PUBLISHED BY : **Apram Singh**
Quantum Publications®
(A Unit of Quantum Page Pvt. Ltd.)
Plot No. 59/2/7, Site - 4, Industrial Area,
Sahibabad, Ghaziabad-201 010

Phone : 0120 - 4160479

Email : pagequantum@gmail.com **Website: www.quantumpage.co.in**

Delhi Office : 1/6590, East Rohtas Nagar, Shahdara, Delhi-110032

Software Engineering (CS/IT : Sem-4 & 6)

1st Edition : *2010-11*
2nd Edition : *2011-12*
3rd Edition : *2012-13*
4th Edition : *2013-14*
5th Edition : *2014-15*
6th Edition : *2015-16*
7th Edition : *2016-17*
8th Edition : *2017-18*

Price: Rs. 90/- only

Printed at : Giriraj Printers, Delhi-110093.

CONTENTS

RCS 402 : Software Engineering

1
UNIT

Introduction

PART-1

Introduction to Software Engineering, Software Components, Software Characteristics, Software Crisis.

CONCEPT OUTLINE : PART-1

- **Software engineering :** Software engineering is concerned with all aspects of computer-based development including hardware, software and process engineering.
- Software contains computer programs, operating procedure and associated documentation which is necessary for software system.
- **Software crisis :** Software crisis is a set of difficulties or problems encountered while developing software.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. What do you mean by software engineering ? Discuss the objective / aim of software engineering.

Answer

Software engineering :

1. According to Boehm, "Software engineering is the practical application of scientific knowledge in the design and construction of computer programs and associated documentation required to develop, operate and maintain them".
2. Software engineering is a discipline whose aim is the production of fault free software that satisfies the user's needs and that is delivered on time and within budget.
3. It deals with cost effective solutions to practical problems by applying scientific knowledge.
4. Software engineering is the application of methods and scientific knowledge to create practical cost-effective solutions for the design, construction, operation and maintenance of software.

Objective / aim of software engineering :

i. To understand user conceptual models and development of better specification.

ii. To improve design language and reusable code.

iii. To satisfy the user's requirements.

iv. To achieve low maintenance and production cost.

v. To provide the software within budget and time.

vi. To achieve high performance.

Que 1.2. **Explain the components of software.**

OR

What is principle aim of software engineering discipline ? Define software components.

Answer

Principle aim of software engineering : Refer Q. 1.1, Page 1–2C, Unit-1.

Components of software :

i. **Set of programs :**

Program : A program is a set of instruction.

1. It is a collection of source code and objects code. Examples of small programs are factorial of number or print a sequence of number up to given limit.
2. A program is a subset of software and it becomes software only if documentation and operating procedure manuals are prepared.

ii. **Software documents :**

Software documentation consist all the description, programs, graphics and instruction pertaining to design, coding, testing and preparations of software. Good software contain following type of documentations :

1. Analysis and specification
2. Design
3. Coding
4. Testing

iii. **Operating procedure :**

a. Operating procedure provide information about what the software is, how to work with it, how to install it on our system and how to control all the activities of the software.

b. The main aim of this manual is to provide help to operating staff for producing desired output with the help of that particular software.

c. It can be divided in two parts :

1. User manual
2. Operational manuals

Que 1.3. **Define the term software engineering. Discuss the various characteristics of software with examples.**

AKTU 2012-13, Marks 10

Answer

Software engineering : Refer Q. 1.1, Page 1–2C, Unit-1.

Characteristics of software :

i. **Software is developed or engineered, it is not manufactured in classical sense :**

1. Engineering / developing and manufacturing both are logically different in working pattern and providing end product.
2. While for getting good quality of either hardware or software a good design is compulsory.
3. But in manufacturing phase of hardware, a quality problem can introduce if proper attention is not paid at the time of manufacturing as once a product is manufactured it cannot be modified easily.
4. While this type of problem does not exist in case of software as it can be easily modified and errors can be removed at the same movement.

ii. **Software does not wear out :**

1. In introduction stage of both hardware and software, there are chances of high failure rate.
2. Defects are corrected and then failure rate is reduced and steady state comes.
3. In case of hardware after some time this failure rate start rising again as the hardware parts begin to wear out with time which may caused by cumulative effect of dust, sudden temperature change (high/low), vibration and other environmental maladies.

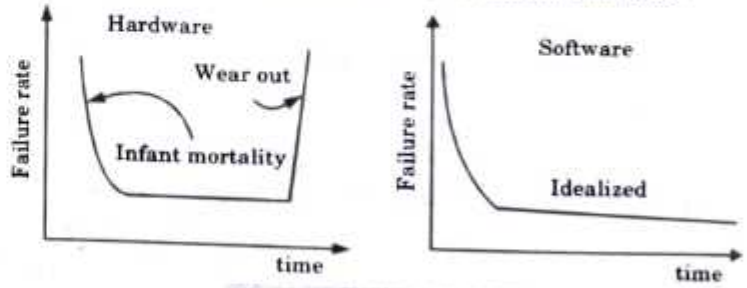


Fig. 1.3.1. Failure curves.

4. This failure rate of hardware in relation with time form a tub like curve which is also called "bath tub curve".
5. In case of software, these failure are due to some undiscovered errors but once the errors are corrected the curve become flat and continue at the same rate until its features become obsolete which is called idealized curve.
6. In real practice this ideal curve is not possible because software does not wear out but it becomes deteriorate and requires some maintenance (changes).

iii. Most software is custom-built rather than being assembled from existing components :

1. Nowadays industries are moving towards component based assembly of product which means all products are not manufactured at the same place.
2. This type of design uses some already manufactured hardware element available in the market.
3. So that designer can concentrate on truly innovative element of design.
4. While software is designed as per the requirement of the user.
5. At the time of designing software, the developer tries to design and implement software element in such a way that they may be reuse in many different programs.
6. For example, GUI is build using reusable component which can be used to develop other menus.

iv. Software is intangible :

1. Intangible products are those products which we cannot touch and whose quality cannot be measured until the whole product is checked.
2. For example, a small software program which is used to generate telephone bill cannot be said ok by only checking that, it is taking the correct input as customer name, address, number of local and STD call etc., until it is checked that it is producing correct output in required format.

Que 1.4. Explain why a software system that is used in a real world environment must change or become progressively less useful.

AKTU 2015-16, Marks 15

Answer

1. Systems must change or become progressively less useful for a number of reasons :

a. The presence of the system changes the ways of working in its environment and this generates new requirements. If these are not satisfied, the usefulness of the system declines.

b. The business in which the system is used changes in response to market forces and this also generates new system requirements.

c. The external legal and political environment for the system changes and generates new requirements.

d. New technologies become available that offer significant benefits and the system must change to take advantage of them.

2. Software change is very important because organizations are now completely dependent on their software systems and have invested millions of dollars in these systems.
3. Their systems are critical business assets and they must invest in system change to maintain the value of these assets.
4. A key problem for organizations is implementing and managing change to their legacy systems so that they continue to support their business operations.
5. There are a number of different strategies for software change :

a. Software maintenance : Changes to the software are made in response to changed requirements but the fundamental structure of the software remains stable. This is the most common approach used to system change.

b. Architectural transformation : This is a more radical approach to software change than maintenance as it involves making significant changes to the architecture of the software system. Most commonly, systems evolve from a centralised, datacentric architecture to client-server architecture.

c. Software re-engineering : This is different from other strategies in that no new functionality is added to the system. Rather, the system is modified to make it easier to understand and change. System re-engineering may involve some structural modifications but does not usually involve major architectural change.

Que 1.5. Discuss the following in brief :

i. Software characteristics

ii. Difference between module and software component

Answer

i. Software characteristics : Refer Q. 1.3, Page 1–4C, Unit-1.

ii. Difference between module and software component :

S. No.	Module	Software component
1.	The generic meaning of module is a group of reusable code, not tied to one specific program.	The generic meaning of component is a module with the additional restriction of substitutability using a specific interface.
2.	Module is used to model the system in function view.	Components are used to model a system in technical view.
3.	Module is a partitioned system into implementation units, independent task assignment. Modules might or might not be a component.	Component is a runtime entity (can be made up of modules), independent runnable unit.

Que 1.6. Explain the following statement : "Software engineering is a layered technology". AKTU 2015-16, Marks 10

Answer

Software development is totally a layered technology. That means, to develop software, one will have to go from one layer to another.

The layers are related and each layer demands the fulfillment of the previous layer.

Fig. 1.6.1 is the upward flowchart of the layers of software development.

Fig. 1.6.1.

Software engineering is divided into following four layers :

1. **A quality focus :**
   a. Any engineering approach must rest on quality.
   b. The "Bed Rock" that supports software engineering is quality focus.
2. **Process :**
   a. Foundation for SE is the process layer.

   b. SE process is the GLUE that holds all the technology layers together and enables the timely development of computer software.
   c. It forms the base for management control of software project.
3. **Methods :**
   a. SE methods provide the "Technical Questions" for building software.
   b. Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.
4. **Tools :**
   a. SE tools provide automated or semi-automated support for the "Process" and the "Methods".
   b. Tools are integrated so that information created by one tool can be used by another.

Que 1.7. What is software crisis ? Discuss main reasons and results of software crisis.

Answer

Software crisis is a set of difficulties or problems encountered while developing software.

Reasons of software crisis :

1. Communication gap between end user and software developer.
2. Misinterpretation of requirement, improper problem definition, no knowledge about end user environment.
3. High software cost compare to hardware cost.
4. Changes in end user requirement increase the problem for developer.
5. Inaccurate scheduling and cost estimation of project.
6. Data collection and analysis is not up to mark or not done timely.
7. Increase in size of software.
8. Increase in complexity of problem area.

Results of software crisis :

1. Poor quality of software.
2. Late delivery of software.
3. High cost compare to estimate cost.
4. Not able to meet current demand for which it should accommodate.

PART-2

Software Engineering Processes, Similarity and Differences from Conventional Engineering Processes, Software Quality Attributes Software Development Life Cycle (SDLC) Model : Waterfall Model, Prototype Model, Spiral Model, Evolutionary Development Model, Iterative Enhancement Models.

CONCEPT OUTLINE : PART-2

- **Software engineering process :** It is a set of activities that leads to the production of a software product.
- **Conventional engineering process :** It is a structured logical approach to develop a stable final product.
- **Software quality :** It is the conformance to implicit stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected to all professionally developed software.
- **Software development life cycle :** It represents number of identifiable stages under which software goes during its life.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.8. **Define conventional engineering processes. How it is different from software engineering processes ?**

Answer

1. Conventional engineering process is a structured logical approach to develop a stable final product.
2. Conventional engineering process uses the software tools to design and analyze their own systems.
3. Therefore, developed electronic document goes through analysis, design implementation and testing phases just like software engineering process.

Difference between software engineering process and conventional engineering process :

S. No.	Issues	Software engineering process	Conventional engineering process
1.	Foundations	Based on computer science, information science and discrete mathematics.	Based on science, mathematics and empirical knowledge.
2.	Cost	Compilers and computers are cheap, so software engineering and consulting are often more than half of the cost of a project.	Construction and manufacturing costs are high, so conventional engineering may only be 15% of the cost of a project.
3.	Innovation	Often new and untested elements are applied in software projects.	Only known and tested principles are applied and untested innovations are used only if necessary to meet product requirements.
4.	Replication	Replication is trivial. Most development effort goes into building new or changing old design to add new features.	Changing old design and creating new design requires significant development efforts.
5.	Management status	Few software engineers manage anyone.	Many of conventional engineers manage construction, manufacturing or maintenance crews, so they all are treated as managers.

Que 1.9. **Explain software quality attributes.**

AKTU 2012-13, Marks 05

OR

Define software quality.

Answer

Software quality is a field of study and practice that describe the desirable attributes of software products. It is concerned with following :

1. Conformance of requirements.

2. Fitness for the purpose
3. Level of satisfaction

The major software quality attributes are as follows :

1. **Correctness :** A system is functionally correct if it behave according to its functional requirement specification.
2. **Reliability :** A system is said to be reliable if it gives desired output even in case of change of components (like hardware) or in case of overload conditions.
3. **Robustness :** It explain how does a system behave in situations not specified by its requirement *i.e.*, incorrect input, hardware failure, loss of power.
4. **Portable :** Software system portability is the ease with which a software system can be adapted to run on computer other than one for which it was designed.
5. **Understandability :** How well do the developers understand a system they have developed, support evolvability and understandability.
6. **Performance :** In software engineering performance is equated with efficiency *i.e.*, how quickly does it perform its operation.
7. **Interoperability :** Can a system cooperate with another system.
8. **Verifiable :** It checks whether the properties of the system be verified or not.
9. **Timeliness :** The ability to deliver a system on time.
10. **Efficiency :** It is the ability of software system to fulfill its purpose with best possible utilization of necessary resources such as time, storage, transmission channel, peripherals etc.

Que 1.10. What is software quality ? What are three dimensions of software quality ? Explain briefly.

AKTU 2013-14, Marks 05

AKTU 2015-16, Marks 10

Answer

Software quality : Refers Q. 1.9, Page 1–10C, Unit-1.

Three dimensions of software quality are :

1. **Quality of design :** It is the extent to which the design reflects a product or service that satisfies customer needs and expectations. All the necessary characteristics should be designed into the product or service at the outset.

2. **Quality of conformance :** It is the extent to which the product or service conforms to the design standard. The design has to be faithfully reproduced in the product or service.
3. **Quality of use :** It is the extent by which the user is able to secure continuity of use from the product or service. Products need to have a low cost of ownership, be safe and reliable, maintainable in use, and easy to use.

Que 1.11. Explain Software Development Life Cycle (SDLC). Discuss various activities during SDLC.

AKTU 2014-15, Marks 05

AKTU 2016-17, Marks 10

Answer

Software Development Life Cycle (SDLC) :

1. Software development life cycle a diagrammatic representation which also provides description of various phases and their sequence in life cycle of software product.
2. Software life cycle represents number of identifiable stages under which software goes during its life.
3. We have different life cycle models, each one have its own advantages and disadvantages.
4. We can choose any one of them on the basis of :
   i. Development speed
   ii. Product quality
   iii. Project visibility
   iv. Administrative overhead
   v. Risk exposure

Phases of software development life cycle models :

i. Requirement definition (system analysis and system specification)
ii. System and component (software) design
iii. Implementation and unit testing
iv. Integration and system testing
v. Operation and maintenance

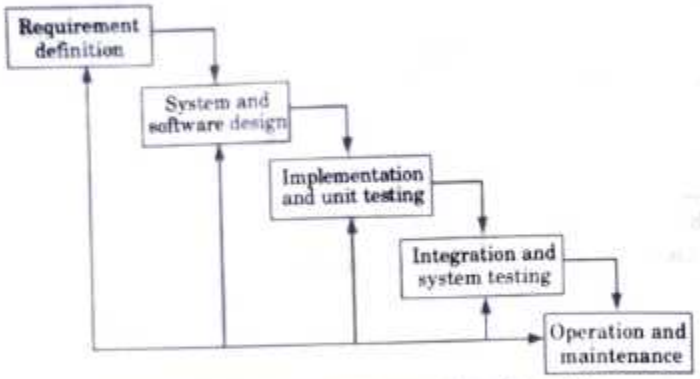


Fig. 1.11.1. Software development life cycle model.

