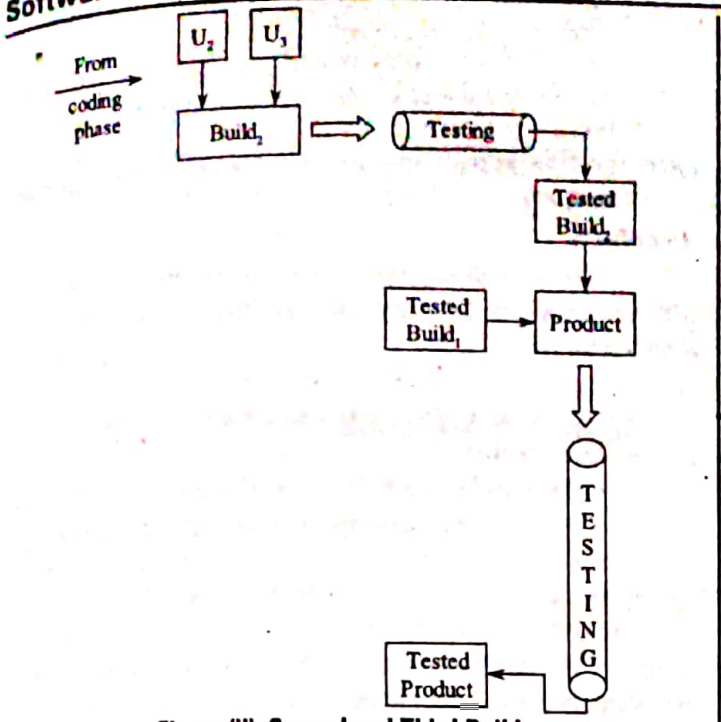


Figure (ii): Second and Third Build

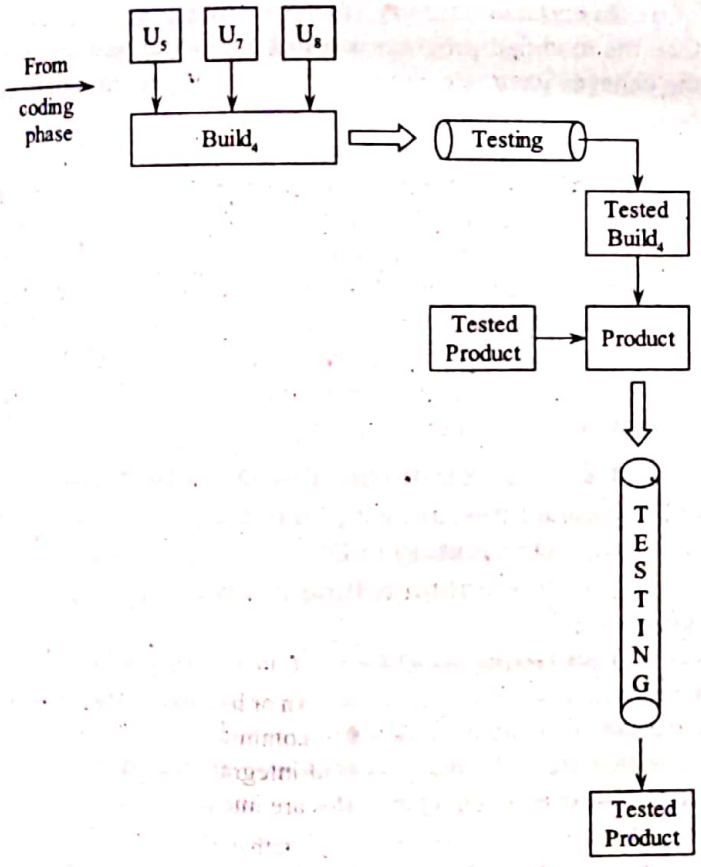


Figure (iii): Fourth and Fifth (Final) Build

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.

Q26. What are the testing strategies for conventional software? Explain them in detail.

Answer : **Model Paper-III, Q17**

Conventional software is tested using two strategies,

1. Unit Testing
2. Integration Testing.

1. Unit Testing

For answer refer Unit-V, Q19.

2. Integration Testing

For answer refer Unit-V, Q20, 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-V, Q21.

(ii) Bottom-up Integration Testing

For answer refer Unit-V, Q22.

(iii) Regression Testing

For answer refer Unit-V, Q24.

(iv) Smoke Testing

For answer refer Unit-V, Q25.

Q27. 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.

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.

Q28. Differentiate between regression and integration testing.

Answer :

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.

5.1.2 Test Strategic For O – 0 Software

Q29. Explain the following in detail,
(a) Unit testing in OO
(b) Integration testing in OO.

Answer :

(a) Unit Testing in OO Context: Unit testing in OO context involves testing of an encapsulated class. A class and its instance encapsulates attributes and operations in it. Operations are the units that are to be tested. Hence they are called as testable units. However, it is not possible to use the basic scheme of unit testing. This is because a class holds multiple varient operations and each operation may be a part of several variant classes.

In OO context, an individual operation is tested as part of a class. For example, consider a hierarchy of a class in which an operation say *P* is defined for a superclass. This operation is also inherited by several subclass and is used by each subclasses. However, the testing is done within the private attributes and operations pertaining to that subclass. The operation is tested in the context of individual subclasses since it can be used in various ways. Hence it can be concluded that unit testing for OO software is operated by the operations that are encapsulated by class and the behaviour of the class.

(b) **Integration Testing in OO Context:** Integration testing in OO context is quite difficult since there is no hierarchical structure. In addition, integration of operations one by one into class becomes cumbersome since it involves direct and indirect communication of components that develop the class. Hence there are two strategies that are adopted for integration testing in OO context. They are,

(i) **Testing Based on Thread:** This method integrates threads which are responsible for reacting/responding to an exent or input for the system. Threads refer to sets of classes. They are integrated and tested one at a time. Regression testing is performed on the threads to ensure that they are free from any side-effects.

(ii) **Testing Based on Use:** This method tests the independent classes and initiates the process of developing the system. Independent classes are the classes which use less number of server classes. Then it tests the dependent classes present in the next layer. These classes use independent classes. The process of sequential layers testing continues until the complete system is being developed.

5.1.3 Tactics : Software Testing Fundamentals

Q30. Define testability. What are the characteristics of testability?

Answer :

Testability: Testability is defined as the way of testing a program. A program is tested to detect any errors present in it. A test is said to be good if it finds errors quickly with less effort.

Characteristics of Testability: Testability has the following chracteristics,

1. **Simplicity:** The program must have simple functions, structure and code. It must have features which easily satisfies the requirements. It must have modularized architecture which does not spread the errors more. It must employ a standard platform for programming so that code inspection and maintenance becomes easier. These characteristics result in faster and effortless testing.
2. **Understandability:** The design of the architecture and the dependencies between internal, external and shared components must be understood clearly. Any modification done to the design must be modified to the testers soon. Technical document must be precise, in-detail, organized and quickly accessible. Collection of more information leads to smart testing.
3. **Operability:** The system must be designed and implemented in high-quality. This eventually results into very few bugs thereby allowing the testing to continue without any hinderence. A System which operates well can be tested more efficiently.
4. **Observability:** The inputs provided during testing genrates variant outputs. Variables and system states can be observed and queried during the execution process. Since the source code is accessible, the internal errors are detected and reported automatically. In addition, the incorrect results are recognized. Hence, everything which is seen/observed tested.
5. **Controllability:** The execution of code is done using some combination of inputs. Eventually, the outputs are also generated using some combination of inputs. I/O formats with consistency and proper structure is used. The tester directly controls the hardware and software states and variables. Hence the software and tests can be automated, optimized, specified and reproduced.
6. **Decomposability:** The system is made up of or decomposed into independent modules which can be tested individually. Hence it is possible to isolate the problems and perform testing independently.
7. **Stability:** The modifications done to the system are not frequent and can be controlled whenever they occur. They also do not disprove the already existing tests. The software is able to quickly recover from failures. Hence less number of modifications leads to less hinderence in testing.

5.1.4 Black Box And White Box Testing

Q31. What questions do black-box tests answer?

Answer : Model Paper-I, Q17(a)

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.

Q32. 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.

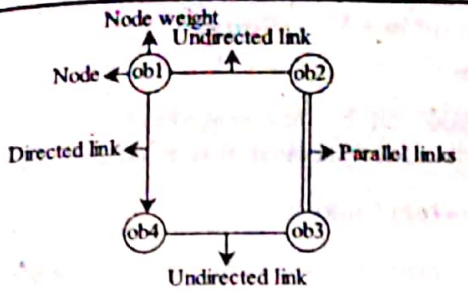


Figure (1)

Example

Consider the graph of the word processing application.

Start dimension : Default or preferences.

Background color : White.

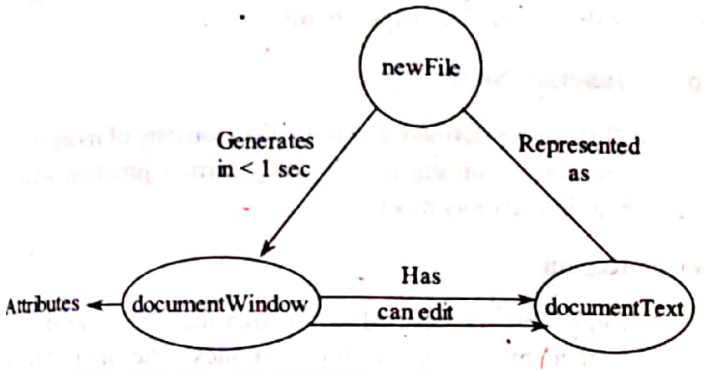


Figure (2): Graph for Word-processing Application

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.

Q33. Explain in detail about white box testing.

Answer :

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.

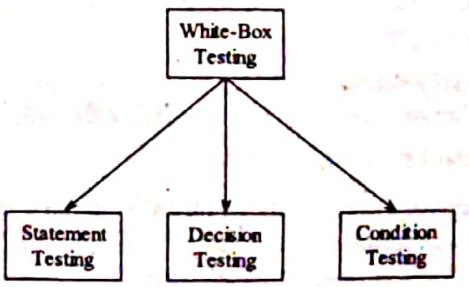


Figure: White Box Testing

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.

Q34. Give an overview of white-box testing techniques with help of flow graph.

Answer : Model Paper-II, Q17(a)

White-box Testing

For answer refer Unit-IV, Q33.

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

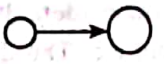
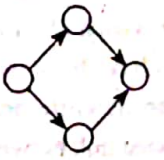
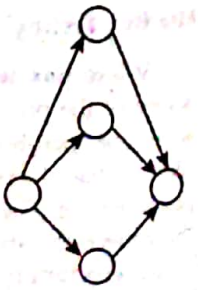
This can be defined as a node that consists of more than one arrow coming into it. They form a junction when multiple arrows meet.

(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.

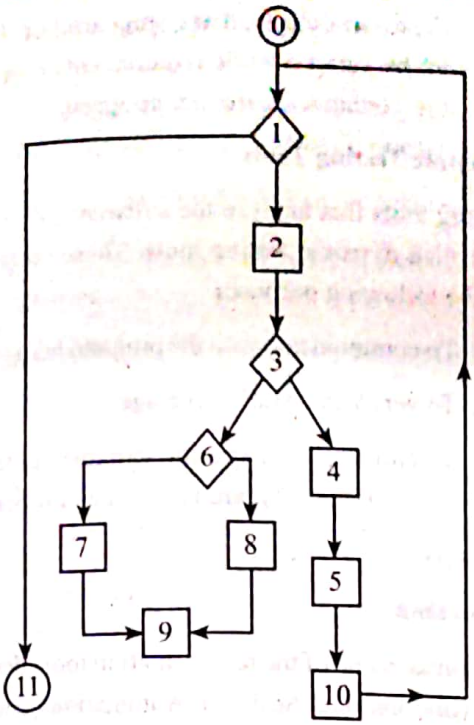


Figure: Flowchart

The flow graph for this flowchart is as follows,

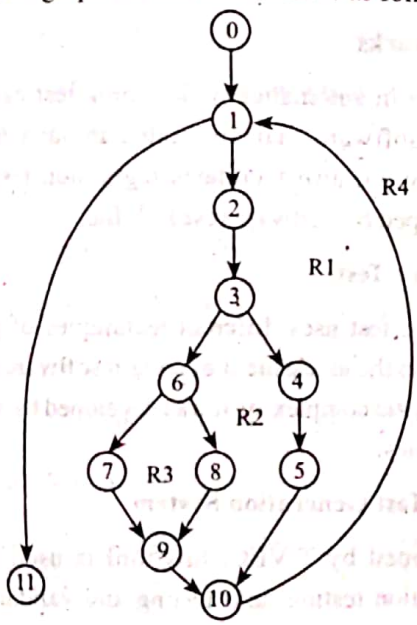


Figure: Flow Graph

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.

Q35. What is rapid cycle testing? Explain the merits and demerits of it.

Answer :

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.

Q36. 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.

Q37. Describe Boundary Value Analysis (BVA) testing for software.

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}.

Q38. Explain Black box testing. Give an account of Equivalence partitioning and Boundary value analysis techniques.

Answer :

Black-box Testing

For answer refer Unit-V, Q31, 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-V, Q37.

Q39. Differentiate between black box and white box testing.

Answer :

	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.

5.1.5 Basis Path Testing

Q40. What is basis path testing? Explain flow graph notation in detail.

Answer :

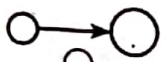
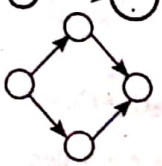
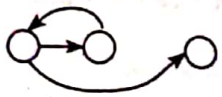
Model Paper-I, Q16(b)

Basis Path Testing

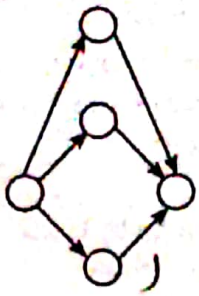
Basis path testing is a white-box testing method which generates logical complexity measure for a procedural design. This measure is used to define the basis set of execution paths. The test cases conducted on the basis set execute each and every statement in the program atleast once.

Flow Graph Notation

Flow graph is the logical representation of a control flow. It has many constructs with their own symbols repesenting 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.

CC = E-N+2

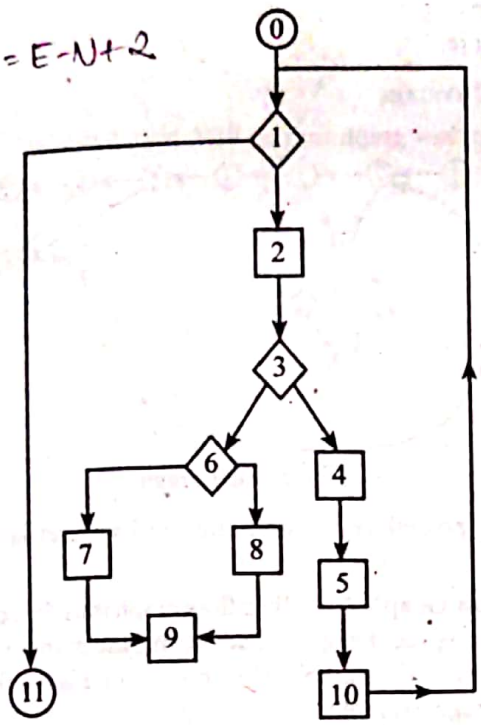


Figure: Flowchart

The flow graph for this flowchart is as follows,

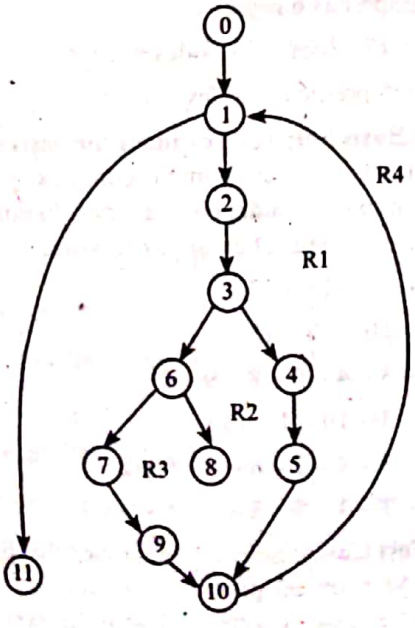


Figure: Flow Graph

A flow graph has various symbols each having a different meaning.

Following are those symbols,

1. **Circle:** It is called a node and is used to represent statements.
2. **Boxes and Diamonds:** They represent data and decisions.
3. **Edge:** It is also called as link and is represented by arrows. It indicates flow of control.
4. **Region:** It is an area surrounding edges and nodes.

Q41. Explain the concept of independent program paths in detail.

Answer :

Independent Program Paths: An independent path is a path which traverses along an edge that has not been traversed yet.

Example: Consides the following flow graph.

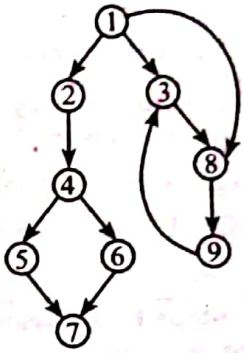


Figure: Flow Graph

Some of the independent paths for this flow graph are as follows,

Path 1: 1 – 3

Path 2: 1 – 2 – 4 – 5 – 7

Path 3: 1 – 3 – 8 – 9 – 3

Each new path in a flow graph brings a new edge. Total number of paths in the basis set is determined by the cyclomatic complexity.

Cyclomatic complexity provides a quantitative measure of the logical complexity of the program. It defines the number of independent paths in the basis set. It provides an upper bound for the number of times test must be conducted to make sure that all the statements in a program get executed at least once. Cyclomatic complexity can be calculated by using any of the following three methods.

1. Total number of regions present in the flow graph (G) correlates to cyclomatic complexity V(G).
2. V(G) = Number of edges (E) – Number of nodes (N) + 2
3. V(G) = Number of predicate nodes (P) + 1

Example: Consider the following flow graph.

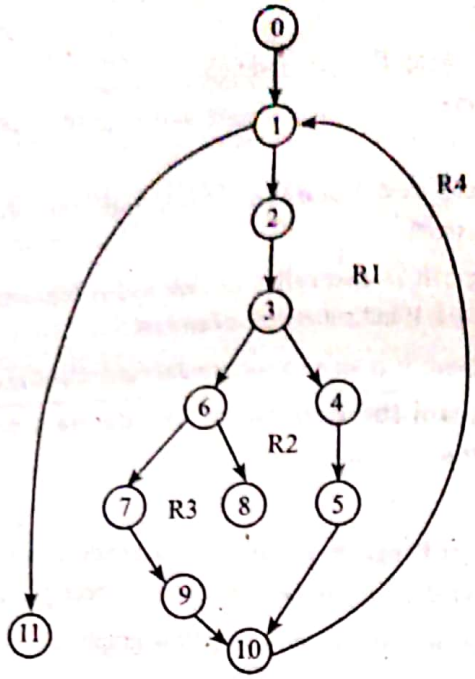


Figure: Flow Graph

By using the three methods of cyclomatic complexity, the following results have been obtained.

1. 4 Number of regions have been obtained.
2. V(G) = 14 edges – 12 nodes + 2 = 4
3. V(G) = 3 predicate nodes + 1 = 4

Q42. What are the steps for deriving basis set? Also illustrate with an example.