End product of each phase :

a. The output of requirement definition is software requirement specification (SRS) document.

b. The output of system and software design is entity relationship (ER) diagram, data flow diagram (DFD), flow chart etc.

c. The output of implementation and unit testing is coded program.

d. The output of integration and system testing is code without error or bugs.

e. The output of operation and maintenance phase is implemented system that is running well at client end.

Que 1.12. Define waterfall model. **AKTU 2014-15, Marks 2.5**

AKTU 2016-17, Marks 7.5

OR

Write the advantages and disadvantages of waterfall model. Why the use of iterative waterfall is not good for large projects ?

Answer

Waterfall model :

1. Waterfall Model is a sequential model that divides software development into different phases.
2. In waterfall model progress is seen as flowing steadily downwards (like a waterfall) through the phases of feasibility study, analysis, design, coding, testing, implementation and maintenance.

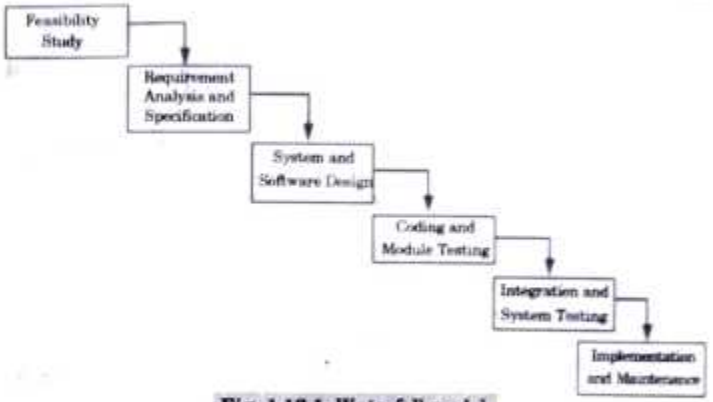


Fig. 1.12.1. Waterfall model.

The different phases of waterfall model are :

1. **Feasibility study :**
   a. This phase is used to check whether the new proposed system is economically, technically and operationally feasible or not.
   b. In this information is gathered about what outputs to be produce, what input is required and what process can be used and then different solution strategies are formulated.
   c. The output of this phase is feasibility report.
2. **Requirement analysis and specification :**
   a. This phase analyze and specifies the requirement of user / customer and document them properly.
   b. In requirement analysis the data are gathered from users.
   c. Finally the requirements organized systematically in the form of document called software requirement specification (SRS) document.
   d. The output of this phase is software requirement specification (SRS) document.
3. **System and software designing phase :** In design phase, overall structure or architecture is developed which is transformation of requirement specified in software requirement specification (SRS). There are two main types of design approach :
   a. Traditional design approach : In this function, software requirement specification (SRS) document is decomposed into small sub functions and information flow among these sub functions are also checked. It is done in two parts :
      1. Architectural design
      2. Detailed design

b. **Object-oriented approach :** In this design approach problem and their solution are represented in the form of object and then relationship is developed among these objects.

4. **Coding and module testing :**
   a. In this phase, system design is translated into source code also called program code.
   b. Here programming for different module is done in selected programming language.
   c. The output of this phase is programmed module.

5. **Integration and system testing :**
   a. Individually tested modules are integrated here according to planned system to develop the system.
   b. There are two main testing : Alpha testing – It is done at developer end and beta testing – It is done at user end.
   c. The output of this phase is testing and integration report.

6. **Implementation / Installation and maintenance :**
   a. In this phase system is installed at the user end and it is checked if there is any upgradation required in hardware or software element at user end.
   b. Once the software is properly installed there is need of maintaining the software. This ensures that software is working properly at user site.

Advantages of waterfall model :

i. Easy to understand.
ii. Each stage has defined input and output.
iii. Helps in project planning.

Disadvantage of waterfall model :

i. Iteration not possible as it is one way street.
ii. Requirements freezing at starting stage.
iii. Sequencing – no stage can start until the previous stage is completed.
iv. A rigid model.
v. Difficulty in accommodating changes after project development.

The use of iterative waterfall model for large project is not good because :

i. It is difficult to predict how the new system will be.
ii. There is difficulty in predicting the entire requirements at very beginning of project, because even end user does not know all requirements initially.

Que 1.13. **What are the phases in the waterfall model ? Which phase consumes the maximum effort for developing a software ? Discuss with proper justification.** **AKTU 2012-13, Marks 10**

Answer

Phases in waterfall model : Refer Q. 1.12, Page 1–13C, Unit-1.

The testing phase consumes the maximum effort for developing software.

Justification :

This phase is not just about testing but contains activities such as document reviews and code walkthroughs. It has a high degree of overlap with the other phases which results in higher quality in the final product. This phase is done by both the developers and the users.

Que 1.14. **Explain the working of prototype model with suitable diagram.**

OR

Discuss the prototype model. What is the effect of designing a prototype on the overall cost of the software project ?

OR

Explain about prototyping model of software development. What are the advantages of it over waterfall model ? Discuss.

Answer

There are certain drawbacks in waterfall model. This model was developed to overcome from two main drawbacks of waterfall model. They are :

i. Difficulty in predicting how the new system will be.
ii. Difficulty in predicting the entire requirements at very beginning of project, because even end users do not know all requirements initially.

Prototype model :

1. A prototype is a partially developed product. It has limited functional capabilities, low reliability and inefficient performance.
2. The prototype model suggests that before developing the actual software, a working prototype of the system should be built.
3. The model starts with an initial requirements gathering phase.
4. A quick design in carried out and the prototype model is built using several shortcuts.
5. The prototype product usually turns out to be a very crude version of the actual system. The prototype model is shown in the Fig. 1.14.1.

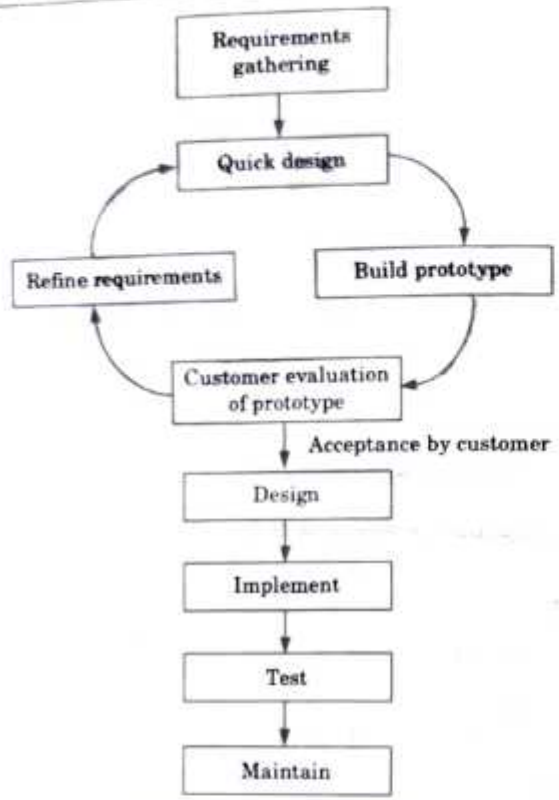


Fig. 1.14.1. Prototype model.

Advantages :

1. A partial product is built in the initial stages. Therefore customers get a chance to see the product early in the life cycle and thus give necessary feedback.
2. Requirements become clearer resulting into an accurate product.
3. New requirements can be easily accommodated, as there is scope for refinement.
4. Flexibility in design and development is also supported by the model.

Disadvantages :

1. Developers in a hurry may build prototypes and end up with sub-optimal solutions.
2. If end user is not satisfied with initial prototype, he may lose interest in the project.
3. Poor documentation.

Effect of designing a prototype on the overall cost of software project :

1. The prototype may be a usable program, but is not suitable as the final software product. The reason may be poor performance, maintainability or overall quality.
2. The code for the prototype is thrown away; however the experience gathered from developing the prototype helps in developing the actual system.
3. Therefore, the development of a prototype might involve extra cost, but overall cost might turn out to be lower than that of an equivalent system developed using the waterfall model.

Que 1.15. Explain evolutionary development model in brief.

Answer

1. Evolutionary models are iterative type models which allow to develop more complete versions of the software.
2. In this model, the system is first broken down into several modules or functional units that can be incrementally implemented and delivered.
3. The developers first develop the core modules of the system.
4. This initial product skeleton is refined into increasing levels of capability by adding new functionalities in successive versions.
5. In this approach, each successive version of the product is a functioning system capable of performing some more useful work.
6. This model is also known as the successive versions model.

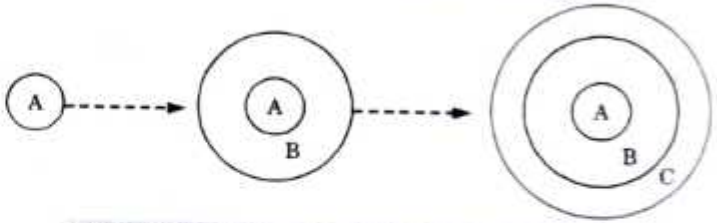


Fig. 1.15.1. A, B, C are modules of a software product that are incrementally developed and delivered.

Advantages :

1. Early delivery of portions of the system even though some of the requirements are not yet decided.
2. The core modules get tested thoroughly, thereby reducing chances of errors in the final product.
3. Use of early releases as tool for requirements elicitation.

Disadvantages :

1. For most practical problems, it is difficult to subdivide the problem into several functional units that can be incrementally implemented and delivered.
2. Model can be used only for very large problems, where it is easier to identify modules for incremental implementation.

Que 1.16. **Why are evolutionary models considered by many to be the best approach to software development in a modern context ?**

AKTU 2013-14, Marks 05

Answer

1. Evolutionary models are considered to be best approach because timelines for the development of modern software are getting shorter and shorter, customers are becoming more diverse (making the understanding of requirements even harder), and changes to requirements are becoming even more common (before delivery), we need a way to provide incremental or evolutionary delivery.
2. The evolutionary process accommodates uncertainty better than most process models, allows the delivery of partial solutions in an orderly and planned manner, and most importantly, reflects what really happens when complex systems are built.

Que 1.17. **Explain why programs which are developed using evolutionary development are likely to be difficult to maintain ?**

AKTU 2014-15, Marks 05

Answer

1. The specifications of evolutionary development projects are often abstract, and as the project continues, the development and validation portions of software engineering overlap one another.
2. When a system is produced using the evolutionary development model, features tend to be added without regard to an overriding design.
3. With each modification, the software becomes increasingly disorganized.
4. System maintenance (fixing bugs; or adding new features after the system has been released) is hampered by these problems, as it is harder to identify the source of bugs in poorly designed systems.
5. Also, keeping the documentation up-to-date over successive "evolutions" is uncommon. Poor documentation also makes maintenance more difficult.
6. This is why programs which are developed using evolutionary development are likely to be difficult to maintain.

Que 1.18. **Draw neat sketch of spiral model and explain its different activities. What are the different cycles indicate in this model ? What are its advantages over traditional iterative process models ? Why is it not suitable for small projects ?**

AKTU 2013-14, Marks 10

OR

Explain the spiral life cycle model with its merits and demerits.

OR

Define spiral model.

AKTU 2014-15, Marks 2.5

AKTU 2016-17, Marks 2.5

Answer

1. The spiral model is a risk-driven process model generator for software projects.
2. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.
3. Its main feature is risk avoidance rather than documentation or coding.
4. This model is more flexible than any other model as number of phases through which the product will be developed is not fixed, it depends on software requirement.
5. The two basic step of this model are :
   i. Identify the sub-problem which is having highest risk.
   ii. Find solution for that particular problem (risk).
6. Generally, there are four spirals in Boehm spiral life cycle model.
7. The inner (first) spiral is concept development cycle, the second spiral indicates new product development cycle; the third spiral represents product enhancement cycle and fourth spiral is known as product maintenance cycle.
8. Each phase of this model is split into four quadrant (sections) having specific functions :
   a. In the first quadrant, we do identification of objectives; find out different alternative for achieving the objective and present constraints.
   b. In the second quadrant, we evaluate these alternatives on the basis of objective and constraints. The main focus in this step is given on evolution of alternative on the basis of risk as risk causes the chances of unmet objectives. This model helps in coping up many project risks.
   c. In the third quadrant, project development and validation is carry out.
   d. In the fourth quadrant, the project is reviewed and decision is made up whether to continue with a further loop of spiral or not.

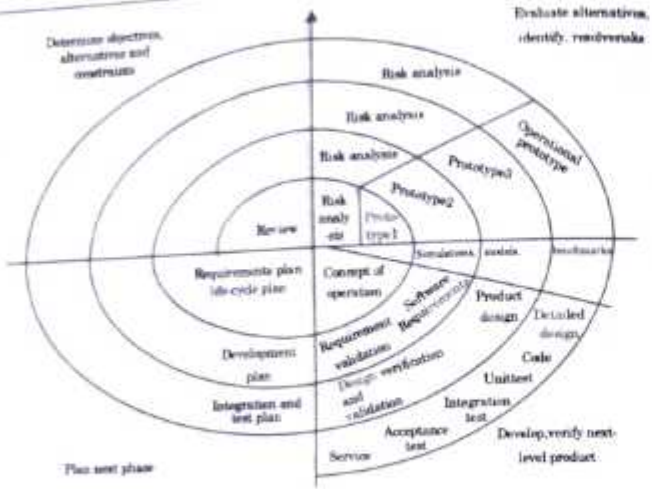


Fig. 1.18.1.

Advantages of Boehm spiral life cycle model :

i. This model tries to resolve all possible risks involved in the project starting with the highest risk.

ii. Little documentation is required as compared to waterfall model.

iii. Efficient use of prototyping and component based design.

iv. It is very flexible model.

Disadvantages of Boehm spiral life cycle model :

i. This model is not suitable for small projects.

ii. This is time consuming model.

iii. The cost of risk analysis is high which makes the model costly.

Advantages of spiral model over traditional iterative process model :

i. The risk analysis and validation steps eliminate errors in the early phase of development.

ii. The model makes use of techniques like reuse, prototyping and component based design.

iii. It becomes equivalent to another life cycle model in appropriate situations. The model is not suitable for small projects as cost of risk analysis may exceed the actual cost of the project.

Que 1.19. Why spiral model is called a meta model ?

Answer

1. The spiral model is called meta model, because its each stage is followed by explicit recognition of risk.

2. A cycle of the spiral begins by elaborating objectives such as performance and functionality.
3. The next step is to resolve risks by information gathering activities such as more detailed analysis, prototyping and simulation.
4. Once risks have been assessed, some development is carried out, followed by a planning activity for the next phase of process.
5. The spiral model encompasses the strength of the waterfall model while including risk analysis, risk management support and management processes.

Que 1.20. Current trends in software engineering are moving away from the waterfall model for large projects and moving toward iterative methods such as the spiral model ? What are we gaining and losing as a result ? Explain with suitable examples.

Answer

1. While in the spiral model, the customer is made aware of all the happenings in the software development, in the waterfall model the customer is not involved.
2. This often leads to situations, where the software is not developed according to the needs to the customer.
3. In the waterfall model, when the development process shifts to the next stage, there is no going back.
4. This often leads to roadblocks, especially during the coding phase.
5. However, in the spiral model, since there are different iterations, it is rather easier to change the design and make the software feasible.
6. In the spiral model, one can revisit the different phases of software development, as many times as one wants, during the entire development process.
7. This also helps in backtracking, reversing or revising the process.
8. However, the same is not possible in the waterfall model, which allows no such scope.
9. Often people have the waterfall model or spiral model confusion due to the fact that the spiral model seems to be a complex model.
10. It can be attributed to the fact that there are many iterations, which go into the model.
11. At the same time, often there is no documentation involved in the spiral model, which makes it difficult to keep a track of the entire process.
12. On the other hand, the waterfall model has sequential progression, along with clear documentation of the entire process.
13. This ensures one has a better hold over the entire process.