Answer : Example

Steps for Deriving Basis Set

The steps for deriving basis set are as follows,

1. Draw a flow graph
2. Evaluate the cyclomatic complexity
3. Evaluate the basis set
4. Prepare the test cases.

Example: Consider a procedure "average" which calculates the average of integers less than or equal to 100. In addition, it also calculates the sum and total numbers that are valid. It accepts a number (num), a maximum number (max) and a minimum number (min) as input. It returns average (avg), sum. Input and sum valid as output. The Procedural Design Language (PDL) for this procedure and the nodes is as follows,

```
1.   n = 1
     sum.input = sum.valid = 0;
     sum = 0;
2.   <> – 999 AND sum.input < 100
3.   do while num [n]
4.   increment sum.input by 1;
5.   num [n] > = min AND num[n] < = max
6.   if
7.   then increment sum.valid by 1;
          sum = sum + num[n]
     else skip
8.   endif
     increment i by 1;
9.   end do
10.  if sum.valid > 0
11.  then avg = sum/sum.valid;
12.  else avg = – 999;
13.  end if
14.  end average
```

The flow graph for this PDL is as follows,

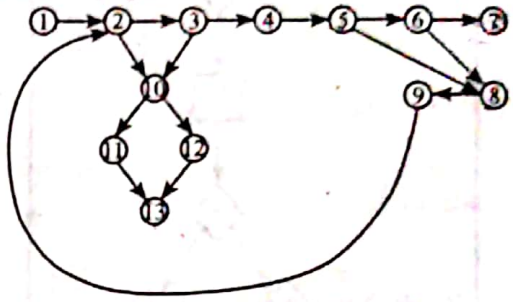


Figure: Flow Graph

The procedure for deriving the basis set is described below,

Draw a Flow Graph: Initially a flow graph must be constructed by considering the design or code using the symbols and rule for developing a flow graph. The above figure is the flow graph for the average example.

Evaluate the Cyclomatic Complexity: Then the cyclomatic complexity V(G) must be evaluated by considering the conditional statements present in the PDL. The cyclomatic complexity corresponding to the above flow graph is as follows,

1. flow graph has 6 regions
2. V(G) = 17 edges – 13 nodes + 2 = 6
3. V(G) = 5 predicate nodes + 1 = 6

Evaluate the Basis Set: Then evaluate the basis set of linearly independent paths. The cyclomatic complexity provides the upper bound on linearly independent paths. In this example six such paths are expected. They are as follows,

Path1: 1 – 2 – 10 – 11 – 13

Path2: 1 – 2 – 10 – 12 – 13

Path3: 1 – 2 – 3 – 4 – 5 – 8 – 9 – 2

Path4: 1 – 2 – 3 – 10 – 11 – 13

Path5: 1 – 2 – 3 – 4 – 5 – 6 – 8 – 9 – 2

Path6: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 2

Prepare the Test Cases: Select the data such that the conditions at predicate nodes are set properly for each path testing. The result of each test case is compared with the expected results. The execution of all the test cases ensures that all the statements in the program are executed atleast once.

Q43. Explain the concept of graph matrix.

Answer :

A Graph Matrix is a matrix with equal rows and columns. It is said to be a square matrix because its size is equal to the number of nodes on the flow graph. The rows and columns represent nodes while the matrix entries represent edges/ links between nodes. A graph matrix in general is a tabular representation of a flow graph and is a useful software tool in basis path testing. Consider a simple flow graph.

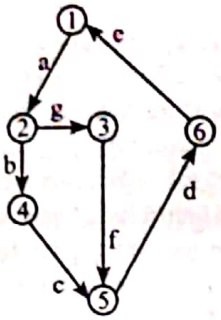


Figure: Flow Graph

Conventions to node are described below as follows,

NODES	1	2	3	4	5	6
1		a				
2			g	b		
3					f	
4					c	
5						d
6	e					

Figure: Graph Matrix for the Flow Graph

Each letter in the matrix relates to a link between two nodes. A node is identified by a number and an edge is identified by a letter.

A graph matrix can be used as a evaluation tool for program control structure during testing. Addition of a link weight to each matrix entry provides additional information about control flow. A link weight with value 1 indicates that there is connection while a value 0 indicates that there is no connection. In addition to these values, link weights can be provided with other properties like memory, resources and processing time necessary during testing. It also provides the probability for a link to be executed.

5.1.6 Control Structure Testing

Q44. Explain the following,

(a) Condition testing

(b) Data flow testing.

Answer :

(a) Condition Testing

This type of testing tests each and every condition in a program module and ensures that the program is free from errors. A condition is a boolean variable or a relational expression preceeded with a NOT(¬) operator. A relational expression is of the following form.

Exp1<relational-operator>Exp2

Here Exp1 and Exp2 are the arithmetic expressions and the relational operator may contain any of the following operators i.e <, ≤, >, ≥, =, # (A compound condition is a condition containing more than two simple conditions, boolean operators and parenthesis. It includes boolean operators like OR(|), AND(&) and NOT(¬)).

An incorrect condition states that atleast one part of the condition is incorrect. Possible types of errors in a condition include relational operator errors, arithmetic expression errors, Boolean variable errors, Boolean parenthesis errors and boolean operator errors like incorrect, missing or extra boolean operators.

(b) Data Flow Testing

This type of testing selects test paths based on the place where the definitions are defined and the uses of variables in the program consider that each statement in a program is allocated with a unique statement number and each function does not alter its parameters or global variables. The sets representing the definitions and used of variables are as follows,

def(S) = {A| statement S has a definition of A}

use(S) = {A | statement S has a use of A}

If statement S is assumed to be a if statement or loop statement then its def set will be empty and use set will include the elements based on the condition of statement S. For statement S, the definition of variable A is said to be "live" at statement S if there is a valid path from S to S with no other definition of A.

A Definition-Use (DU) chain of variable A is represented by [A, S, S'] with S and S' as the statement numbers and A contained in def(S) and use(S'). The definition of A in statement S is at to be "live" at statement S'. The strategy of DU testing states that every DV chain must be visited atleast conce during the testing. However this principle fails in programs containing if-then-else statements. This is because, the part does not contain any variable and there is no else part. Due to this, the else part containing the if statement is not covered in the test.

Q45. Explain Loop Testing in detail.

Answer :

Loop testing concentrates on the validity of loop constructs. It classifies four types of testing based on different loops. They are,

1. Testing simple loops
2. Testing nested loops
3. Testing concatenated loops
4. Testing unstructured loops.

1. **Testing Simple Loops:** Simple loops can be tested by using one of the following tests,

(i) Skip the complete loop.

(ii) Allow only one pass through the loop.

(iii) Allow only two passes through the loop.

(iv) Allow m passes through the loop provided m < n where n is the maximum number of passes that can be allowed through the loop.

(v) Allow n – 1, n, n + 1 passes through the loop.

2. **Testing Nested Loops:** Nested loops can be tested by performing the following sequence of steps.

(i) Begin the testing from the innermost loop and set the other loops to minimum values.

(ii) Perform simple loops tests on the innermost loop and set the outer loops at minimum loop and set the outer loops at minimum iteration values. Set the other tests to out-of-range or excluded values.

(iii) Move towards the next loop and perform test by keeping the outer loops at minimum values and nested loops to "typical" values.

(iv) Repeat these steps until the entire loops have been tested.

3. **Testing Concatenated Loops:** Concatenated loops can be tested using simple loop tests if the loops are independent otherwise nested loop tests are applied if the loops are dependent.

4. **Testing Unstructured Loops:** Unstructured loops are tested by applying either simple loop text, nested loop test or concatenated loop test based on the design of the loop. This type of loop must be redesigned to reflect the use of structured programming constructs. The process of redesigning must be carried out before the test commences.

5.1.7 O – O Testing Methods

Q46. Write short notes on the following,

(a) Steps to design a test case.

(b) Test-case design implications of OO concepts.

(c) Applicability of conventional test-case design methods.

Answer :

(a) Steps to Design a Test Case

The steps to design a test case are as follows,

1. Identify each test case distinctly and relate it with class for testing.
2. Specify the purpose of testing.
3. Develop a list of testing steps for every test. This list must consists the following details.
   (i) A record of defined states of the testing class.
   (ii) A record of messages and operations to be conducted as a result of test.
   (iii) A record of exceptions that are expected to occur due to class testing.
   (iv) A record of external conditions.
   (v) Additional information for understanding and implementing the test.

(b) Test-case Design Implications of OO Concepts

A class is developed by using the requirements and design models. The main aim of test-case design is to test classes. Since the attributes, operations are encapsulated, testing operations which are outside the class are not useful. The concrete and abstract behavior of an object is reported in testing process. However encapsulation generates obstacles and make it difficult to provide the object behavior.

Inheritance provides additional challenges at test-case design. Whenever new context is used, retesting has to be performed even after reuse has been achieved. In addition, testing has become complex due to the multiple inheritance which requires testing the number of contexts. If the subclasses and super classes are in same problem then a list of test cases are used which are inherited from the super class for testing the subclass. On the other hand if both the subclass and super class are used in different context then new test cases have to be defined.

(c) Applicability of Conventional Test-case Design Methods

The various applicability of conventional test case design methods are as follows,

(i) The white box testing methods are applicable to class.

(ii) Basis path, loop testing or data flow techniques are used to confirm that each and every statement is tested.

(iii) Black-box testing methods are applicable for OO systems.

Q47. Explain briefly about,

(i) Fault-based testing.

(ii) Test cases and the class hierarchy.

Answer :

(i) Fault-Based Testing

Fault-based testing is used to develop tests that detect the plausible faults. The test cases that are designed operate upon design or code. The tester finds the plausible faults by performing the tests beginning from analysis model.

This technique will be effective if the testers correctly recognize a possible fault. Analysis and design models assist is recognizing the faults with minimum effort.

Integration testing observes for possible faults in operation calls or message connections. It detects three type of faults in this context unexpected result, wrong operation or message used and incorrect invocation. The possible faults can be determined during the invocation of operations by examining the behavior of operation.

In considers the calling code for determining errors. Integration testing is used for both attributes and operations. It is possible to determine the behaviors of an object with the help of values assigned to attributes. Testing is performed to verify whether the different type of object behavior have correct values. The integration test is conducted to identify errors in client object but not at server.