Que 1.21. **Explain iterative enhancement model in brief.**

Answer

1. It is a combination of benefits of waterfall and prototype model. It is very popular model used by industries.
2. In this model software is developed in increments, each increment adds some functional capability to the system until full system is developed.
3. It provide better testing result as testing after each increment is easy as compare to entire model testing of waterfall model and prototyping used in this model help in identifying the system requirements.
4. In this model a partial product is developed on few easily understandable requirements of overall requirements, and then a project control list is developed which contain the entire task which have to be performed in final implementation.
5. This helps in finding out how far the product is from final product. This product is given to the user to work and slowly enhancement is done in this product which increases its functionality.

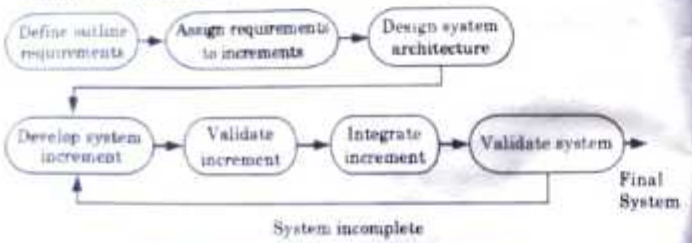


Fig. 1.21.1. Iterative enhancement model.

6. That is why it is called incremental model. The model prioritizes the system requirement and implements them in group.
7. The process is iterated until the control list is empty and final implementation of system is done.
8. In this model developer themselves provide specification so they have good control over system development.

Advantages of iterative enhancement model :

i. Product delivered in parts so the cost of product is also distributed.
ii. Easy testing as testing of the system is done after each increment.
iii. User can see working of software in early stage.
iv. User requirement become clearer.
v. Low failure risk.

Disadvantages of iterative enhancement model :

i. Total cost of product development is high.

ii. Well defined project planning is required for distribution of work.
iii. Testing each model causes extra cost.

Que 1.22. **What are software process models ? Distinguish iterative enhancement model and spiral model.** **AKTU 2013-14, Marks 05**

Answer

Software process model :

1. A software/system process model is a description of the sequence of activities carried out in an software engineering project, and the relative order of these activities.
2. It provides a fixed generic framework that can be tailored to a specific project.
3. Project specific parameters will include :
   a. Size (person-years)
   b. Budget
   c. Duration

Difference :

S. No.	Properties of model	Iterative enhancement model	Spiral model
1.	Handle large project	Not Appropriate	Appropriate
2.	Cost	Low	Expensive
3.	User involvement	Intermediate	High
4.	Testing	After every iteration	At the end of the each phase
5.	Overlapping phases	Yes (Parallel development exists)	No
6.	Objective	Rapid development	High assurance

Que 1.23. **Define the term "software engineering". Explain the major differences between software engineering and other traditional engineering disciplines.** **AKTU 2014-15, Marks 05**

Answer

Software engineering : Refer Q. 1.1, Page 1-2C, Unit-1.

Difference between software engineering and traditional engineering :

S. No.	Software engineering	Traditional engineering
1.	Software engineering is based on computer science, information science and discrete mathematics.	Traditional engineering is based on mathematics, science and empirical knowledge.
2.	Software engineers construct non-real (abstract) artefacts.	Traditional engineers construct real artefacts.
3.	In software engineering, there are two main concerns that are cost of development and reliability which are measured by the number of errors per thousand lines of source code.	In traditional engineering, two main concerns for a product are cost of production and reliability measured by time to failure.
4.	Software engineers often apply new and untested elements in software projects.	Traditional engineers generally try to apply known and tested principles and limit the use of untested innovations to only those necessary to create a product that meets its requirements.
5.	Software engineers emphasize projects that will live for years.	Some traditional engineers solve long-ranged problems that ensure for centuries.

Que 1.24. Explain Rapid Application Development (RAD) model.

AKTU 2012-13, Marks 05

Answer

1. Rapid Application Development (RAD) is a software development methodology that uses minimal planning in favour of rapid prototyping.
2. In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.
3. RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

Following are the various phases of the RAD model :

1. **Business modeling :**
   a. The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels.
   b. A complete business analysis is performed to find the vital information for business.
2. **Data modeling :**
   a. The information gathered in the business modeling phase is reviewed and analyzed to form sets of data objects vital for the business.
   b. The attributes of all data sets and the relation between these data objects are identified and defined.
3. **Process modeling :**
   a. The data object sets defined in the data modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model.
   b. Process descriptions for adding, deleting, retrieving or modifying a data object are given.
4. **Application generation :** The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.
5. **Testing and turnover :**
   a. The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration.
   b. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage.

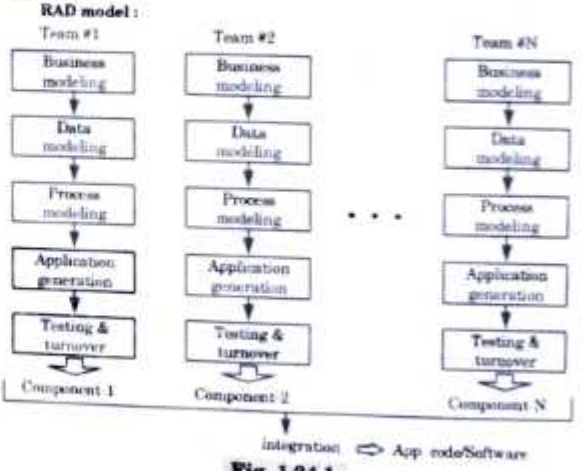


Fig. 1.24.1.

Que 1.25. **What are the characteristics of a software process ?**

AKTU 2016-17, Marks 10

Answer

Following are the software process characteristics :

1. **Understandability :** To what extent is the process explicitly defined and how easy is to understand the process definition ?
2. **Visibility :** Do the process activities give clear results so that the progress of the process is extremely visible ?
3. **Robustness :** Can the process continue in spite of unexpected problems ?
4. **Reliability :** Is the process designed in such a way that process errors are avoided or trapped before they result in product errors ?
5. **Maintainability :** Can the process evolve to reflect changing organizational requirements or identified process improvements ?
6. **Rapidity :** How fast can be the process of delivering a system from a given specification be completed ?
7. **Supportability :** To what extent can CASE tools support the process activities ?

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Discuss software engineering and its components.

Ans. Refer Q. 1.1 and Q. 1.2.

Q. 2. What is software crisis ? Discuss main reasons and results of software crisis.

Ans. Refer Q. 1.7.

Q. 3. Differentiate between software engineering process and conventional engineering process.

Ans. Refer Q. 1.8.

Q. 4. Write short note on waterfall model.

Ans. Refer Q. 1.12.

Q. 5. Explain spiral model.

Ans. Refer Q. 1.18.

Q. 6. What are software process models ?

Ans. Refer Q. 1.22.

2 UNIT

Software Requirement Specifications (SRS)

PART-1

Requirement Engineering Process : Elicitation, Analysis, Documentation, Review and Management of User Needs, Feasibility Study.

CONCEPT OUTLINE : PART-1

- **Requirement elicitation** : It is the process during which software requirements are discovered, expressed and revealed from system requirements.
- **Requirement analysis :** It is the activity during which the requirement gathered during elicitation are analysed for conflicts ambiguities, inconsistencies, missing requirements.
- **Documentation :** It is the activity which is written at the end of requirement elicitation and analysis.
- **Feasibility study :** It is to determine whether developing the product is financially and technically feasible.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.1. **Discuss the significance of requirement engineering. Also, write the various steps of requirement engineering with proper explanation.**

Answer

Significance of requirement engineering :

1. Requirement engineering is the process of determining user expectations for a new or modified product.
2. These user expectations, called requirements, must be quantifiable, relevant and detailed.
3. In software engineering, such requirements are often called functional specifications.
4. Requirements analysis is an important aspect of project management.
5. Requirements analysis involves frequent communication with system users to determine specific feature expectations.
6. Its goal is to create a requirements specification that is complete, correct, and understandable to both customers and developers.

Steps of requirement engineering :

This process consists of four steps as shown in Fig. 2.1.1.

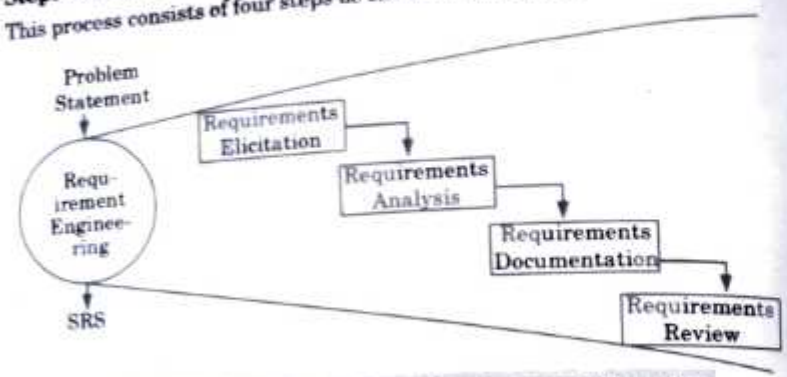


Fig. 2.1.1. Crucial process steps of requirement engineering.

i. **Requirements elicitation :**
   a. This is also known as gathering of requirements.
   b. Here, requirements are identified with the help of customer and existing systems processes, if available.

ii. **Requirements analysis :**
   a. The requirements are analysed in order to identify inconsistencies, defects, omissions etc.
   b. We describe requirements in terms of relationships and also resolve conflicts, if any.

iii. **Requirements documentation :**
   a. This is the end product of requirements elicitation and analysis.
   b. The requirement documentation is very important as it will be the foundation for the design of the software.
   c. The document is known as software requirements specification (SRS).

iv. **Requirements review :**
   a. The review process is carried out to improve the quality of the SRS.
   b. It may also be called as requirements verification.
   c. For maximum benefits, review and verification should not be treated as a discrete activity to be done only at the end of the preparation of SRS.
   d. It should be treated as continuous activity that is incorporated into the elicitation, analysis, and documentation.

Que 2.2. What is the significance and use of requirement engineering ? What are the requirement validation techniques ?

Answer

Significance of requirement engineering : Refer Q. 2.1, Page 2–2C, Unit-2.

Use of requirement engineering :

Requirement engineering is used in :

1. Determining the feasibility of producing a particular product as part of the product line.
2. Determining the production, testing, and deployment of the particular product.
3. Determining the evolution of the product line (that is, the incorporation of changes) that results from that product development. Product-specific requirements often "grow up" to become product line requirements if they can be slightly generalized or if they pop up in more than one product. That is the primary mechanism for the evolution of software product lines over time.

Requirement validation technique :

1. **Test case generation :** In this technique, the various tests are developed for requirements. The requirement check can be carried out with following :
   a. **Verifiability :** Is the requirement realistically testable ?
   b. **Comprehensibility :** Is the requirement properly understood ?
   c. **Traceability :** Is the origin of requirement clearly stated ?
   d. **Adaptability :** Can a requirement be changed without a large impact on other requirements ?
2. **Automated consistency analysis :** If the requirements are expressed in the form of structured or formal notations, then CASE tools can be used to check the consistency of the system. A requirements database is created using a CASE tool that checks the entire requirements in the database using rules of method or notation. The report of all inconsistencies is identified and managed.
3. **Prototyping :** Prototyping is normally used for validating and eliciting new requirements of the system. This helps to interpret assumptions and provide an appropriate feedback about the requirements to the user. For example, if users have approved a prototype, which consists of graphical user interface, then the user interface can be considered validated.

Que 2.3. What is requirement elicitation ? Explain in detail the procedure and problems in requirement elicitation.

Answer

1. Requirement elicitation is the activity during which software requirements are discovered, expressed, and revealed from system requirements.
2. Its source may be system requirement documentation, customer or market analysis.
3. Various technologies are used to involve the user on the basis of level of involvement required.

Elicitation procedure : A general elicitation procedure contain following steps :

1. Identifying relevant source of information *i.e.*,
   a. User b. Customer c. Domain experts
2. Determining what information is needed by asking questions.
3. Analyzing the gathered information, looking for implications, inconsistencies, or unresolved issues.
4. Confirming our understanding of the requirements with the user.
5. Creating requirement statements.

Problems in requirement elicitation :

1. User is unable to explain his requirements.
2. Less knowledge of technology and may not consider what is possible.
3. Use of different language by developer and user.
4. Lack of skills in developer.
5. User is not providing some information due to some reason.
6. Requirements are coming from different sources.

Que 2.4. What is system documentation ? Explain in detail.

Answer

1. System documentation is written text or illustration that either explains how system operates or how to use it.
2. System documentation includes all the descriptions, graphics, programs and instructions pertaining to the design, implementation, and operation of a system.

The five types of documentation manuals required by a system are :

i. **System design documentation :**
   a. The purpose of this documentation is to provide technical details of the system at a top level.
   b. Its contents can be derived from the end products of analysis phase.
   c. System design documentation consist of the following :

   1. Formulation of the problem
   2. Feasibility issue
   3. Overview of the system, its sub system and their interfaces
   4. Output report and screen displays
   5. Input and source documents
   6. Data dictionary

ii. **Software documentation** : Software documentation consist of following :
   a. **Hardware specification :** Description of equipment, peripheral devices, communication hardware.
   b. **System software specification :** Description of the operating system, all other system software such as DBMS, screen formatter, forms generator and communication software.
   c. **Application software specification :** Description of each application program.
   d. **File design or database design :** Description of each file used in system, its access method and its organization.

iii. **Operation documentation :** Its goal is to help the operation staff, run programs in proper order, distribute the output, and handle the errors which come during execution of programs. It consists of the following :
   a. **Multiple system flowcharts with supportive narratives :** Description of each program, its purpose, and the execution of the program.
   b. **Operation procedure for each program :** Description of input required and its format, file required, output produced, error conditions and operators required etc.

iv. **User reference documentation :** It consists of the following :
   a. **Objectives of documentation :** It provides outline of the purpose and use of the document.
   b. **Input documents :** It covers complete list and samples of all the input documents used by the system.
   c. **Output reports and screen display :** It contain the complete list and sample of all reports, with their frequencies and distributions, and examples of screen displays.
   d. **Processing logics and error conditions :** It provide outline of processing logic for each output report, error condition etc.

v. **System maintenance staff documentation :** It is the list of computer department staff members who would be contacted in case of any problem that the user cannot rectify.

Que 2.5. Write short note on review and management of user needs.