(ii) Test Cases and the Class Hierarchy

The concept of inheritance complicates the testing and does not necessitate a complete testing of all derived classes. Consider the two classes, a class X that consists of operations Y( ) and Z( ) and another class A that redefines Z( ) to be used in local context. This imposes a problem of whether to test the A :: Z( ) as it depicts the new design and code and whether to retest A :: Y( ) when A :: Y( ) calls Z( ) then the behavior of Z( ) is changed and this new behavior can be mishandled. Hence, it requires new tests and the test for A :: Y( ) is essential when the design and code of Y( ) is not depending on Z( ) then retest is not required for the code in the A class.

The two distinct operations X :: Z( ) and A : Z have dissimilar specifications and implementations. Each operation should have a list of test requirements that are derived from specification and implementation. The test requirements investigate for the plausible faults like integration faults, condition faults, boundary faults and so on. Those faults have similar operations and their list of test requirements overlaps. If the 00 design is better then the overlap is greater. New tests are required for A :: Z ( ) requirements as they are not satisfied by X :: Z ( ) tests. Therefore, the X :: Z is tests are used for the class A objects. The test inputs are suitable for X and A class but the expected output varies in the A class.

Q48. Explain in detail about Scenario Based Test Design.

Answer :

Scenario-Based Test Design

Scenario-based testing focuses on the user's action but not on product's action. That is the tasks which to be performed by the users are identified and applied on their variants as tests.

Scenarios detect the interaction errors by using the test cases which are more difficult and more practical than fault-based tests.

Considering the example of the design of scenario-based tests for a text editor. Following are the use cases that are reviewed in this example.

Use Case: Establishing the final draft.

Background

It is of no significance to print the final draft, reading it and detect some errors which are not seen on on-screen image. By using a use case, it is possible to define a series/flow of events that occur when this event happens.

1. Printing the complete document.
2. Modifying the pages in the document whenever required.
3. Printing each page after modification.
4. Printing a sequence of pages whenever required.

The above scenario defines a test specific user requirements. The user requirements are clear and are as follows,

(i) A technique for printing individual pages and

(ii) A technique for printing number of pages.

When the test begins, it is necessary to test the editing after printing. An effort is required to design to detect the errors in the editing function due to printing function. Hence the errors represents that two software functions are improperly independent.

Use Case

Printing a new copy.

Background

The steps to print a new copy of the document is as follows,

1. Initially open the document.
2. Then, print the document.
3. Finally, close the document.

In new editors, the printing format is selected/used once and it will be same for next printing. After the completion of "establishing the final draft" scenario, the print option is selected by clicking on the print button present in the dialog box. This will print the last printed page. Hence, the proper scenario to perform this task is as follows,

Use Case: Printing a new copy.

1. Initially open the document.
2. Then select the print option from the menu.
3. Next check for the range of pages to be printed.
4. Then, click the print button.
5. Finally, close the document.

Unfortunately, this scenario depicts a specification error. The editor is not performing the way as the user expects. Users mainly check for range of pages to be printed. After selecting the print option for range of required number of pages. By an error, the pages that have been printed were less than the given.

Q49. Explain how surface structure and deep structure is tested.

Answer :

Surface Structure: Surface structure is defined as the structure which can be seen from outside of an 00 program i.e the structure is clearly visible to the end user. Instead of performing functions, the users are provided with objects that are manipulated. The tests are conducted based on user tasks and are captured by understanding, watching and talking with the main users / customers.

Consider an example of conventional system that uses command oriented interface in which a user utilizes the set of commands for testing the checklist. If there exist no test scenario to perform the command then the test is done on user tasks. Where as in object-based interface, the tester utilizes the set of objects as a checklist during testing.

The best tests can be produced by examining the system in a new way. For example, a command-based interface system will generate through tests if the operations are considered to be independent of objects.

Deep Structure: Deep structure is defined as the inside technical information of an 00 program and code. Deep structure can be tested for dependence, behavior and communication mechanisms of design model for 00 software. It uses requirements and design models for this purpose.

Consider an example of UML collaboration diagram or deployment model that indicates the collaborations among objects and subsystem which are not clear externally. The test case addresses the issue asking whether a task performs collaboration on the collaboration diagram.

5.2 DEBUGGING : DEBUGGING TECHNIQUES, THE ART OF DEBUGGING

Q50. With the help of a diagram, explain the process of debugging.

Model Paper-I, Q17(b)

Answer :

Debugging

Once testing is successfully carried out and bugs are uncovered, the next step is the removal of these bugs and is named as "debugging". The process of debugging is depicted below.

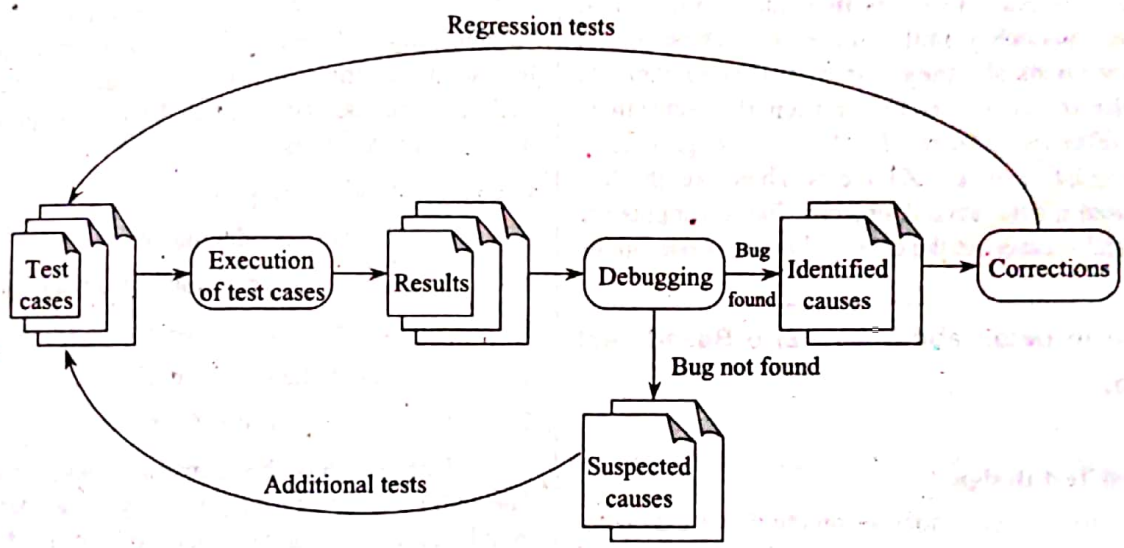


Figure: Debugging Process

The test cases are executed and the results are gathered. These results are compared with the expected results. Any variation between the two means 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.

Q51. State and explain various debugging tactics.

Answer :

While dealing with the debugging process, it has to be noted that the main aim behind the debugging process is to find the cause and accordingly eliminate the error. To do this, we often resort to several systematic approaches or by just suspecting (i.e., the error may be present at certain location) or sometimes it all depends on the luck or the combination of all these three strategies. We do not have any direct method which can take us to the cause of error however 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:** When the software engineer gets fed-up with all his attempts in finding the cause of an error, he usually resorts to The Brute Force approach. This is an approach which 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 to be effective only when applied to smaller programs. In this mechanism we initially look for the symptoms of errors and from there we start moving backwards manually until we find the cause of error. However, as number of source lines increases, managing backward paths becomes tedious.

(iii) **Cause Elimination Process:** This process begins by organizing those parts of code which includes 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 we will be left out with real causes of errors, which are later 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.

The Ultimate Resort: Be it any debugging strategy, not more than one hour should be spent on debugging. And, if still there is no outcome within one hour, then seeking help from others is a better option.

Q52. Discuss about software tools for debugging.

Answer :

Software tools for debugging give automated assistance in debugging the problems of software. The purpose of debugging tools 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 bug reports.

(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.

5.3 PRODUCT METRICS

5.3.1 A Framework for Product Metric

Q53. Define the following,
(i) Measures
(ii) Metrics
(iii) Indicators.

Answer :

(i) Measures

A measure is defined as a quantitative indication that gives description about the size, amount, extent or capacity of a certain attribute related to a process/product. 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 usually is 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 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.

(iii) Indicators

An indicator is defined as a metric or a collection of metrics by 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 an efficient way. Thus, indicators are evaluated by collecting measures and developing metrics by a software engineer.

Q54. What is the challenge of product metrics?

Answer :

The challenge of product metrics is to develop a complexity metric, which gives the complete measure about the complexity of software. Though, several complexity measures have been developed, every measure is different from one another in both system complexity as well as system attributes. Consider an example in which a metric for evaluating 'an attractive sports bike' is considered. In such evaluation, the observers consider characteristics like the body design, cost, fuel economy, performance or other mechanical features. Since, these characteristics differ fro one another, the evaluation of a single metric value for attractiveness of sports bike is complicated such problem arises with the software systems. But, if is however, essential to measure and control the complexity of software.

As it is difficult to evaluate a single metric value, collect measures related to several internal attributes of a software program. Such collected measures can be treated as individual indicators that provide information about the quality of design and analysis models. This may result in certain problems. Fenton noticed this when he declared the following statement, "Even though, there is a difficulty in determining measures (that differentiates several distinct attributes), the measures must fulfill the conflicting aims/goals". This acts as a solution to the measurement theory. Even if the Fenton's statement is valid, several people raise question regarding the product measurement. "Can the product measurements developed over the past few years provide an objective and consistent strategy for evaluating quality of design and analysis models to the software engineering".

Another question arises about the product metrics i.e., how they are related to reliability and quality of computer system. According to Fenton, the answer to the question is. Though, it is assumed that intuitive connections exist among internal/external design of product metrics and process attributes, there have been actually certain relationships that are conducted by some scientific experiments. Consequently, it is determined that measurement plays an important role inorder to attain quality of analysis and design models.