Answer

Review of user needs :

i. It is a manual process that involves people from both client and contractor organizations.

ii. They check the requirements for anomalies and omissions.

iii. The review process may be managed in the same way as program inspections.

iv. Alternatively, it may be organized as a broader activity with different people checking different parts of the document.

v. Following activities are performed in user needs review :

1. Plan a review
2. Review meeting
3. Follow-up actions
4. Document after review
5. Understandability
6. Checking for redundancy
7. Completeness
8. Consistency
9. Adaptability
10. Conformation of standards

Management of user needs :

1. Management of user needs is the process of documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders. It is a continuous process throughout a project.
2. A requirement is a capability to which a project outcome (product or service) should conform.
3. The purpose of management of user needs is to ensure that an organization documents, verifies, and meets the needs and expectations of its customers and internal or external stakeholder.
4. Following activities are performed in management of user needs :
   a. Investigation
   b. Feasibility
   c. Design
   d. Construction and test
   e. Requirements change management
   f. Release

Que 2.6. What do you mean by feasibility study ? Explain various types of feasibility and important steps that are carried out during the feasibility study phase.

Answer

1. Feasibility study is the analysis of risks, costs and benefits relating to economics, technology and user operations.
2. The output of feasibility study is a document known as feasibility study report.
3. The report must answer the following key questions :
   i. Will the new system provide a better way to do jobs ?
   ii. What the proposed system will do ?
   iii. What will be the estimated cost of proposed system ?
   iv. What will be the benefits from proposed system ?
4. There are several types of feasibility depending on the aspect they covers. Some important feasibility is as follows :
   a. **Technical feasibility :** The technical feasibility study basically centers on alternatives for hardware, software and design approach to determine the functional aspect of system.
   b. **Operational feasibility :** Operational feasibility is a measure of how people are able to work with system. This type of feasibility demands if the system will work when developed and installed.
   c. **Economical feasibility :** Economical feasibility is a measure to find out whether it is economically worthwhile to invest in the project. Cost benefit analysis is carried out to determine economical feasibility.

The feasibility study involves following eight steps :

1. Form a project and appoint a project leader.
2. Prepare system flow chart.
3. Enumerate potential candidate system.
4. Describe and identify the characteristics of candidate system.
5. Determine and evaluate performance and cost effectiveness of each candidate system.
6. Weight system performance and cost data.
7. Select best performance system.
8. Prepare and report final project directive to management.

Que 2.7. Explain the cost benefit analysis to determine the economic feasibility on the basis of direct cost, indirect cost, tangible benefits and intangible benefits.

Answer

i. The cost benefit analysis is necessary to determine the economic feasibility.

ii. The basic objective of cost benefit analysis is to find out whether it is economically worthwhile to invest in the project.

iii. If the return on the investment is good then the project is considered economically worthwhile.

iv. Cost benefits analysis is performed by first listing all the costs associated with the project.

1. **Cost benefit analysis on the basis of cost :** The cost consist of direct cost / tangible cost and indirect costs / intangible cost.
   a. **The direct cost / tangible cost :**
      i. The direct cost is directly associated with the project such as cost of buying the equipments (hardware like printer, computers, disk drive etc.), software, salary to staff.
      ii. Cost involved in preparation of physical location such as air conditioner, lights, wiring etc.
      iii. Operating cost such as use of paper, pen drives etc.
   b. **The indirect cost / intangible cost :**
      i. The indirect cost is not directly associated with project, it is a result of operations that are directly associated with system of a given project.
      ii. It is often referred as overhead.
      iii. Indirect cost involves time spend by user in discussing problems with system analyst, gathering the data about the problems etc.
2. **Cost benefit analysis on the basis of benefits :** The benefits can also be broadly classified as tangible benefits and intangible benefits.
   a. **Tangible benefits :** The tangible benefits are directly measurable. They are :
      i. Decreasing salary cost by automating manual procedure.
      ii. Preventing costly but frequent errors.
      iii. Sending bills early in the month.
      iv. Increasing control over inventory level.
      v. Increase in production.
   b. **Intangible benefits :** The intangible benefits are not directly measurable. They are :
      i. Better service to customer.
      ii. Superior quality of product.
      iii. Upgrading or creating new customer services.
      iv. Developing a new image in the market.
      v. Reducing repetitive or monotonous work for employee.

PART-2

Information Modeling, Data Flow Diagrams, Entity Relationship Diagram, Decision Tables, SRS Document, IEEE Standards for SRS.

CONCEPT OUTLINE : PART-2

- Information modeling has various methods
  i. Informal approach
  ii. Formal approach
  iii. Object-oriented approach
  iv. Structure analysis and design techniques
  v. Problem statement language
  vi. Entity relationship modeling
- A DFD is a simple graphical notation that can be used to represent a system in terms of the input data to the system.
- ER diagram is based on perception of a real world that consists of basic objects called entities and of relationship among these objects.
- SRS is a specification for a particular software product program or set of programs that performs certain function in a specific requirement.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.8. What are the different approaches for information modeling ? Explain each in brief.

Answer

Following are various approaches used for information modeling :

1. **Informal approach :**
   a. In informal approach, no methodology is used. In this approach no formal system model is built.
   b. The problem and the system models are essentially built into mind of analyst and directly translated from the mind of the analyst to the SRS.
2. **Formal approach / structured approach :**
   a. In structured analysis techniques, we use functional based decomposition while modeling the problem.

b. It focuses on the function performed in the problem domain and data consumed and produced by these functions.

3. **Object-oriented modeling :**
   a. The object-oriented modeling uses approach in which problem is partitioned with respect to objects.
   b. In object-oriented modeling a system is viewed as a set of objects.
4. **Structured analysis and design techniques (SADT) :**
   a. Structured analysis and design techniques (SADT) is designed for information processing systems.
   b. A model in SADT is a hierarchy of diagrams that supports a top-down approach for analysis and specification.
   c. Its specification language is a combination of graphical language and natural language.
5. **Problem statement language (PSL) / Problem statement analyzer (PSA) :**
   a. The problem statement language is designed to specify the requirements of information systems. It is a textual language.
   b. In PSL, the system input/output flow deals with the interaction of data system with the environment and specifies all the inputs received and all the outputs produced.
   c. PSA operates on the database of information collected from the PSL description of requirements.
6. **Entity relationship modeling :**
   a. If the application is a database application, the entity relationship (ER) approach can be used effectively for modeling some parts of problem.
   b. The main focus of ER modeling is the data items in the system and the relationship between them.
   c. It aims to create conceptual schema for the data from user perspective.

Que 2.9. **What is a data flow diagram ? Explain rules for drawing good data flow diagrams.**

Answer

1. A data flow diagram (DFD) is a graphical representation of the flow of data through a system.
2. A DFD also known as 'bubble chart' has the purpose of clarifying system requirements and identifying major transformations.
3. It is a graphical tool because it presents a picture.

4. The DFD may be used to represent a system or software at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

Four simple notations are used to complete a DFD which are as follows :

i. **Data flow :** The data flow is used to describe the movement of information from one part of the system to another part. Flows represent data in motion. It is a pipeline through which information flows.

ii. **Process :** A circle or bubble represents a process that transforms incoming data to outgoing data. Process shows a part of the system that transforms inputs into outputs.

iii. **External entity :** A square defines a source or destination of system data. External entities represent any entity that supplies or receive information from the system but is not a part of the system.

iv. **Data store :** The data store is used to collect data at rest or a temporary repository of data. It is represented by open rectangle.

General guidelines and rules for constructing DFDs :

1. Choose meaningful names for processes and external entities.
2. Number the processes.
3. Avoid complex DFD.
4. Remember that a DFD is not a flow chart.
5. All names should be unique.
6. Make sure the DFD is internally consistent.
7. Processes are always running, they do not start or stop.
8. All data flows are named.
9. Numbers of processes at each level should be small (not more than 7).
10. A process numbered in a DFD is leveled into n processes then each new process is numbered $p.1, p.2, \ldots\ldots p.n$, respectively.
11. All data flows entering or outgoing from context DFD process should also be maintained in its leveled DFD.
12. It should not contain loops.
13. It should not contain crossing lines.

Que 2.10. **Develop a level-1 DFD for the library management system.**

OR

What is a data flow diagram ? Explain rules for drawing good data flow diagrams with the help of suitable example.

AKTU 2013-14, 2016-17; Marks 10

Answer

Data flow diagram and its rule : Refer Q. 2.9, Page 2–11C, Unit-2.

Example : Level-1 DFD for library management system :

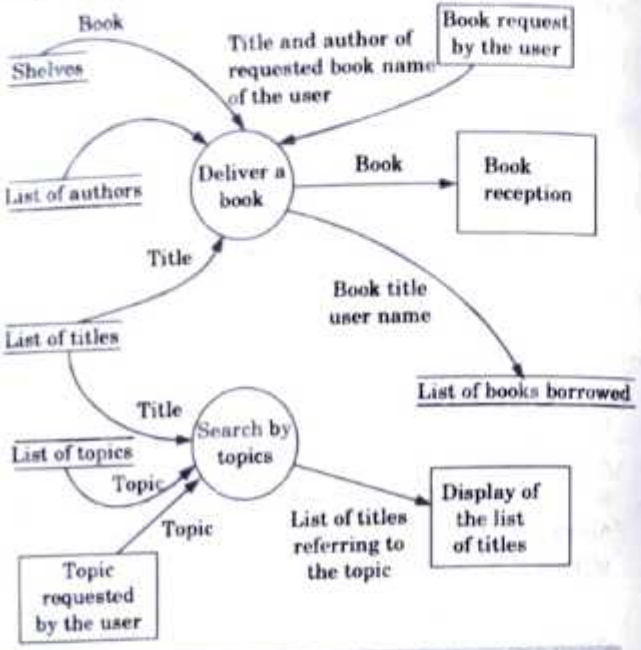


Fig. 2.10.1. A DFD describing a simplified library information system.

In this DFD, we have assumed two processes :

1. Deliver a book :
   a. User requests a book.
   b. Retrieve the data from database *i.e.*, shelves, list of authors and list of titles.
   c. Book issue and update the borrow list.
2. Search by topics :
   a. User requests a topic.
   b. Retrieve data from database *i.e.*, list of titles and list of topics.
   c. Display the list of titles which are referring that topic.

Que 2.11. What is a flow chart ? How is the flow charting techniques useful for software development ?

AKTU 2014-15, Marks 05

Answer

A flow chart is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows.

This diagrammatic representation illustrates a solution model to a given problem.

Flow charting technique is useful for software development in following ways :

1. **Synthesis :** Flow charts are used as working models in designing new program and software systems.
2. **Documentation :** Program documentation consists of activities, such as collecting, organizing, storing, and maintaining all related records of a program.
3. **Coding :** Flow charts guide the programmer in writing the actual code in a high level language, which is supposed to give an error-free program developed expeditiously.
4. **Debugging :** The errors in a program are detected only after its execution on a computer. These errors are called bugs and the process of removing these errors is called debugging. In the debugging process, a flow chart acts as an important tool in detecting, locating, and removing bugs from a program.
5. **Communication :** A flow chart is a pictorial representation of a program. Therefore, it is an excellent communication technique to explain the logic of a program to other programmers/people.
6. **Analysis :** Effective analysis of a logical problem can be easily done with the help of a related flow chart.
7. **Testing :** A flow chart is an important tool in the hands of a programmers, which helps him in designing the test data for systematic testing of programs.

Que 2.12. Distinguish between a data flow diagram and a flow chart with example.

AKTU 2012-13, Marks 05

Answer

S. No.	Data flow diagram (DFD)	Flow chart
1.	Data flow diagram presents the flow of data.	Flow chart presents steps to complete a process.
2.	Data flow diagram describes the path of data from external source to internal store or vice versa.	Flow chart does not have any input from or output to external source.
3.	The processing of data is taking place in a particular order or several process are taking place simultaneously is not described by a data flow diagram.	The timing and sequence of the process is aptly shown by a flow chart.
4.	Data flow diagram defines the functionality of a system.	Flow chart shows how to make a system function.
5.	Data flow diagram is used to describe the path of data that will complete that process.	Flow chart is used in designing a process.
6.	Types of data flow diagrams are as follows : a. Physical data flow diagram b. Logical data flow diagram	Types of flow chart are as follows : a. System flow chart b. Data flow chart c. Document flow chart d. Program flow chart

DFD example for student administration :

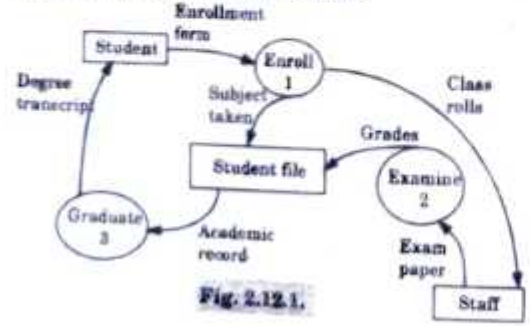


Fig. 2.12.1.

Flow chart example for software installation :

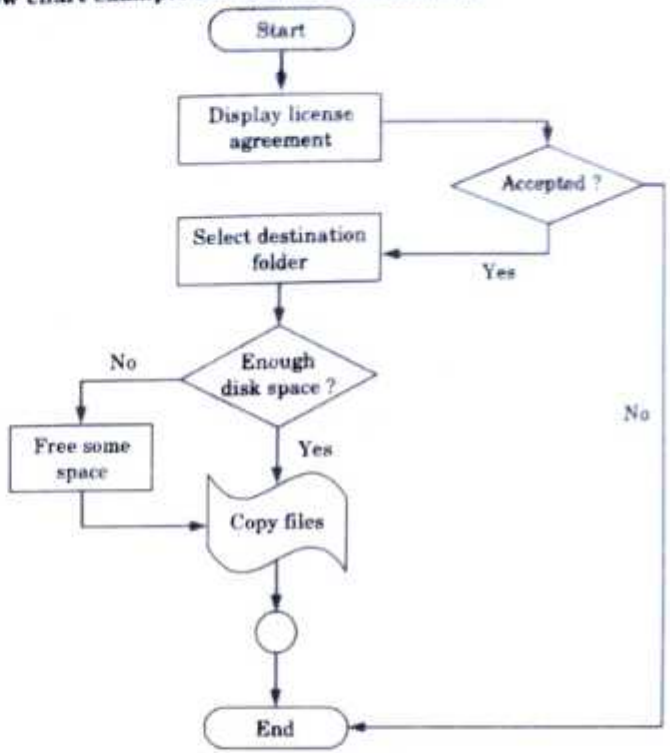


Fig. 2.12.2.

Que 2.13. What is entity relationship model ? Describe components of ER model.

Answer

Entity relationship model diagram :

It is a detailed logical representation of the data for an organization and uses three main constructs *i.e.*, data entities, relationships, and their associated attributes.

1. **Entities :**

   An entity is a fundamental thing of an organization about which data may be maintained. An entity has its own identity, which distinguishes it from other entity.

2. **Relationships :**

   A relationship is a reason for associating two entity types.

3. **Attributes :**

Each entity type has a set of attributes associated with it. An attribute is a property of characteristic of an entity that is of interest to the organization. Following are some typical entity types and their associated attributes :

STUDENT : Student_ID, Student_Name, Phone_Number

EMPLOYEE : Employee_ID, Employee_Name, Address

Components of ER diagram :

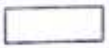
S. No.	Component name	Symbol	Description
1.	Rectangles		Represents entity sets
2.	Ellipses		Represents attributes
3.	Diamonds		Represents relationship sets
4.	Lines		Links attributes to entity sets and entity sets to relationship sets
5.	Double ellipses		Represents multivalued attributes
6.	Dashed ellipses		Represents derived attributes
7.	Double rectangles		Represents weak entity sets
8.	Double lines		Represents total participation of an entity in a relationship set

Fig. 2.13.1. Components of E-R diagram.