Q55. Explain briefly about the goal-oriented software measurement.

Answer :

Goal-Oriented Software Measurement

Meaningful metrics for any component of software process can be determined by a method called GQM (Goal Question Metric). This method is entirely focused so as to perform the following,

1. To define a clear goal of measurement such goal must be related to a product characteristic/process activity that is to be evaluated.
2. To define a group of queries. The answers to such queries must be provided so as to atain the desired goal.
3. To determine metrics which are efficiently formulated. Such metrics can then be used to provide answers to those queries.

Every individual measurement goal can be defined by means of a goal definition template. The representation of such template is as follows,

Analyze the attribute/activity name that is to be evaluated *for the purpose* of achieving the entire goal of analysis *with respect to* the characteristics of the specified attribute/activity *from the viewpoint* of team/users who perform measurement *in the context of* the location in which measurement is being performed.

Now, consider an example called 'SafeHome' Here the goal definition template for SafeHome is as follows,

Analyze the software architecture for 'SafeHome' For *the purpose* of measuring components of software architecture *with respect to* the notion of developing an extensible SafeHome *from the viewpoint* of the software developers team who measure the attributes/components. *In the context* of the environment where ideas regarding the improvements about products are considered over the next 3 years.

In order to successfully fulfill the goal of measurement, the software team must provides answers to queries that arise when a clear definition about the goal is given (by GQM). Some of the queries are as follows,

1. The complexity of software components defined in such away that modifications and extensions can be done to them.
2. Every software architecture component has its specific data and function.

Thus, answer to these queries can be provided by means of at least one metric and measure.

Q56. How should we assess the quality of a proposed software metric? Also, describe the important product metrics areas.

Answer :

Quality of a Proposed Software Metric

The quality of a proposed software metric can be assessed based on the following attributes,

1. **Easy and Efficient:** It must be easy to know about the way of deriving a metric. Also, it must not consume excess time and effort for computation.
2. **Clear and Consistent:** The results of the metrics must be in a clear and consistent way.
3. **Independent of Programming Language:** The derivation of metrics must consider program structure, analysis model or design model.
4. **Intuitively Persuasive:** The intutitive notions of the engineers regarding the products attributes must be fulfilled by the metric.

5. **Uniform use of Dimensions:** The measures that does not generate eccentric combinations of unit/dimensions must be used for mathematical computation of metric.
6. **Efficient Mechanism:** The metric must produce a high quality final product software by applying an efficient and effective mechanism.

Product Metrics Areas

The important product metrics areas are,

1. **Metrics for Analysis Model:** The metrics for analysis model are used to provide information about several different characteristics/aspects of analysis model. Such metrics comprise the following,
   (i) **Size of System:** This metric evaluates the complete system size in the form of data, which are represented in terms of components of analysis models.
   (ii) **Quality of Software Specification:** This metric defines the clarity and totality of requirements specification.
   (iii) **Delivery of Software Functionality:** This metric gains an implicit measure of functionality packaged inside the software.
2. **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.
   (iv) **Metric of Software Architecture:** This metric is used to define the quality of software architectural design.
3. **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.
4. **Metrics for Testing:** The metrics for testing are used to efficiently design the test cases as well as to perform testing of source code. Such metrics comprise the following,
   (i) **Metric of Software Bugs:** This metric is used to determine bugs existing in the software code.
   (ii) **Metric of Testing:** This metric is used to measure the effectiveness of tests that are being performed.
   (iii) **Metric of Software Process:** This metric is evaluated when testing is being performed.
   (iv) **Metric of Statement and Branch Coverage:** This metric is used to design the test cases, which gives complete statement and branch coverage.

5.3.2 Metrics For Each Phase Of Software Development

Q57. Discuss in detail on function-based metrics.

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,

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.

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.

External Inquiries (EQs): The inquiries that are generated on-line and whose response is also given on-line are called external inquiries.

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.

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,

$$FP = Total \times \left[(0.65) + 0.01 \times \sum_{i=1}^{14} f_i \right]$$

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 include 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.

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.

Q58. Compute the function point value for a project with the following information domain characteristics,

Number of external inputs: 32

Number of external outputs: 60

Number of external inquires: 24

Number of external interface files: 2

Number of internal logical files: 8

Assume that all complexity adjustment values are average.

Answer :

The following is the relationship which is used to compute the Function Points (FP).

$$FP = \text{Count total} \times [(0.65) + 0.01 \times \sum_{i=1}^{14} (f_i)]$$

Count total is obtained by adding all FP entries. ... (1)

f_i = Value adjustment factors

Since the average complexity is assumed for the adjustment factors, $f_i = 3$

$$\therefore \quad \sum_{i=1}^{14} f_i = 14 \times 3 = 42 \quad \text{... (2)}$$

Also given,

External inputs = 32

External outputs = 60

External inquiries = 24

External interface files = 2

Internal logical files = 8

Given adjustment values are "*average*"

Calculation of count total:

Functional units	Simple	Average	Complex	Count	Average × Count
External input	3	4	6	32	128
External output	4	5	7	60	300
External inquiries	3	4	6	24	96
External interface files	5	7	10	2	14
Internal logical files	7	10	15	8	80
Count total					**618**

Calculation of Function Point Value

Substituting the values of count total and $\sum_{i=1}^{14} f_i$ in equation (1), we get,

$$FP = 618 \times [0.65 + 0.01 \times 42]$$
$$= 618 \times [0.65 + 0.42]$$
$$= 618 \times 1.07$$
$$= 619.07$$

Q59. 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 those 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) and is given by.

$$M_C = N_{UF} / (N_S \times N_I)$$

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,

$$M'_C = N_V / N_V + N_{nV}$$

Q60. Discuss in detail the architectural design metrics.

Answer :

Metrics for Architectural Design: Architectural design metrics focuses 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,

$$D_{com}(n) = \frac{var(n)}{f_{out}(n)+1}$$

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,

$$S_{com}(n) = f^2_{out}(n)$$

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 a 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,

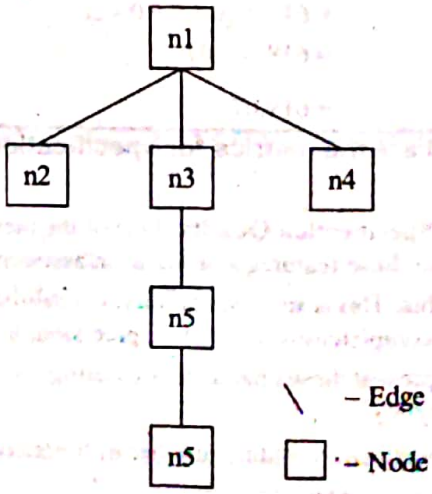


Figure: An Example Architecture

Size = 5 + 6

= 11 (Number of arcs + Number of nodes)

Width = 3 (Longest horizontal path) i.e., $n_1 - n_2 - n_4 - n_6$

Depth = 4 (Longest path from root node to leaf)

$(n_1 - n_2 - n_5 - n_6)$

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,

$$DSQI = \sum_{i=1}^{6} w_i y_i$$

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, $w_i = 0.167$.

Q61. Why do we need metrics for design model? Describe in detail the architectural design metrics.

Answer :

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 helps the designer to use the mechanisms of object oriented paradigm (i.e., inheritance, encapsulation, polymorphism) in a convenient and understandable manner so as to enhance 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.

Architectural Design Metrics

For answer refer Unit-V, Q60.

Q62. 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 $(X_4) = (220 \times 3) = 66$

(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,

$$\boxed{D_6 = 1 - \frac{X_7}{X_1}}$$

$$= 1 - \frac{600}{1140}$$

$$= 1 - \frac{10}{19} = \frac{9}{19}$$

$$= 0.474$$

DSQI (Discrete structure Quality Index) is calculated as follows,

$$\boxed{\text{DSQI} = \sum_{i=1}^{6} w_i D_i}$$

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

$\boxed{\text{DSQI} = 4.612}$

Q63. Discuss the nine distinct and measurable characteristics of Object Oriented design.

Answer :

For the purpose of defining metrics for an OO system, 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 we usually resort to following elements,
   (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 turn 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.

Q64. Discuss the class-oriented metrics, the CK metrics suite.

Answer :

Class-oriented metrics has 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.

Basing 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, we can refer them 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 properites 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, we cannot 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_1, COM_2, 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 in turn, the software reusability feature gets decreased.

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.

Q65. 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 the is method inheritance factor. Mathematically MIF is given by,

$$MIF = \frac{\sum_{i=1}^{n} M_i(c_i)}{\sum_{i=1}^{n} M_a(c_i)}$$

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,

$$CF = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} rel\_bet\_clnt\_Srvr\_class(c_i - c_j)}{(n^2 - n)}$$

Where,

$$rel\_bet\_clnt\_Srvr\_class \begin{cases} = 1, & \text{if the client class } c_c \text{ and server class } c_s \text{ are related and } c_c \neq c_s \\ 0, & \text{Otherwise} \end{cases}$$

The higher the value of CF, the higher is the software complexity and lower is the software reusability, maintainability and understandability.

Q66. Write short notes on,

(a) Component level design metrics.

(b) Operation Oriented metrics.

Answer :

(a) Component Level Design Metrics

Component level design metrics are used as soon as a procedural design is available. These are,

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:** A set of data tokens being present in one or more data slices.

(iv) **Superglue Tokens:** A 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.

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,

Metrics for Global Coupling

G_d = Count of global variables being used for data

G_c = Count of global variables being used for control.

Metrics for Environmental Coupling

W = Count of modules invoked by the module being considered.

R = Count of modules invoking the module being considered.

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,

$$C = 1 - U_{cop}$$

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) Operation Oriented Metrics

Lorenz and Kidd proposed the following metrics in favour of operation oriented metrics. These metrics considers 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:** Usually operation size is 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:** For measuring operation complexity we can resort to 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.

Q67. 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.

Q68. Write short notes on metrics for source code.