Que 2.14. **What are the advantages and disadvantages of ER diagram ?**

Answer

Advantages of ER diagram :

i. It is easy and simple to understand with minimal training. Therefore the model can be used by the database designer to communicate design to the end user.

ii. It has explicit linkages between entities.

iii. In the ER model it is possible to find a connection from one node to all the other nodes.

Disadvantages of ER diagram :

i. Limited constraint representation.

ii. Limited relationship representation.

iii. No representation of data.

iv. Loss of information.

Que 2.15. **How is a data dictionary useful during software development ?** **AKTU 2012-13, Marks 05**

OR

What is a data dictionary definition ?

Answer

i. **Data dictionary :**

a. Data dictionaries are simple repositories to store information about all data items defined in DFDs.

b. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developer use the same definitions and terminologies.

c. Typical information stored includes :

1. The name of the data item is self-explanatory.
2. Aliases include other names by which this data item is called for example, DEO for Data Entry Operator and DR for Deputy Registrar.
3. Description/Purpose is a textual description of what the data item is used for or why it exists.
4. Related data items capture relationships between data items for example, total_marks must always equal to internal_marks plus external_marks.
5. Range of values records all possible values, for example, total marks must be positive and between 0 to 100.
6. Data flows capture the names of the processes that generate or receive the data item.
7. If data item is primitive, then data structure definition/form captures the physical structure of the data item.
8. If the data is itself a data aggregate, then data structure definition/form captures the composition of the data items in terms of other data items.

d. The mathematical operators used within the data dictionary are defined in Table 2.15.1.

Table 2.15.1. Data dictionary notation and mathematical operators.

Notation	Meaning
$x = a + b$	x consists of data elements a and b
$x = [a/b]$	x consists of either data element a or b
$x = a$	x consists of an optional data element a
$x = y\{a\}$	x consists of y or more occurrences of data element a
$x = \{a\}z$	x consists of z or fewer occurrences of data element a
$x = y\{a\}z$	x consists of some occurrences of data element a which are between y and z

e. The data dictionary is used to :
   i. Create an ordered listing of all data items.
   ii. Create an ordered listing of a subset of data items.
   iii. Find a data item name from a description.
   iv. Design the software and test cases.

Que 2.16. What is data modeling ?

Answer

1. Data modeling is the formalization and documentation of existing processes and events that occur during application software design and development.
2. Data modeling techniques and tools capture, and translate complex system designs into easily understood representation of the data flows and processes, creating a blueprint for construction and/or re-engineering.
3. A data model can be thought of as a diagram or flowchart that illustrates the relationships between data.
4. There are several different approaches to data modeling, including :
   a. **Conceptual data modeling :** It identifies the highest-level relationships between different entities.
   b. **Enterprise data modeling :** It is similar to conceptual data modeling, but addresses the unique requirements of a specific business.
   c. **Logical data modeling :** It illustrates the specific entities, attributes and relationships involved in a business function and serves as the basis for the creation of the physical data model.
   d. **Physical data modeling :** It represents an application and database specific implementation of a logical data model.

Que 2.17. What is decision table and decision tree ?

Answer

Decision tables :
1. A decision table is a tabular representation of conditions and actions.
2. It is used when the process logic is very complicated and involves multiple conditions.
3. It is not possible to represent the process logic efficiently with structural English.
4. The main parts of decision tables are :
   a. **Conditions stubs :** It lists all the conditions relevant to the decision.
   b. **Action stubs :** It lists all the possible actions that will take place for a valid set of conditions.
   c. **Rules :** It specifies which set of conditions will trigger which action.

Decision tree :
1. A decision tree is a graph that uses a branching method to illustrate every possible outcome of a decision.
2. Decision trees are similar to decision tables, but they are more graphical and they resemble a tree.
3. It is drawn sideward, with the root of the tree at the left-hand and the leaves at the right hand.
4. There are three types of nodes in a decision tree which are :
   a. Decision nodes
   b. Chance nodes
   c. Terminal nodes

Que 2.18. Define the decision table. Discuss the difference between decision table and decision tree. **AKTU 2014-15, Marks 05** **AKTU 2015-16, Marks 10**

Answer

Decision table : Refer Q. 2.17, Page 2–20C, Unit-2.
Difference :

S. No.	Decision table	Decision tree
1.	We cannot call decision tree from decision table.	We can call decision table from a decision tree.
2.	But in a decision tree we cannot include more than one 'Or' condition.	In decision table we can include more than one 'Or' condition.
3.	Decision tree is being used whenever there are more numbers of properties.	Decision table is being used whenever there are small numbers of properties and for simple logic.

Que 2.19. **Develop a context-level model for the library system.**

Answer

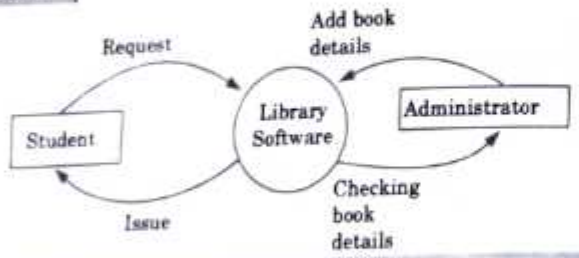


Fig. 2.19.1. Context-level model for library management system.

In this DFD, we have assumed a process library software which perform following activities :

a. Student requests a book.

b. Book issue to student.

c. Administrator can add and check the book details.

Que 2.20. **Draw a DFD for result preparation automation system of B. Tech. course of any university. Clearly describe the working of that system. Also, mention all assumptions made by you.**

Answer

Level-n DFD

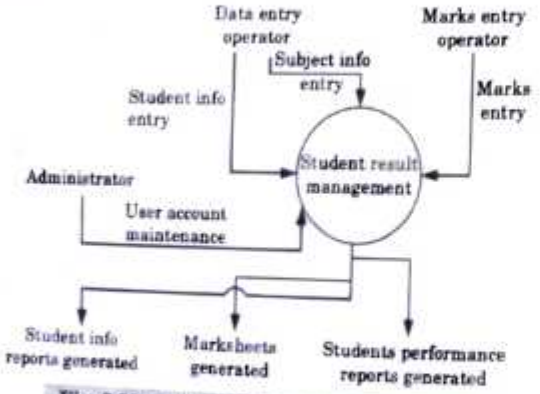


Fig. 2.20.1. 0-Level DFD or context diagram of result management system.

The Level 1 DFD is given below :

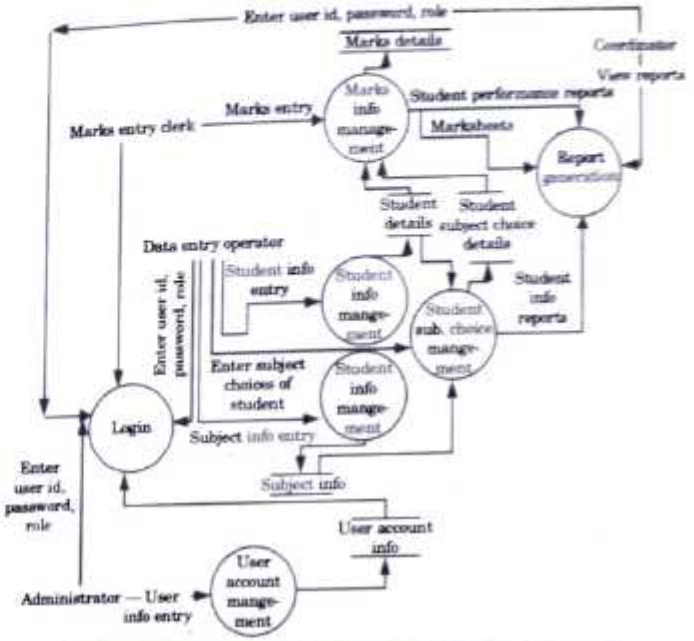


Fig. 2.20.2. Level 1 DFD of result management system.

In this DFD, we have assumed following process :

a. **Marks information management :**

   i. Clerk insert marks of student in marks detail database.

   ii. Retrieve student performance report according to database *i.e.*, marks detail, student detail and student subject choice detail.

b. **Student information management :**

   i. Data entry operator inserts the student info entry and maintain the student detail database.

c. **Subject information management :**

   i. Data entry operator inserts the subject info entry and maintain the subject info database.

d. **Student subject choice management :**

   i. Data entry operator inserts the subject choices of student and maintain student subject choice detail database.

ii. Retrieve the information from the database *i.e.*, subject info, and student detail.

iii. Give the student info report to report generation process.

e. Report generation :

i. Insert the student info report, student performance report, and marksheet.

ii. Report of student will be generated.

f. Login :

i. Insert user id, password, role for login successfully.

ii. All entries are matched from the database *i.e.*, user account info.

g. User account management :

i. Administrator enters user info entry in database *i.e.*, user account info.

Que 2.21. **Why requirements are hard to elicit ? Explain requirements elicitation technique; use case diagram using example of banking system.** **AKTU 2013-14, Marks 10**

Answer

Because of following reasons requirements are hard to elicit :

1. Lack of technical knowledge and unawareness of the technical aspects of the system from the stakeholder's side.
2. Some times stakeholders may demand unrealistic things and they do not know what they are exactly expecting from the system.
3. Stakeholders express their requirements in the most general terms, it is difficult to find the technical aspects of the system from the general terms.
4. Different people expect different services from the proposed system requirements, engineer has to discover the good sources of requirements and commonalities.
5. Business procedures, political influences and the budget matters where the clear analysis is required.

The use case diagram for banking system explaining the requirements elicitation technique is as follows :

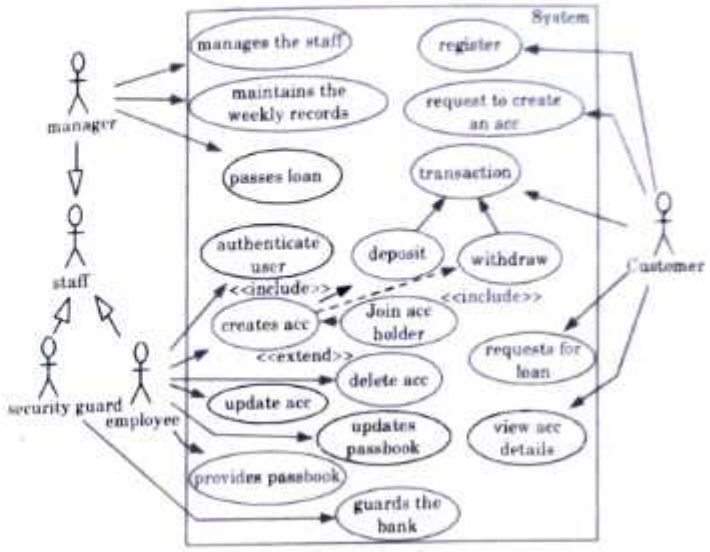


Fig. 2.21.1.

Que 2.22. **What is Software Requirement Specification (SRS) document ? Describe the goals of SRS. Also, give its importance.**

OR

Write short note on SRS document. **AKTU 2013-14, Marks 05**

Answer

1. Software requirement specification is a communication tool between stakeholders and software designer.
2. The SRS is a specification for a particular software product, program or set of programs that performs certain functions in a specific environment.
3. It serves a number of purposes depending on who is writing it.
4. First, the SRS could be written by the customer of a system.
5. Second, the SRS could be written by a developer of the system.
6. The two scenarios create entirely different situations and establish entirely different purpose for the document.
   a. First case, SRS is used to define the needs and expectations of the users.
   b. The second case, SRS is written to serve as a contract document between customer and developer.

Goals of SRS document :

1. Feedback to customer.
2. Problem decomposition.
3. Input to design specification.
4. Production validation check.

Importance of software requirements specification :

1. An SRS document minimizes the time and effort required by developer to achieve desired goals and also minimizes the development cost.
2. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations.
3. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated.

Que 2.23. Discuss the advantages of Software Requirements Specification (SRS) documents. Why is SRS known as the black box specification of a system ?

Answer

Advantages of SRS document :

1. A SRS document provides a reference for validation of the final product.
2. A high quality SRS document is a prerequisite to high quality software.
3. A high quality SRS document reduces the development cost.
4. Reduce the development effort.
5. Provide a basis for estimating costs and schedules.
6. Serve as a basis for enhancement.
7. Provide a baseline for validation and verification.

Black box specification :

1. SRS document should only specify what the system should do and refrain from stating how to do these.
2. This means that the SRS document should specify the external behaviour of the system and not discuss the implementation issues.
3. This view with which a requirement specification is written has been shown in Fig. 2.23.1.
4. In Fig. 2.23.1 the SRS document describes the output produced for the different types of input and a description of the processing required to produce the output from the input (shown in ellipses) and the internals of the system (shown as a rectangle) are not discussed at all.

5. The SRS document should view the system to be developed as a black box, and should specify the externally visible behaviour of the system.
6. For this reason, the SRS document is also called as black box specification of a system.

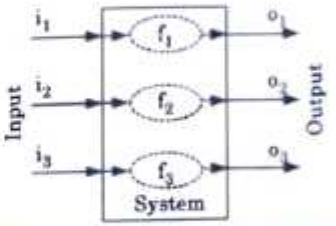


Fig. 2.23.1. The black box view of a system as performing a set of functions.

Que 2.24. Discuss the organization of a SRS. Also, write important issues of this organization.

OR

Define IEEE standard for SRS. **AKTU 2013-14, Marks 05**

Answer

Organization of SRS :

i. Organization of the SRS document depends to a large extent on the system analyst himself, he is often guided by the policies and standards followed by the company.

ii. Also, the organization of the document and the issues to a large extent depend on the type of the product being developed.

iii. However, irrespective of the company principles and product type, the three basic issues that any SRS document should discuss are :
   a. Functional requirements,
   b. Non-functional requirements, and
   c. Guidelines for system implementation.

IEEE standard of SRS :

i. **Introduction**
   a. Purpose
   b. Document conventions
   c. Intended audiences
   d. Additional information
   e. Contact information/SRS team members
   f. References

ii. **Overall description**
   a. Product perspective
   b. Product function
   c. User class and characteristics
   d. Operating environment
   e. User environment
   f. Design/implementation constraints
   g. Assumptions and dependencies

iii. **External interface requirement**
   a. User interface
   b. Hardware interface
   c. Software interface
   d. Communication protocols and interfaces

iv. **System features**
   a. System features part A :
      1. Description and priority
      2. Action/result
      3. Functional requirements
   b. System features part B :
      1. Description and priority
      2. Action/result
      3. Functional requirements

v. **Other non-functional requirements**
   a. Performance requirements
   b. Safety requirements
   c. Security requirements
   d. Software quality attributes
   e. Project documentation
   f. User documentation

vi. **Other requirements**

Appendix A : Terminology/Glossary/Definitions list

Appendix B: To be determined

Important issues of organization :

1. Without developing the SRS document, the system would not be implemented according to customer needs.
2. Software developers would not know whether what they are developing is what exactly required by the customer.

3. Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
4. It will be very much difficult for user document writers to write the user's manuals properly without understanding the SRS document.

Que 2.25. List five desirable characteristics of good SRS document.

Answer

Software requirement specification has following main quality characteristics :

i. **Complete :**
   a. SRS should be complete.
   b. It defines all situations that will be encountered and system capability to address them.

ii. **Consistent :**
   a. It should be reliable.
   b. Its capability function and performance level should be well-matched.

iii. **Accurate :** It should specify the system capability in real world environment and its interaction with real world environment.

iv. **Modifiable :** Logical and hierarchical structure of SRS should facilitate easy modification on requirement.

v. **Valid :** A valid SRS is one to which developer and customer can understand well that is why it is written in natural language.

vi. **Verifiable :** A verifiable SRS is reliable from one level of abstraction to another.

Que 2.26. What are the main activities of requirements analysis and specification ? Also discuss the desirable characteristics of good SRS document. **AKTU 2012-13, Marks 10**

Answer

Activities : The main activities carried out during requirements analysis and specification phase are as follows :

1. Requirements gathering and analysis
2. Requirements specification

Characteristics : Refer Q. 2.25, Page 2–28C, Unit-2.

Que 2.27. List five desirable characteristics of good SRS document. Discuss the relative advantages of formal and informal requirements specifications. **AKTU 2014-15, Marks 05** **AKTU 2016-17, Marks 10**

Answer

Characteristics : Refer Q. 2.25, Page 2–28C, Unit-2.

Advantages of formal requirement :

1. Formal requirement specifications will ensure that the customer gets what they want. If requirement are set up for every possibility, there will be no need for a programmer to guess at what their customers would like to see
2. Just as specification help the customer get what they want from the programmer, they also help protect the programmer from requirements changes from the customer. These types of changes can result in entire projects needing to be restarted from scratch.
3. Laying out formal requirement specifications before beginning implementation will also help to avoid many arguments about changing requirements during the implementation process.

Advantages of informal requirement :

1. It does not require much technical skill.
2. It requires understanding of what the content and scope of the each section of the requirement documents must be.
3. The specification can be accessible and readable to customer, design engineer, test engineer, and maintenance personnel.

Que 2.28. List and discuss the major quality requirement of a software requirement specification document ?

Answer

SRS document should document the following :

1. **Functional requirements of the system :**
   a. The functional requirements are the services which the end user expects the final product to provide.
   b. The functional requirements of the system should clearly describe each of the functions that the system needs to perform along with the corresponding input and output data set.
2. **Non-functional requirements of the system :**
   a. Non-functional requirements deal with the characteristics of the system that cannot be expressed functionally.
   b. These are the constraints imposed on the system and deal with issues like maintainability, security, performance etc.
   c. The non-functional requirements also include reliability issues, accuracy of results, human-computer interface issues, operating and physical constraints etc.

3. **Constraints on the system :**
   The constraints on the functions of a system may contain desirable things that the system should or should not do. A constraint can be classified as follows :
   a. Interface constraints.
   b. Performance constraints : response time, security, reliability, storage space etc.
   c. Operating constraints.
   d. Life cycle constraints : maintainability, reusability, portability, flexibility etc.
   e. Economic constraints.

PART-3

Software Quality Assurance (SQA) : Verification and Validation, SQA Plans, Software Quality Frameworks, ISO 9000 Models, SEI-CMM Model.

CONCEPT OUTLINE : PART-3

- SQA is the quality that ensures customer satisfaction by offering all the customer deliverables on performance and standards.
- Validation is "Are we building the right product".
- ISO 9000 : It is a certification which serve as a reference for contact between independent parties.
- SEI CMM model is a strategy for improving the software process, irrespective of actual life cycle model used.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 2.29. Discuss verification and validation and their objectives.

Answer

1. Software verification is a set of activities that ensure that software correctly implements a specific function *i.e.*,
   "Are we building the product right".
2. Requirement validation is a set of activities that ensure that the software that is going to be build is traceable to customer requirements *i.e.*

"Are we building the right product".

3. Traditionally software verification and validation is defined as system engineering methodology to ensure that quality is built into software during development.
4. The analysis and test activities performed by verification and validation (V & V) evaluate and assess the software products and the development processes during each software life cycle phase in parallel with, not after the completion of, the development effort.
5. This provides early identification of errors, improving software performance, identification of program risks.
6. V & V attacks on two of major contributor of software failure :
   i. Incorrect or missing requirements and
   ii. Poor organization in software architecture.
7. It also acts as powerful risk management tool.

Objectives of verification and validation :

1. Determine that it performs its intended functions correctly.
2. Ensure that it performs no unplanned functions.
3. Measure and assess the quality and reliability of software.
4. Ensure that quality is built into the software and that the software will satisfy the software requirements.
5. Satisfy the user that system is being developed according to standards and specifications.

Que 2.30. Write a short note on software quality assurance.

AKTU 2013-14, Marks 05

Answer

1. Software Quality Assurance (SQA) is a process that ensures that developed software meets and complies with defined or standardized quality specifications.
2. SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets desired quality measures.
3. SQA helps to ensure the development of high-quality software.
4. SQA practices are implemented in most types of software development, regardless of the underlying software development model being used.
5. SQA incorporates and implements software testing methodologies to test software.
6. Rather than checking for quality after completion, SQA processes test for quality in each phase of development until the software is complete.

7. With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards.

Que 2.31. Explain Software Quality Assurance (SQA) with life cycle.

AKTU 2016-17, Marks 10

Answer

In each phase of SDLC the SQA activities are as follows :

1. **Software initiation phase :**
   a. SQA should be involved in writing and reviewing the management plan to ensure that processes, procedures and standards identified in the plan are appropriate, clear, accurate and verifiable.
   b. During this phase, also SQA provides the quality assurance management plan.
2. **Software requirements phase :**

   During this phase, SQA makes sure that software requirements are complete and testable, and properly expressed as functional, performance and interface requirements.
3. **Software design phase :**

   SQA activities during the design phase include :
   a. To ensure compliance with approved design standards as specified in the management plan.
   b. Assuring all software requirements are covered in software components.
   c. To ensure interface control documents are consistent with the standard form and content.
   d. To ensure that allocated modules are included in the detailed design.
4. **Software implementation phase :**

   SQA activities during the phase of implementation include checking :
   a. Results of coding and design activities including the schedule contained in the software development plan.
   b. Status of all deliverables.
   c. The activities of configuration management and software development library.
5. **Software integration and test phase :**

   SQA activities during the integration phase and testing include :
   a. Test preparation and check list of all the testing deliverables.
   b. Ensure that all tests are performed according to test plans and procedures.

c. Ensure that test reports are complete and accurate.

d. Certifying that the test is completed and the software and documentation are ready for delivery.

6. **Software operations and maintenance phase :**

During this phase, there will be mini-development cycles to improve or correct software production related bugs reported by the end users.

Que 2.32. What do you understand by the term SQA plans ?

Answer

1. SQA plan is a standard activity which ensures the quality goals of the organization.
2. It gives details and templates on all activities, which have become part of the standard implementation.

The SQA plan include the following :

1. Project planning
2. Models of data, classes and objects, processes, design architecture
3. SRS
4. Test plans for testing SRS
5. User manuals, online help
6. Audit and review

In SQA plan mainly three kind of activities are performed :

1. Organization specific
2. Software specific
3. Customer specific

Que 2.33. What is software quality ? Write the various attributes of software quality.

Answer

Refer Q. 1.10, Page 1–11C, Unit-1.

Que 2.34. In a software development organization, identify the persons responsible for carrying out the quality assurance activities. Explain the principal tasks they perform to meet this responsibility. **AKTU 2015-16, Marks 10**

Answer

Table 2.34.1 shows the activities carried out for the quality assurance along with the person responsible for each respective activity.

Table 2.34.1.

S. No.	Activities	Person responsible
1.	Inspection	Any responsible person who knows about the project
2.	Reviews	Project manager
3.	Development of standards, procedures and guidelines	Software developer
4.	Auditing	Auditors

Principle tasks performed to enhance software quality are :

1. **Inspections :** Inspection means a careful check for the quality of product. It is done when a piece of work is completed, copies of work are distributed to co-workers who then examine the work to note defects. A meeting is held to discuss work and defects requiring to rework.
2. **Reviews :** Review is a QA activity that checks the quality of project deliverables. This involves examining the software, its documentation and records of process to discover errors. Reviewing is a public process of error detection.
3. Development of standards, procedures and guidelines etc.
4. **Auditing :** Production of reports for the top management summarizing the effectiveness of the quality system in the organization.

Que 2.35. What do you understand by ISO 9000 certification ? Give its benefits and limitations.

Answer

ISO 9000 certification :

1. ISO 9000 is a set of international standards on quality management and quality assurance to help companies to maintain an efficient quality system.
2. ISO 9000 specifies a set of guidelines for repeatable and high quality development.
3. It is important to realize that ISO 9000 standard is a set of guidelines for the production process and is not directly concerned with product itself.
4. **ISO 9000 is a series of three standards :** Based on type of software industries there are different types of ISO standard, which are as follows :

a. **ISO 9001** : This standard applies to the organizations engaged in design, development, production and servicing of goods. This is the standard that is applicable to most software development organizations.

b. **ISO 9002** : This standard applies to those organizations which do not design products but are only involved in production. This includes steel industry who buy the products from external sources and only involved in manufacturing those products. Therefore, ISO 9002 is not applicable to software development organizations.

c. **ISO 9003** : This standard applies to organizations involved only in installation and testing of the product.

Benefits of ISO 9000 certification :

1. Continuous improvement.
2. Eliminate verification.
3. Higher real and perceived quality.
4. Boost employee morale.
5. Improved customer satisfaction.

Limitations of ISO 9000 certification :

1. It does not automatically lead to total quality management (TQM) *i.e.*, continuous improvement.
2. It does not provide any guideline for defining an appropriate process.
3. It is not full proof and thus variation in the certification norms may exist.

Que 2.36. What are the various ISO 9001 requirements ?

Answer

The main requirements of ISO 9001 as related to software development are as follows :

1. **Management responsibility :**
   a. Management must have an effective quality policy.
   b. The responsibility and authority of all those whose work affect quality must be defined and documented.
2. Quality system.
3. Contract reviews.
4. **Document control :**
   a. There must be proper procedure for document approval, issue and removal.
5. Product identification.
6. Process control.
7. Inspection and testing.

8. **Inspection, measuring and test equipment** : If inspection, measuring and test equipment are used, they must be properly maintained and elaborated.
9. **Handling** : Handling deals with the storage, packing and delivering of the software product.
10. Quality audits.
11. Training.

Que 2.37. Given software product and its requirement specification document, explain how would you design the system test suite for this software product ? AKTU 2014-15, Marks 10

Answer

1. For a given software requirement specification and a software product, to facilitate development, we need to design a collection, or suite, of test cases that cover all the modules and functions in the system.
2. If the software we are writing is an extension of an existing system, the existing code base should ideally contain a complete suite of good unit tests. If it does not, our first step is to complete the test suite for the existing system. When we build the test suite for the new system, it will be anchored in a reliable initial test suite.
3. The pre-existence of a test suite provides programmers with "live" goals for the code to achieve as they write, along with a suite of realistic examples that can test the validity of the new code itself.
4. A test suite is a collection of "unit tests." The suite should have one unit test for every module or class in the code base and one unit test for each use case.
5. A unit test should aim for 100% code coverage and 100% use case coverage. That is, every line of code in each module being tested should be exercised by at least one call in the unit test. Moreover, every step of each use case being tested should also be covered.
6. Designing a unit test for a domain class or database module begins by writing at least one call for every function and constructor in that class or module.
7. Designing a unit test for a user interface module must begin with a test for every user option that appears on the form. Usually, these options appear as buttons, means, textboxes, or other widgets common to an interactive web page.
8. To unit test a use case, we begin by indentifying as a "unit" those user interface elements *i.e.*, forms and associated modules that combine to implement that use case.

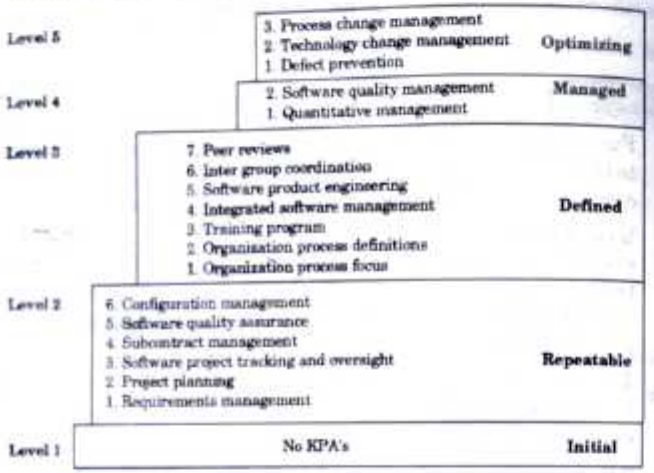
Que 2.38. **Write a detailed note on SEI CMM.**

OR

What are different CMM levels ? What does CMM level specify ? Explain briefly. **AKTU 2013-14, Marks 05**

Answer

1. The capability maturity model (CMM) is a strategy for improving the software process, irrespective of actual life cycle model used.



2. CMM is used to judge the maturity of the software processes of an organization and to identify the key practices that are required to increase the maturity of these processes.
3. The model is based on best practices followed in the organizations world wide.

The five CMM level of maturity are :

i. **Level 1 : Initial :**

a. At the level 1, the processes followed by the organizations are not well defined.

b. Processes are immature. As a result, a stable environment is not available for software development.

c. Further, success and failure of any project depends on the team member's competence.

d. There is no basis for predicting product quality, time for completion etc. No Key Process Areas (KPA's) are defined at level 1.

ii. **Level 2 : Repeatable :**

a. The level 2 focuses on establishing basic project management policies.

b. The managers are able to predict the cost and schedule of the project based on earlier experience. At this level, organization can be called a disciplined and organized organization.

c. The key process areas to achieve this level of maturity are :

1. Requirement management
2. Project planning
3. Software project tracking and oversight
4. Subcontract management
5. Software quality assurance
6. Configuration management

iii. **Level 3 : Defined :**

a. At this level, standard process to be used across the organization are defined and documented.

b. Standardization of organization wide processes helps the manager and other team members to perform efficiently.

c. The key processes associated with level 3 are :

1. Organization process focus
2. Training program
3. Organization process definition
4. Integrated software management
5. Software product engineering
6. Inter group coordination
7. Peer reviews

iv. **Level 4 : Managed :**

a. At this level of CMM, quantitative quality goals are set for the organization for the software product as well as software processes.

b. Here quality and productivity of the process are measured and maintained by organization wide software process database and feedback is given to management.

c. The key processes associated at management level are :

1. Quantitative process management
2. Software quality management

v. **Level 5 : Optimizing :** It focuses on continuous process improvement in organization using quantitative feedback. The main features at level 5 are :

1. Defect prevention
2. Technology change management
3. Process change management

Que 2.39. Discuss the merits of SEI-CMM based quality assessment.

Answer

Merits of SEI-CMM are :

1. CMM method provides an effective and objective way of comparing different organizations against each other for potential risks in their ability to perform on a contract for a sponsoring organization.
2. Lower costs in implementing of CMM.
3. Improved customer satisfaction and their demands.
4. Establishment of a transparent internal organization that is clear to all employees.
5. Provides for a very objective view.
6. The organization being assessed does not require any personnel to go through additional training as part of the assessment.
7. The organization needs to hire only two people from a consulting organization to perform the assessment so the cost can be significantly reduced.
8. The results of the assessment are highly accurate because the methodology requires great care and consistency in determining the satisfaction or weaknesses for each part of the model.