Answer :

Metrics for Source Code

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, we come across a new ratio 'V_r' referred as volume ratio 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,

$$V_r = \frac{2}{n_1} \times \frac{n_2}{T_2}$$

Q69. Discuss the metrics for testing.

Answer :

Whenever we consider the metrics for testing, it has to be noted that they (metrics) are derived by considering the implementation of process of testing. Hence, in this case, the testers usually resorts to metrics laid for analysis, design and code phases, which 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 effort 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

$P_{(m)}$ = Percentage of testing effort required for module 'm'.

Metrics Associated with Object Oriented Testing: These metrics helps 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 needs to be tested (checking the side-effects generated by the methods).

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.

Q70. What is the maintenance metric suggested by IEEE?

Answer :

Metrics for Maintenance

Even though the software metrics can be used as metrics for maintenance, IEEE came out with 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.

5.4 SOFTWARE QUALITY : DEFINITION

Q71. What is meant by software quality? Discuss clearly the McCall's software quality factors.

Answer :

Model Paper-II, Q17(b)

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.

(a) **McCall's Quality Factors:** Following figure depicts the McCall's software quality factors.

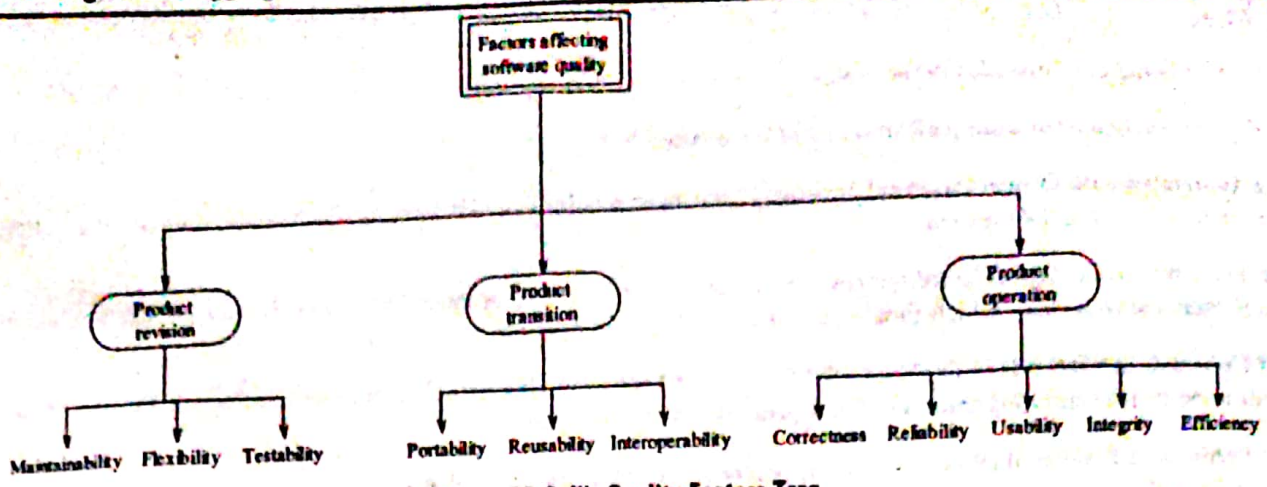


Figure: McCall's Quality Factors Tree

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:** Under this category those factors are grouped which may exert their impact while the product is under operation. 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:** Under this category those factors are grouped which may exert their impact while the product is being moved from one environment to another. 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:** Under this category those factors are grouped which may exert their impact while the product undergoes changes. 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.

Q72. What is software quality? Explain the determinants of software quality in detail.

Answer :

Software Quality

For answer refer Unit-V, Q71, Topic: Software Quality.

Determinants of Software Quality: A process is one of the controllable factors that improves the software quality. There are three determinant 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 the three determinants of the software quality.

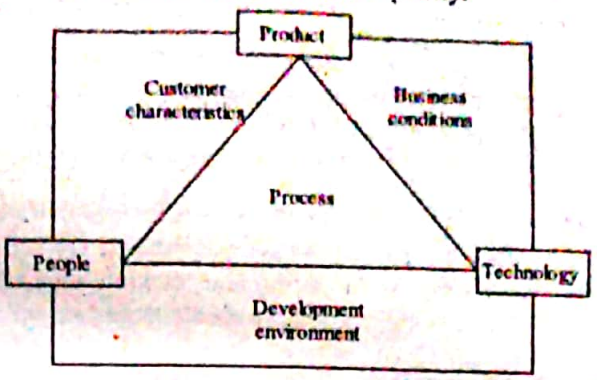


Figure: Determinants of Software Quality

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.

Such as deadlines and business rules and customer characteristics such as case of communication and collaboration.

Q73. Discuss about ISO 9126 quality factors.

Answer :

ISO 9126 Quality Factors: ISO has defined six quality factors for a given software as ISO 9126 software quality factors. These factors can be used as a basis for indirect measures and not direct measures. Moreover, the quality of a system can be assessed by using these 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:** 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:** 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 time span during which the given software can be used. Hence, recoverability, maturity, fault and tolerance can be termed as the sub-attributes of reliability.

5.4.1 Quality Assurance : Basic Elements

Q74. What is meant by SQA? Discuss in detail SQA activities.

Answer :

SQA

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 so that the management can be focused to the product's quality. Hence, any dissatisfaction reported in this process is usually dealt by the quality managers.

Software quality assurance is a critical component in software engineering. It consists of various tasks that are related to two constituencies,

(a) Software engineers performs 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, conducts, format technical reviews and performs 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 and 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 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 choose 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:** From a set of work products SQA group reviews few selected products and any deviations from the software process 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.

Q75. 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 majorly of two types.

There are two types of standards applied by quality assurance. They are,

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.

Q76. Is it possible to assess the quality of software if the customer 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 the 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 a little bit 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 them 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.

Q77. Why is there often tension between a software engineering group and independent software quality assurance group? Is this healthy?

Answer :

Software Engineering (SE) group mainly deals with the software quality by utilizing the technical work, conducting formal technical reviews and accomplishing the software testing with a proper plan whereas SQA group mainly deals with identification of problems i.e., bugs, errors or defects occurred in software quality. It is also responsible for analyzing, reporting, record keeping, quality assurance planning etc. Besides this, SQA group helps the SE group in order to produce a high quality end product. Both groups perform quality assurance activities. These are directed according to a plan developed during project. The plan performs the following,

1. Keep tracks of the documents produced by SQA group.
2. Reviews and auditing
3. Project estimation
4. Checking whether project supports the standards or not.
5. Provides feedback to software engineering team.

The work flow between SE group and SQA group is carried out as follows,

Firstly, software engineering group selects a specific process for their technical work. Basically, a software engineer analyzes models, creates a computer program and finally reports it. The intent of SE group is to show that the developed software works in an efficient manner as per the customers requirements. Secondly, SQA group checks the selected process and reviews it by identifying and keep tracking the deviations arised during the process. Later, it reports the final results of SE group to the project manager. Then project manager informs the erroneous results to SE group. By this moment, conflict arises among two groups with respect to their tasks. From the view point of SQA group, the software quality work carried out by SE group is destructive. SE group assures that the program works effectively, but some uncovered errors exist which are identified by the SQA group. This tension is considered as healthy enough because, one can produce high quality end product due to the feedback of SQA group, the quality of software will be refined.

Hence, for each and every software project, conflict exists between software engineering group and SQA group in order to produce a good quality software product.

5.4.2 Formal Approaches

Q78. Illustrate the formal approaches to SQA.

Answer :

Software quality assurance is planned and systematic approach to evaluate the quality of and adherence to the software product standards, procedures and processors. It assures that standards and procedures are established and followed in lifestyle of software acquisition. The SQA evaluation of a product is possible to be accomplished related to applicable standards. The SQA consists of activities which are applied in software software process. It confines to the following,

1. A quality management approach
2. An effective software engineering technology
3. Formal technical reviews applied in to software process
4. A multi-tiered testing strategy
5. Control of software documentation along with modifications
6. Methods to ensure compliance with software development standards
7. Measurements and reporting mechanisms.

In last two decades, a small community of software engineering has argued that a more formal approach to software quality assurance is needed. A computer program is nothing but a mathematical object. A rigorous syntax and semantics are defined for each of the programming language and similar rigorous approach to specification of software requirement is available. The mathematical proofs of correctness can be applied to represent that program conforms to specification after the requirements model is represented in rigorous manner. There are a number of approaches proposed to formal proof of correctness.

The two formal approaches to SQA are depicted as follows,

1. Six Sigma Quality Assurance Approach

This approach has set up a benchmark for excellence. It does this by raising standards of quality management approach. The quality system suggested by it lead to a buzz in industries such as healthcare, retail, BPO etc. Such type of methods are developed with the purpose of testing products and services to meet criteria of desired standard and customer's expectations related to the products are met successfully. The six sigma quality management became one of the prime concern for organizations throughout the world with ever increasing demand of quality products.

2. Six Sigma Green Belt Approach

This approach is all about assuring the quality is production as well as about promising and delivering the quality. The promising quality reveals about assurance to vital customers about best quality of products and services. It is mandatory to adapt to industry approved testing techniques in order to deliver the quality consistently. All such aspects must be taken care by quality assurance officer.

5.4.3 Statistical Software Quality Assurance

Q79. Discuss clearly the statistical SQA with sample example.

Answer :

Statistical SQA

Following are few important steps favouring statistical quality assurance in case of software,

Step 1 : It initially begins with collection of information on defects occurring in a given software and categorizing them.

Step 2 : Cause of each defect is recognized.

Step 3 : 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.

Step 4 : 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 here 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 one which were encountered during software development as well as the one 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 found a table indicating the causes and their percentages of errors introduced by them. 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.

Q80. 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 suggest following three steps which are also referred as core steps.

Step 1 : Initially gather the data related to customer requirements, their deliverable and finally estimate the required project goals in most sophisticated manner.

Step 2 : With an objective of obtaining the quality performance, compute the current proceedings and their outputs.