Que 2.40. Difference between the following :

i. **Verification and validation**
ii. **ISO and SEI-CMM models**

OR

Compare ISO and SEI-CMM model. **AKTU 2014-15, Marks 05** **AKTU 2016-17, Marks 10**

Answer

i. **Verification and validation :**

S. No.	Verification	Validation
1.	Am I building the product right ?	Am I building the right product ?
2.	Verification is a static practice of verifying documents, design, code and program.	Validation is a dynamic mechanism of validating and testing the actual product.
3.	It does not involve executing the code.	It always involves executing the code.
4.	It is low level activity.	It is higher level activity.
5.	Verification is carried out by quality assurance team.	Validation is carried out by testing team.
6.	Verification uses methods like inspections, reviews, walkthroughs etc.	Validation uses methods like black box testing, white box testing etc.
7.	Verification is to check whether the software conforms to specifications.	Validation is to check whether software meets the customer expectation and requirements.

ii. **ISO and SEI-CMM models :**

S. No.	ISO	SEI-CMM
1.	It applies to any type of industry.	CMM is specially developed for software industry.
2.	ISO 9000 addresses corporate business process.	CMM focuses on the software Engineering activities.
3.	ISO 9000 specifies minimum requirement.	CMM gets into technical aspect of software engineering.
4.	ISO 9000 restricts itself to what is required.	It suggests how to fulfill the requirements.
5.	ISO 9000 provides pass or fail criteria.	It provides grade for process maturity.
6.	ISO 9000 has no levels.	CMM has five levels.
7.	ISO 9000 does not specify sequence of steps required to establish the quality system.	It reconnects the mechanism for step by step progress through its successive maturity levels.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Discuss requirement engineering.

Ans. Refer Q. 2.1.

Q. 2. Write short note on data flow diagram.

Ans. Refer Q. 2.9.

Q. 3. What is flow chart ? How is flow charting techniques useful for software development ?

Ans. Refer Q. 2.11.

Q. 4. Explain software requirement specification (SRS).

Ans. Refer Q. 2.22.

Q. 5. Discuss verification and validation.

Ans. Refer Q. 2.29.

Q. 6. Explain SQA with life cycle.

Ans. Refer Q. 2.31.

Q. 7. Explain the following :

i. ISO 9000 certification.

ii. SEI CMM model.

Ans.

i. Refer Q. 2.35.

ii. Refer Q. 2.38.

☺☺☺

3 UNIT

Software Design

PART-1

Basic Concepts of Software Design, Architectural Design, Low Level Design : Modularization, Design Structure Charts, Pseudo Codes, Flow Charts

CONCEPT OUTLINE : PART-1

- **Software design :** It is the process of inventing and selecting programs that meet the objectives for a software system.
- A system is modular if it is composed of well-defined, conceptually simple and independent units interacting through well-defined interfaces.
- A structure chart of a program is a graphic representation of its structure in which a module is represented by a box with the module name written in the box.
- Flow chart is a diagrammatic representation of the procedure for solving the problem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. What do you mean by term software design ? Also list out the general task involved and the design principles.

Answer

1. Software design is the process of inventing and selecting programs that meet the objectives for a software system.
2. Software design is the practice of taking a specification of externally observable behaviour and adding details needed for actual computer system implementation, including human interaction, task management and data management details.
3. Software design is the activity of specifying the nature and composition of a software system satisfying client needs and desires, subject to design constraints.

The general tasks involve in the design process are :

i. Design the overall system processes.
ii. Segmenting the system into smaller, compact workable modules.
iii. Designing the database structure.

iv. Specifying the details of the program to be created to achieve the desired functionality.
v. Designing the input and output documents.
vi. Designing controls for the system.
vii. Documenting the system design.

Basic design principles are :

1. Free from the suffering from "tunnel vision" : A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job.
2. The design should be traceable to the analysis model.
3. The design should not repeat the same thing.
4. The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.
5. The design should exhibit uniformity and integration.

Que 3.2. What do you mean by good software design ? Discuss the criteria for a software design to enhance the quality of software.

Answer

Good software design :

1. The definition of "a good software design" can vary depending on the application being designed.
2. For example, the memory size used by a program may be an important issue to characterize a good solution for embedded software development since embedded applications are often required to be implemented using memory of limited size due to cost, space, or power consumption considerations.
3. For embedded applications, one may sacrifice design comprehensibility to achieve code compactness.
4. Therefore, the criteria used to judge how good a given design solution is can vary widely depending upon the application.
5. However, most researchers and software engineers agree on a few desirable characteristics that every good software design for general application must possess. Those characteristics are :
   a. **Correctness :** It produces the right results and exhibits the right behaviour.
   b. **Usability :** The user finds it easy to make it perform the desired functions.
   c. **Maintainability :** It can be easily maintained and updated.
   d. **Availability :** It is available (*i.e.*, will run) when it is needed.
   e. **Useful :** It performs a task that someone wants to be performed.

Que 3.3. **Discuss the various phases of software design.**

Answer

The software design process can be decomposed mainly into the following three levels (or phases) of design :

1. **Interface design :**
   a. Interface design is the specification of the interaction between a system and its environment.
   b. This phase proceeds at a high level of abstraction with respect to the inner workings of the system.
   c. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts.
   d. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices, which are collectively called agents.
   e. Often the major task of interface design is user interface design, especially for systems with complex graphical user interfaces.
2. **Architectural design :**
   a. Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them.
   b. In architectural design the overall structure of the system is chosen, but the internal details of major components are ignored.
3. **Detailed design :**
   a. Detailed design is the specification of the internal elements of all major system components, their structure, properties, relationships, processing, and often their algorithms and data structures.
   b. Detailed design consists of a wide range of "detail" because there is usually quite a lot to fill in between the architecture and the code.

Que 3.4. **Why it is necessary to design the system architecture before specifications are completed ? What are the steps involved in design stage of a software ?**

Answer

1. The system architecture have to be designed before specifications are completed to :
   a. provide a means of structuring the specification
   b. develop different subsystem specifications concurrently
   c. allow manufacture of hardware by sub-contractors
   d. provide a model for system costing

2. Writing specification for the whole system might bring great complexity and it is difficult to formulate it.
3. Therefore, it is easier to divide the system into simpler subsystems and define their specification.
4. Hence we can concurrently develop subsystems and the specifications readily into the implementation stage.

Design stage of software : Refer Q. 3.3, Page 3–4C, Unit-3.

Que 3.5. **What do you understand by the term modularity ? Explain the important properties of a modular system.**

OR

Enlist the principles of modularity.

Answer

1. Modularity is the single attribute of software that allows a program to be intellectually manageable. It enhances design clarity, which in turn eases implementation, debugging, testing, documenting, and maintenance of the software product.
2. 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.
3. Fig. 3.5.1 establishes the relationship between cost / effort and number of modules in software.

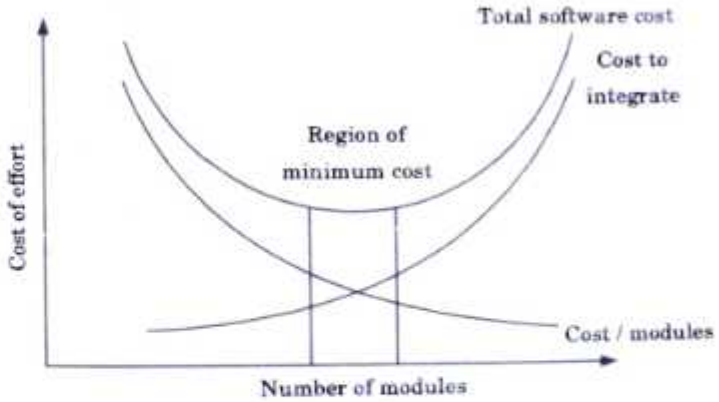


Fig. 3.5.1.

Important properties of modular system are :

1. Each module is a well-defined subsystem that is potentially useful in other applications.
2. Each module has a single, well-defined purpose.

3. Modules can be separately compiled and stored in a library.
4. Modules can use other modules.
5. Modules should be easier to use than to build.
6. Modules should be simpler from the outside than from the inside.

Principles of modularity : Certain principles must be followed to ensure proper modularity :

1. Linguistic modular units
2. Few interfaces
3. Small interfaces (weak coupling)
4. Explicit interfaces
5. Information hiding

Que 3.6. What is structure charts ? Explain in detail.

Answer

1. The structure chart of a program is a graphic representation of its structure.
2. A structure chart is a nice representation mechanism for a design that uses functional abstraction.
3. It shows the modules and their call hierarchy, the interfaces between the modules, and what information passes between modules.
4. It is a convenient and compact notation that is very useful while creating the design.
5. In a structure chart, a module is represented by a box with the module name written in the box.
6. An arrow from module A to module B represents that module A invokes module B. B is called the subordinate of A, and A is called the superordinate of B.
7. The arrow is labeled by the parameters received by B as input and the parameters returned by B as output, with the direction of flow of the input and output parameters represented by small arrows.

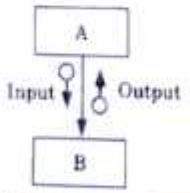


Fig. 3.6.1. The structure chart.

Que 3.7. What is structure charts ? Explain rules for drawing good structure charts diagrams with the help of a suitable example.

AKTU 2016-17, Marks 15

Answer

Structure chart : Refer Q. 3.6, Page 3-6C, Unit-3.

Rules for drawing good structure charts are :

1. Review the DFDs and object models.
2. Identify modules and relationships.
3. Add couples, loops, and conditions.
4. Analyze the structure chart, the DFDs, and the data dictionary.

Example :

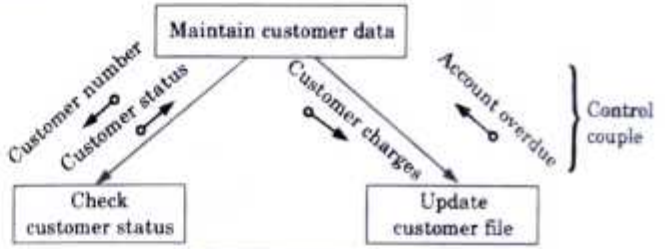


Fig. 3.7.1. Structure chart.

Que 3.8. What is pseudo-code ? Explain in detail with an example.

Answer

1. Pseudo-code is a tool that can be used to express algorithms more precisely.
2. Pseudo-code is a combination of programming code and ordinary English.
3. Pseudo-code allows algorithms to be specified precisely without having to worry about programming language syntax.
4. Pseudo-code is not actually executed on a computer, it simply provides a means for programmers to "Think out" their programs before they implement them.
5. A carefully prepared pseudo code program can be easily converted to a programming language such as C++ or Java.

Example :

Sample of a sales promotion policy :

1. Preferred customers who order more than ₹1,000 are entitled to a 5 % discount and an additional 5 % discount if they used our charge card.

2. Preferred customers who do not order more than ₹1,000 receive a ₹25 bonus coupon.
3. All other customers receive a ₹5 bonus coupon.

Pseudo-code version of the sales promotion policy :

```
If customer is a preferred customer, and
    If customer orders more than ₹1,000 then
        Apply a 5 % discount, and
        If customer uses our charge card, then
            Apply an additional 5 % discount
    Else
        Award a ₹25 bonus coupon
Else
    Award a ₹5 bonus coupon
```

Que 3.9. What do you mean by low level design ?

Answer

1. Low level design is a component level design process that follows step by step refinement process.
2. This process can be used for designing data structure, required software architecture, source code.
3. In this phase, the actual software components are designed.
4. Types of low level design :
   i. **Modularization** : Refer Q. 3.5, Page 3–5C, Unit-3.
   ii. **Structure chart** : Refer Q. 3.6, Page 3–6C, Unit-3.
   iii. **Pseudo-code** : Refer Q. 3.8, Page 3–7C, Unit-3.

PART-2

Coupling and Cohesion Measures, Design Strategies : Function Oriented Design, Object Oriented Design, Top-down and Bottom-up Design.

CONCEPT OUTLINE : PART-2

- Cohesion is a measure of the functional strength of a module whereas coupling is a measure of the degree of functional interdependence between the two modules.
- Function oriented design is the result of focusing attention to the function of the program. This is based on step wise refinement. Whereas object oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that is to be manipulated by the program.
- Top-down is a design strategy in which the problem is divided into small problems where as in bottom-up, the approach is to start at the bottom, with problems that you already know how to solve.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.10. Write a short note on coupling.

Answer

1. Coupling is the measure of the degree of independence between modules.
2. Two modules that are strongly connected, highly dependent on each other is called highly coupled module.
3. When two modules have weak interconnection between them then they are called loosely coupled modules.
4. Coupling increases with complexity of interface.

The priority of coupling from lowest to highest is given in the Table 3.10.1.

Table 3.10.1.

Data Coupling	Best
Stamp Coupling	↑
Control Coupling	↑
External Coupling	↑
Common Coupling	↑
Content Coupling	Worst

Following are the couplings which are shown here on the basis of their priority and by taking example of two modules assuming A and B :

1. **Data coupling :**
   a. Data coupling is when module passes from non-global variable to another module.
   b. Modules can only communicate through passing data elements or informational flags.

c. The two modules have no need to know what goes on inside the other module.

2. **Stamp coupling :**
   a. Stamp coupling occurs when module passes from non-global data structure or entire structure to another module.
   b. A change in data structure in module will affect all modules that use it.
   c. The two modules using stamp coupling must have some knowledge of the internal workings of other modules that use the same data structure.

3. **Control coupling :**
   a. Module A and B are said to be control coupled if they communicate by passing of control information.
   b. The sending module must know a great deal about the inner workings of the receiving module.

4. **External coupling :**
   a. A form of coupling in which a module has a dependency to other module, external to the software being developed or to a particular type of hardware.
   b. This is basically related to the communication to external tools and devices.

5. **Common coupling :**
   a. In common coupling both module A and B share the same data.
   b. With common coupling, it can be difficult to determine which module is responsible for having set of variable to a particular value.
   c. It is not a good type of coupling as if we want to modify the data then we have to check all modules who are sharing the data.

6. **Content coupling :**
   a. Content coupling occurs when one module directly refer to the inner workings of another module.
   b. Modules are highly interdependent to each other.
   c. One module can alter data in other module or change a statement coded in other module.

Que 3.11. Define the concept of module cohesion. Briefly explain the different types of cohesion. **AKTU 2012-13, Marks 10**

Answer

1. Cohesion is the measure of functional relatedness of elements within a single module.

2. When dividing a system into modules, it must ensure that the activities within the module are tightly bound to one another.

Types of cohesion :

1. **Functional cohesion :**
   a. All activities in the module are functionally related or they are performing a similar function such as interest calculation, tax calculation etc.
   b. The chances of a change request affecting more than one module are low if the modules are based on functionality. Hence, this is the best form of cohesion.

2. **Sequential cohesion :**
   a. In this scheme of cohesion, modules are divided into a series of activities such that the output of one module becomes the input to the next module and the chain continues.

3. **Communicational cohesion :**
   a. This form of cohesion relates to a situation where all modules share some common data.
   b. This form of cohesion has clearly defined boundaries, inputs and outputs.

4. **Procedural cohesion :**
   a. In this type of cohesion, activities share the same procedural implementation.