Step 3 : Obtain the vital causes of defects by closely analyzing defect metrics.

The above mentioned steps are referred as core steps. But to further improve our current proceedings, following two steps are proved to be effective.

Step 4 : Improving the current process by removing major causes of bugs.

Step 5 : Controlling over the current process such that any changes made does 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.

Steps 6 : Designing the process by considering customer requirements and curbing the core causes of defects.

Step 7 : 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.

5.4.4 Software Reliability

Q81. What is meant by software reliability? Discuss the measures of it.

Answer :

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, a 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 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. Availability = $\left(\frac{MTTF}{MTTF+MTTR}\right)\times 100\%$.	Continuously running systems such as telephone switching systems.

Table: Software Reliability and Availability Measures

Q82. Discuss in detail about software safety.

Answer :

Software Safety

Any organization before delivering a product definitely worries about safety. Software safety is one of the major issues which attributes to software quality standards. Hence, software safety deals with the mechanisms to make software secure from most probably occurring hazards.

Identification of Hazards: Identification of hazards forms one of the major aspects of software safety. Early identification of hazards leads to development of effective mechanisms, which avoids these hazards to greater extent or can control the software functionality wherever they become inevitable. Hence, these hazards must be identified, and categorized either as risk or criticality.

Analysis of Hazards: Hazard analysis techniques are implemented so as to determine the strength (i.e., to what extent the given hazard can have its impact on the software) and also its probability of occurrence. In this regard the staff responsible for doing so usually resort to petri net models, real time logic, fault tree analysis etc.

Safety Related Requirements: As soon as the hazards are determined and analyzed a list giving the undesirable events and the way system responds to them can be specified.

Once a hazard is identified and analyzed requirements for safety can be specified. These specifications describe the outcomes during the hazards. Eventually, the safety measures included in the software to react during such situations are then demonstrated. Hence, any inconsistencies observed during its demonstration, is noted and rectified later.

5.4.5 ISO 9000 Quality Standards

Q83. Discuss about ISO 9000 quality standards.

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 improvement
5. Carryout 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 required with 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.

Q84. Discuss about software quality management tools.

Answer :

Software Quality Management Tools: The main purpose of providing software quality management tools is to help the project team in evaluating and enhancing the quality of software product.

Some of these tools are,

1. **ARM Tool (Automated Requirements Measurement):** ARM tool was developed by NASA. It is considered as initial software life cycle tool. The main purpose of this tool is to provide the measures availed by project managers, for assessing the quality of software requirements specification document. Aim is to list out the requirements in a right way such that it can be applied to any set of requirement specification.
2. **QPR (Quality Processes Results):** It is developed by QPR software acts as platform for six sigma strategy and other quality management strategies,
3. **Quality Tools Cookbook:** It provides information about following classic quality management tools are,
   (a) **Histogram:** Generally, a histogram is a type of bar chart in which many data points is grouped in the form of a class. So, it depicts the data variations of each class occurring in the data set. In quality control, histogram of a process should be reviewed in order to acquire knowledge about the specific process. These are also called easy to read charts.
   (b) **Control Charts:** Control charts are otherwise named as Shewart charts. Using this, both special cause and common cause variations can be monitored in a process. For instance, in special cause variations occurs, when anybody pushes you while doing some work and common cause variation occurs when the pitches of baseball pitcher are moving as per his wish.
   (c) **Strategic Planning:** Strategic planning is a systematic approach developed to achieve the objectives of business with respect to the actions and resources set for it.
   (d) **Benchmarking:** Benchmarking is a performance appraisal technique. In this, product performance is compared in two ways,
      (i) Externally with competitors and top class companies.
      (ii) Internally within their own organization.

      The main purpose is to understand the processes and practices, then implementing these thereby improving the companies performance.
   (e) **Affinity Diagram:** An affinity diagram is a refinement of brainstorming engineering. This is used when,
      (i) Ideas are to be carried out in an organized manner
      (ii) Pre-existing ideas are to be overcomed
      (iii) Ideas are to be made clear and
      (iv) Integration is to be created within a team.

The fatal quality management tools assist the software organizations in order to detect, analyze, examine and manage the quality of their products.

5.4.6 SQA Plan

Q85. Write in brief about SQA plan.

Answer :

The SQA plan demonstrates a road map like structure to establish software quality assurance. The plain is developed by SQA group and project team. It is in the form of template for SQA activities which are provided for energy software project. The outline for SQA plans as recommended by IEEE are depicted as follows,

1. Primary purpose of plan
2. References
3. Management
   - 3.1 Organization
   - 3.2 Taks
   - 3.3 Responsibilities
4. Documentation
   - 4.1 Purpose
   - 4.2 Required software engineering documents
   - 4.3 Other documents
5. Standards, practices and conventions
   - 5.1 Purpose
   - 5.2 Conventions
6. Reviews and Audits
   - 6.1 Purpose
   - 6.2 Review requirements
     - (i) Software requirements review
     - (ii) Design reviews
     - (iii) Software verification and validation reviews
     - (iv) Functional audit
     - (v) Physical audit
     - (vi) In-process audits
     - (vii) Management reviews
7. Test
8. Problem reporting and corrective action
9. Tools, Techniques and Methodologies
10. Code control
11. Media control
12. Supplier control
13. Records, collection, Maintenance and retention
14. Training
15. Risk Management

The fist section of plans depicts the purpose and scope of document as well as the process activities which are covered by quality assurance. The management section represents the position of SQA's plan in organizational structure. The Documentation section represents the work products generated as an element of software process. They can be either project documents, models, technical documents and user documents. The next section standards, practices and conventions represents the standards/practices applicable while software process. The reviews and audits section represents the reviews and audits that are to be conducted by software engineering team, SQA Group and customer. The test section represents the software test plan and procedure. In addition to this it determines the test record - keeping requirement. The problem reporting and corrective actions determines the procedures for reporting, tracking and also resolving errors and defects. It also locates the organizational responsibilities for them. The remaining plans represent the tools and methods supporting SQA activities and tasks, establishes methods to assemble, safeguard and maintain the records, defines contract management approach, reference software configuration management methods to control the change. In addition to these they also define methods to identify, asses, monitor and control the tasks.

IMPORTANT QUESTIONS

UNIT-I

SHORT QUESTIONS

Q1. What is Software Development Life Cycle? *(Refer Unit-I, Q2)*

Q2. Define the term software and software engineering. *(Refer Unit-I, Q3)*

Q3. Explain software crisis. *(Refer Unit-I, Q5)*

Q4. Distinguish between software products and software services. *(Refer Unit-I, Q6)*

Q5. List and define the various software myths. *(Refer Unit-I, Q9)*

Q6. What is waterfall model? *(Refer Unit-I, Q14)*

Q7. List evolutionary process models. *(Refer Unit-I, Q19)*

Q8. List the task regions in the spiral model. *(Refer Unit-I, Q20)*

ESSAY QUESTIONS

Q9. What do you mean by software engineering? Explain the software engineering layers. *(Refer Unit-I, Q27)*

Q10. What is CMM? Discuss how various maturity levels of CMM can be measured. *(Refer Unit-I, Q31)*

Q11. What is waterfall model? How is it different from other engineering process models? *(Refer Unit-I, Q37)*

Q12. 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? *(Refer Unit-I, Q42)*

Q13. Give an overview of different phases of unified process model. *(Refer Unit-I, Q51)*

Q14. Briefly write about Agility. *(Refer Unit-I, Q55)*

UNIT-II

SHORT QUESTIONS

Q1. List out any five principles of software engineering. *(Refer Unit-II, Q1)*

Q2. What is deployment? *(Refer Unit-II, Q3)*

Q3. What is system modeling? *(Refer Unit-II, Q4)*

Q4. What is the purpose of business process engineering. *(Refer Unit-II, Q5)*

Q5. Define scenario. *(Refer Unit-II, Q7)*

Q6. Write a short note on data objects. *(Refer Unit-II, Q8)*

ESSAY QUESTIONS

Q7. Briefly explain the principles of software engineering. *(Refer Unit-II, Q11)*

Q8. Illustrate the hlerarchy of system engineering. *(Refer Unit-II, Q18)*

Q9. Write in short about business process engineering. *(Refer Unit-II, Q19)*

Q10. Explain about product engineering. *(Refer Unit-II, Q21)*

Q11. What is system modeling? Explain the process of creating models and the factors that should be considered when building models. *(Refer Unit-II, Q22)*

Q12. Briefly explain requirements engineering tasks. *(Refer Unit-II, Q30)*

Q13. Discuss various steps in requirements engineering. What are the work products of engineering the requirements? *(Refer Unit-II, Q31)*

Q14. Explain about building the analysis model. *(Refer Unit-II, Q44)*

UNIT-III

SHORT QUESTIONS

Q1. How do analysis classes manifest themselves as elements of solution space? *(Refer Unit-III, Q1)*

Q2. Write short notes on analysis pattern. *(Refer Unit-III, Q3)*

Q3. What is a stereotype? *(Refer Unit-III, Q4)*

Q4. Define functional Independence. *(Refer Unit-III, Q6)*

Q5. What is the intent of information hiding? *(Refer Unit-III, Q7)*

Q6. Distinguish between classes and components. *(Refer Unit-III, Q8)*

ESSAY QUESTIONS

Q7. Discuss in detail about requirement analysis. *(Refer Unit-III, Q9)*

Q8. Explain in detail the different concepts associated with data modeling. *(Refer Unit-III, Q10)*

Q9. What is object oriented analysis? Discuss the four elements of object model. *(Refer Unit-III, Q11)*

Q10. Discuss in detail about flow-oriented modeling. *(Refer Unit-III, Q16)*

Q11. What is software design? Discuss where exactly design comes in software engineering. *(Refer Unit-III, Q28)*

Q12. State and explain various software design concepts. *(Refer Unit-III, Q34)*

Q13. How to translate the analysis model into the design model? Explain with an example scenario. *(Refer Unit-III, Q40)*

Q14. Discuss about pattern based software design in detail. *(Refer Unit-III, Q45)*

UNIT-IV

SHORT QUESTIONS

Q1. Differentiate between control and stamp coupling. *(Refer Unit-IV, Q1)*

Q2. Write short notes on architectural patterns. *(Refer Unit-IV, Q2)*

Q3. Elaborate on "REP" in detail. *(Refer Unit-IV, Q3)*

Q4. What is software component? Differentiate between hardware and software components. *(Refer Unit-IV, Q4)*

Q5. Write about software documentation. *(Refer Unit-IV, Q8)*

Q6. What is user interface analysis? *(Refer Unit-IV, Q9)*

ESSAY QUESTIONS

Q7. What is software architecture? Why is it important? *(Refer Unit-IV, Q11)*

Q8. What is meant by data design? Explain with an example. *(Refer Unit-IV, Q14)*

Q9. What is architectural style? Discuss various categories of it. *(Refer Unit-IV, Q16)*

Q10. What is coupling? Explain various types of coupling in detail. *(Refer Unit-IV, Q27)*

Q11. Explain about user interface analysis and user analysis. *(Refer Unit-IV, Q37)*

Q12. Elaborate on various interface design steps. *(Refer Unit-IV, Q42)*

Q13. With the help of a diagram explain in detail the interface design evaluation cycle. *(Refer Unit-IV, Q43)*

UNIT-V

SHORT QUESTIONS

Q1. What is meant by software testing? And list its characteristics. *(Refer Unit-V, Q1)*

Q2. Define Testing. *(Refer Unit-V, Q3)*

Q3. Define unit testing and top-down integration testing. *(Refer Unit-V, Q5)*

Q4. Define black box testing strategy. *(Refer Unit-V, Q6)*

Q5. Define metrics. *(Refer Unit-V, Q7)*

Q6. Discuss the importance of quality assurance. *(Refer Unit-V, Q10)*

Q7. Define quality of conformance. *(Refer Unit-V, Q12)*

Q8. What are the attributes of a good test? *(Refer Unit-V, Q13)*

ESSAY QUESTIONS

Q9. What is software Testing? Explain test Characteristics. *(Refer Unit-V, Q14)*

Q10. Discuss in detail about integration testing. What are its types? Explain the big bang approach. *(Refer Unit-V, Q20)*

Q11. What are the testing strategies for conventional software? Explain them in detail. *(Refer Unit-V, Q26)*

Q12. What questions do black-box tests answer? *(Refer Unit-V, Q31)*

Q13. Give an overview of white-box testing techniques with help of flow graph. *(Refer Unit-V, Q34)*

Q14. With the help of a diagram, explain the process of debugging. *(Refer Unit-V, Q50)*

Q15. Discuss the metrics for testing. *(Refer Unit-V, Q69)*

Q16. What is meant by software quality? Discuss clearly the McCall's software quality factors. *(Refer Unit-V, Q71)*

Q17. What is meant by SQA? Discuss in detail SQA activities. *(Refer Unit-V, Q74)*

Q18. Illustrate the formal approaches to SQA. *(Refer Unit-V, Q78)*

Q19. Discuss about ISO 9000 quality standards. *(Refer Unit-V, Q83)*

FACULTY OF ENGINEERING

B.E. V-Semester (AICTE) (Main) Examination

MODEL PAPER 1

SOFTWARE ENGINEERING

(Computer Science and Engineering)

Time: 3 Hours | Max. Marks: 70

Answer All Question from PART-A

Each Question carries equal marks

Answer any five Questions from PART-B

PART-A (10 × 2 = 20 Marks)

1. Define the term software and software engineering. (Unit-I / Q3)
2. What are the advantages of prototyping model over waterfall model? (Unit-I / Q15)
3. List out any five principles of software engineering. (Unit-II / Q1)
4. Write a short note on data objects. (Unit-II / Q8)
5. Write short notes on analysis pattern. (Unit-III / Q3)
6. Distinguish between classes and components. (Unit-III / Q8)
7. What is software component? Differentiate between hardware and software components. (Unit-IV / Q4)
8. Write about software documentation. (Unit-IV / Q8)
9. What is meant by software testing? And list its characteristics. (Unit-V / Q1)
10. Define quality of conformance. (Unit-V / Q12)

PART-B (50 Marks)

11. (a) Explain the evolving role of software. (Unit-I / Q21)

    (b) What is waterfall model? How is it different from other engineering process models? (Unit-I / Q37)

12. (a) Illustrate the hierarchy of system engineering. (Unit-II / Q18)

    (b) Briefly explain requirements engineering tasks. (Unit-II / Q30)

13. (a) Discuss in detail about requirement analysis. (Unit-III / Q9)

    (b) Draw and explain the flow of information during software design. (Unit-III / Q27)

14. Discuss the Mandel's design principles that allow the user to maintain control. (Unit-IV / Q32)

15. (a) Discuss why software architecture plays an important role during development and discuss various architectural terminology. (Unit-IV / Q12)

    (b) Explain about user interface analysis and user analysis. (Unit-IV / Q37)

16. (a) Discuss in detail about integration testing. What are its types? Explain the big bang approach. (Unit-V / Q28)

(b) What is basic path testing? Explain flow graph notation in detail. (Unit-V / Q18)

17. (a) What questions do black-box tests answer? (Unit-V / Q31)

(b) With the help of a diagram, explain the process of debugging. (Unit-V / Q50)

FACULTY OF ENGINEERING

B.E. V-Semester (AICTE) (Main) Examination

MODEL PAPER 2

SOFTWARE ENGINEERING

(Computer Science and Engineering)

Time: 3 Hours — Max. Marks: 70

Answer All Question from PART-A

Each Question carries equal marks

Answer any five Questions from PART-B

PART-A (10 × 2 = 20 Marks)

1. Distinguish between software products and software services. (Unit-I / Q6)
2. List and define the various software myths. (Unit-I / Q9)
3. What is deployment? (Unit-II / Q3)
4. What are the main uses of the requirements models? (Unit-II / Q10)
5. How do analysis classes manifest themselves as elements of solution space? (Unit-III / Q1)
6. What is a stereotype? (Unit-III / Q4)
7. How should a user assess an architectural style that has been derived? (Unit-IV / Q5)
8. What is user interface analysis? (Unit-IV / Q9)
9. Define unit testing and top-down integration testing. (Unit-V / Q5)
10. Define metrics. (Unit-V / Q7)

PART-B (50 Marks)

11. (a) Explain the various software myths. (Unit-I / Q26)

    (b) Briefly write about Agility. (Unit-I / Q55)

12. (a) Briefly explain the principles of software engineering. (Unit-II / Q11)

    (b) Discuss various steps in requirements engineering. What are the work products of engineering the requirements? (Unit-II / Q31)

13. (a) Discuss in detail about flow-oriented modeling. (Unit-III / Q16)

    (b) Explain in detail about CRC modeling. Also discuss in detail the three sections of index card. (Unit-III / Q23)

14. (a) Define interface. Discuss various types of interfaces. Give examples for each. (Unit-III / Q42)

    (b) Discuss about pattern based software design in detail. (Unit-III / Q45)

15. (a) What is meant by data design? Explain with an example. (Unit-IV / Q14)

    (b) What is architectural style? Discuss various categories of it. (Unit-IV / Q16)

16. (a) Explain the component from traditional view with an example. (Unit-IV / Q22)

(b) Explain in detail the following design notations,

(i) Graphical design notation

(ii) Tabular design notation. (Unit-IV / Q30)

17. (a) Give an overview of white-box testing techniques with help of flow graph. (Unit-V / Q34)

(b) What is meant by software quality? Discuss clearly the McCall's software quality factors. (Unit-V / Q71)

FACULTY OF ENGINEERING

B.E. V-Semester (AICTE) (Main) Examination

MODEL PAPER 3

SOFTWARE ENGINEERING

(Computer Science and Engineering)

Time: 3 Hours | Max. Marks: 70

Answer All Question from PART-A

Each Question carries equal marks

Answer any five Questions from PART-B

PART-A (10 × 2 = 20 Marks)

1. What is legacy software? Explain. (Unit-I / Q7)
2. What is waterfall model? (Unit-I / Q14)
3. What is the purpose of business process engineering. (Unit-II / Q5)
4. Discuss briefly on the approaches of requirements modeling. (Unit-II / Q9)
5. What is the intent of information hiding? (Unit-III / Q7)
6. Define functional Independence. (Unit-III / Q6)
7. Elaborate on "REP" in detail. (Unit-IV / Q3)
8. What are the essential steps involved in implementing the interface design? (Unit-IV / Q10)
9. Define black box testing strategy. (Unit-V / Q6)
10. Discuss the importance of quality assurance. (Unit-V / Q10)

PART-B (50 Marks)

11. (a) What do you mean by process framework? Explain the five generic process framework activities. (Unit-I / Q28)

    (b) 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? (Unit-I / Q42)

12. (a) Write in short about business process engineering. (Unit-II / Q19)

    (b) Explain about building the analysis model. (Unit-II / Q44)

13. (a) Discuss in detail the procedure of creating a preliminary use case. (Unit-III / Q13)

    (b) How to translate the analysis model into the design model? Explain with an example scenario. (Unit-III / Q40)

14. (a) Explain the principles of class-based component design in detail. (Unit-IV / Q23)

    (b) Draw the architectural context diagram. Explain different parts used in this diagram. (Unit-IV / Q18)

15. (a) Explain in detail the concept of cohesion. (Unit-IV / Q25)

(b) With the help of a diagram explain in detail the interface design evaluation cycle. (Unit-IV / Q43)

16. (a) What are the strategic approaches to software testing? (Unit-V / Q18)

(b) Explain about "regression testing" in detail. Explain the importance of it. (Unit-V / Q24)

17. What are the testing strategies for conventional software? Explain them in detail. (Unit-V / Q26)