5. **Temporal cohesion :**
   a. In this type of module division, activities occurring in the same period are grouped together.
   b. The set of functions responsible for initialization, start up activities such as setting program counters or control flags associated with programs exhibit temporal cohesion.

6. **Logical cohesion :**
   a. In logical cohesion, activities belonging to the same category are grouped together.
   b. We can group all reporting activities together or all querying activities together.

7. **Coincidental cohesion :**
   a. Coincidental cohesion exists in modules that contain instructions that have little or no relationship to one another.
   b. Coincidental cohesion is to be avoided as far as possible.

Priority of cohesion from lowest to highest :

Table 3.11.1.

Functional Cohesion	Best (high)
Sequential Cohesion	
Communicational Cohesion	
Procedural Cohesion	
Temporal Cohesion	
Logical Cohesion	
Coincidental Cohesion	Worst (low)

Que 3.12. **What do you mean by the terms cohesion and coupling in the context of software design ? How are these concepts useful in arriving at the good design of a system ?** **AKTU 2014-15, Marks 10**

Answer

Cohesion : Refer Q. 3.11, Page 3–10C, Unit-3.

Coupling : Refer Q. 3.10, Page 3–9C, Unit-3.

These concepts are useful in arriving at the good design of a system because :

1. A software engineer must design the modules with goal of high cohesion and low coupling. A good example of a system that has high cohesion and low coupling is the "plug and play" feature of the computer system.
2. Various slots in the motherboard of the system simply facilitate to add or remove the various services / functionalities without affecting the entire system.
3. This is because the add-on components provide the services in high cohesive manner. Fig. 3.12.1 provides a graphical review of cohesion and coupling.

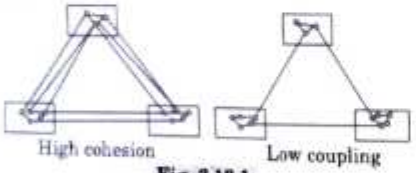


Fig. 3.12.1.

4. Module design with high cohesion and low coupling characterizes a module as black box when the entire structure of the system is described.
5. Each module can be dealt separately when the module functionality is described.

Que 3.13. **What do you understand by coupling and cohesion ? What roles they play in software design ? Describe the properties of best coupling and cohesion giving examples of each.**

AKTU 2013-14, Marks 10

Answer

Coupling : Refer Q. 3.10, Page 3–9C, Unit-3.

Cohesion : Refer Q. 3.11, Page 3–10C, Unit-3.

Roles and properties : Refer Q. 3.12, Page 3–12C, Unit-3.

Que 3.14. **Define the term software design. Also discuss the coupling in the context of software design. For a good design, the modules should have low coupling. Why ?**

AKTU 2015-16, Marks 15

Answer

Software design : Refer Q. 3.1, Page 3–2C, Unit-3.

Coupling : Refer Q. 3.10, Page 3–9C, Unit-3.

Reason : For a good design, the modules should have low coupling because high coupling among modules not only makes a design solution difficult to understand and maintain, but it also increases development efforts. High coupling also makes it very difficult to get these modules developed independently by different team members. These problems are reduced by low coupling.

Que 3.15. **What do you understand by coupling and cohesion ? What roles they play in software design ? Describe the properties of best coupling and cohesion giving examples of each.**

AKTU 2016-17, Marks 15

Answer

Coupling : Refer Q. 3.10, Page 3–9C, Unit-3.

Example : In a module "Managing-Book-lending", functions like "Issue-book", "Return-book", "Query-book" are strongly related.

Cohesion : Refer Q. 3.11, Page 3–10C, Unit-3.

Example : Customer billing system.

Que 3.16. **Describe the difference between the software engineering terms coupling and cohesion.**

AKTU 2013-14, Marks 05

Answer

S. No.	Cohesion	Coupling
1.	Cohesion is a measure of functional strength of a module. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.	Coupling between modules is the strength of interconnections between modules or a measure of interdependence among modules.
2.	Cohesion is to strengthen the bond between elements of the same module by maximizing the relationship between elements of the same module.	The notion of coupling attempts to capture this concept of "how strongly" different modules are interconnected.
3.	Cohesion of a module represent how tightly bound the internal elements of the module are to one another.	Coupling is a measure of the relative interdependence among modules.
4.	The scale of cohesion is non-linear, i.e., low-end cohesiveness is much worse than middle range (but focus on high end cohesion).	A low coupling is always preferred.
5.	For example, in a module "Managing-Book-lending", functions like "Issue-book", "Return-book", "Query-book" are strongly related.	For example, customer billing system.

If a module has logical cohesion then this module likely to have content coupling with others.

Que 3.17. What do you mean functional independence ? Why functional independence is the key factor for a good software design ? Explain. AKTU 2015-16, Marks 10

Answer

Functional independence :

1. The term functional independence means that a module performs a single task and needs very little interaction with other modules.

2. A module that is highly cohesive and also has low coupling with other modules is said to be functionally independent of the other modules.
3. Functional independence is a key to any good design primarily due to the following advantages it offers :

a. Error isolation :

i. Whenever an error exists in a module, functional independence reduces the chances of the error propagating to the other modules.

ii. The reason behind this is that if a module is functionally independent, its interaction with other modules is low.

iii. Therefore, an error existing in the module is very unlikely to affect the functioning of other modules.

iv. Further, once a failure is detected, error isolation makes it very easy to locate the error.

b. Scope of reuse :

i. Reuse of a module for the development of other applications becomes easier.

ii. A functionally independent module performs some well-defined and precise task and the interfaces of the module with other modules are very few and simple.

iii. A functionally independent module can, therefore, be easily taken out and reused in a different program.

c. Understandabililty :

i. When modules are functionally independent, complexity of the design is greatly reduced.

ii. This is because of the fact that different modules can be understood in isolation, since the modules are independent of each other.

iii. Understandability is a major advantage of a modular design.

Que 3.18. What do you mean by structured design ? What tools are used in structure design ?

Answer

1. Structured design provide mean for dealing with complexity that was overwhelming developers as programs grew larger.

2. The structured design methodology is primarily function-oriented and relies heavily on functional abstraction and functional decomposition.
3. The aim is to design a system so that programs implementing the design would have a nice hierarchical structure, with functionally cohesive modules and as few interconnections between modules as possible.
4. There are four major steps in structured design :
   a. Restate the problem as a data flow diagram.
   b. Identify the input and output data elements.
   c. First-level factoring.
   d. Factoring of input, output, and transforms branches.

Tools used for structured design :

1. **Structured chart :** It is a graphical representation of the program's organization.
2. **Modules identification and specification techniques :** In it the individual modules are identified and how they will achieve the goal is specified.

Que 3.19. Discuss the function oriented design strategies.

Answer

1. Function oriented design is the result of focusing attention to the function of the program. This is based on stepwise refinement.
2. Stepwise refinement is based on the iterative procedural decomposition.
3. Stepwise refinement is a top-down strategy where a program is refined as a hierarchy of increasing levels of detail.
4. We start with a high-level description of what the program does. Then, in each step, we take one part of our high-level description and refine it.
5. Refinement is actually a process of elaboration.
6. The process should proceed from a highly conceptual model (abstractions) to lower level details.
7. The refinement of each module is done until we reach the statement level of our programming language.

Que 3.20. Explain object-oriented design.

Answer

1. Object-oriented design is the result of focusing attention not on the function performed by the program, but instead on the data that are to be manipulated by the program.
2. Thus, it is orthogonal to function-oriented design.
3. Object-oriented design begins with an examination of the real world "things" that are part of the problem to be solved.
4. These things are characteristics, individually in terms of their attributes and behaviour.
5. Each object maintains its own state, and offers a set of services to other objects. Objects communicate by message passing (example parameters).
6. Objects are independent entities that may readily be changed because all state and representation information is held within the object itself.
7. Objects may be distributed and may execute either sequentially or in parallel.
8. For example, a car is an object. It has state : whether its engine is running; and it has behaviour : starting the car, which changes its state from "engine not running" to "engine running".
9. Object-oriented technology contains following three key aspects :
   a. **Objects :** Software packages are designed and developed to correspond with real world entities that contain all the data and services required to function as their associated entities messages.
   b. **Communication :** Communication mechanisms are established that provide the means by which objects work together.
   c. **Methods :** Methods are services that objects perform to satisfy the functional requirements of the problem domain. Objects request services of the other objects through messages.

Que 3.21. Discuss the differences between object oriented design and function oriented design. **AKTU 2012-13, Marks 10**

Answer

S. No.	Function oriented design	Object-oriented design
1.	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
2.	Functions are grouped together by which a higher level function is obtained. An example of this technique is structure analysis/structure design (SA/SD).	Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.
3.	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.
4.	Approach is mainly used for computation sensitive application.	Whereas object-oriented design approach is mainly used for evolving system which mimics a business process or business case.
5.	In function oriented design we decompose in function/ procedure level.	We decompose in class level.
6.	It is a top down approach.	It is a bottom up approach.
7.	It views system as black box that performs high level function and later decomposes its detailed function so to be mapped to modules.	Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.
8.	Begins by considering the use case diagrams and scenarios.	Begins by identifying objects and classes.

Que 3.22. **What are the fundamental issues in software design ?**

Answer

Fundamental issues in software design :

1. **Concurrency :** Design for concurrency is concerned with decomposing software into processes, tasks, and threads and dealing with related issues of efficiency, atomicity, synchronization, and scheduling.
2. **Control and handling of events :** This design issue is concerned with how to organize data and control flow as well as how to handle reactive and temporal events through various mechanisms such as implicit invocation and callbacks.
3. **Data persistence :** This design issue is concerned with how to handle long-lived data.
4. **Distribution of components :** This design issue is concerned with how to distribute the software across the hardware (including computer hardware and network hardware), how the components communicate, and how middleware can be used to deal with heterogeneous software.
5. **Error and exception handling and fault tolerance :** This design issue is concerned with how to prevent, tolerate, and process errors and deal with exceptional conditions.

Que 3.23. **Explain top-down and bottom-up approach in software design.**

Answer

Top-down design approach :

1. A top-down design approach starts by identifying the major components of the system, decomposing them into their lower-level components and iterating until the desired level of detail is achieved.
2. Starting from an abstract design, in each step, the design is refined to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly.

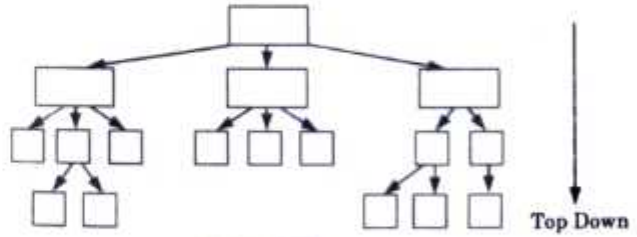


Fig. 3.23.1.

3. A top-down approach is suitable only if the specifications of the system are clearly known and the system development is from scratch.

4. Hence, it is a reasonable approach if a waterfall type of process model is being used.

Bottom-up design approach :

1. A bottom-up design approach starts with designing the most basic or primitive components and proceeds to higher-level components that use these lower-level components.
2. Bottom-up methods work with layers of abstraction. Starting from the very bottom, operations that provide a layer of abstraction are implemented.
3. The operations of this layer are then used to implement more powerful operations and a still higher layer of abstraction, until the stage is reached where the operations supported by the layer are those desired by the system.

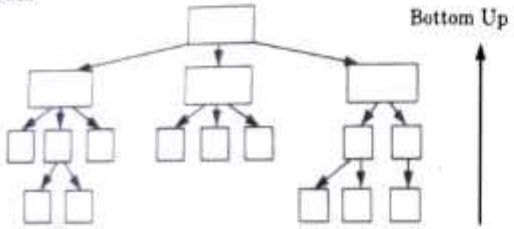


Fig. 3.23.2.

PART-3

Software Measurement and Metrics : Various Size Oriented Measures : Halstead's Software Science, Function Point (FP) Based Measures, Cyclomatic Measures : Control Flow Graphs.

CONCEPT OUTLINE : PART-3

- Software measurement is a new way of managing software development. Measurement enables software managers to assess software quality, improve the software process.
- Software metrics is a continuous application of measurement based techniques to the software development.
- Function points measures functionality from the user point of view, that is, on the basis of what the user requests and receives in return.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.24. Define software metrics. What are various categories of software metrics ?

Answer

Software metrics :

1. Software metrics is a continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products.
2. It covers the techniques for monitoring and controlling the process of the software development.
3. Software metrics are also used to provide management information. This includes information about productivity, quality and process effectiveness.
4. Software metrics are applicable to the whole development life cycle from initiation, when costs must be estimated, to monitoring the reliability of the end product in the field, and the way that product changes over time with enhancement.

Categories of metrics : There are three categories of software metrics :

1. **Product metrics :** It describes the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability etc.
2. **Process metrics :** It describes the effectiveness and quality of the processes that produce the software product. Examples are :
   a. Effort required in the process.
   b. Time to produce the product.
   c. Effectiveness of defect removal during development.
   d. Number of defects found during testing.
   e. Maturity of the process.
3. **Project metrics :** It describes the project characteristics and execution. Examples are :
   a. Number of software developers.
   b. Staffing pattern over the life cycle of the software.
   c. Cost and schedule.
   d. Productivity.

Que 3.25. What is software metric ? How is it different from software measurement ? **AKTU 2014-15, Marks 05**

Answer

Software metric : Refer Q. 3.24, Page 3-21C, Unit-3.

Difference :

S. No.	Metric	Measurement
1.	Metrics are "A method of measuring something, or the results obtained from this".	Measurement is "The action of measuring something or the size, length, or amount of something, as established by measuring".
2.	Metrics are derived combinations of measurements.	Measurements are the raw data.
3.	"Post-release quality" defined as "defects per function point" would be a metric.	Function points released to production is a measure. Severity 1 and 2 defects.

Que 3.26. Define the following terms : objects, message, abstraction, class, inheritance and polymorphism.

AKTU 2014-15, Marks 05

Answer

1. **Object :** An object is an instance of a particular class. In general, any number of objects may be constructed from a class definition. The class to which an object belongs defines the general characteristics of all instances of that class.
2. **Message :** A message is a discrete unit of communication intended by the source for consumption by some recipient or group of recipients.
3. **Abstraction :** It is a simplified representation of something that is potentially quite complex. It is often not necessary to know the exact details of how something works, is represented or is implemented, because we can still make use of it in its simplified form.
4. **Class :** It is a programming language concept that allows data and methods to be grouped together. The class concept is fundamental to the notion of an object-oriented programming language. The methods of a class define the set of permitted operations on the class's data (its attributes).
5. **Inheritance :** It is a feature of object-oriented programming languages in which a sub-type inherits methods and variables from its super-type. Inheritance is most commonly used as a synonym for class inheritance (class inheritance), but interface inheritance is also a feature of some languages, including Java.
6. **Polymorphism :** It is the ability of an object reference to be used as if it referred to an object with different forms. Polymorphism in Java results from both class inheritance and interface inheritance. The apparently different forms often result from the static type of the variable in which the reference is stored.

Que 3.27. What are different technique to estimate size of the program ? Which technique is better and why ?

AKTU 2013-14, Marks 05

Answer

1. The estimation of size of program is very critical and difficult area of the project planning.
2. The difficulties in establishing units for measuring size lie in the fact that the software is essentially abstract.
3. It is difficult to identify the size of a system. Many attempts have been made at establishing a unit of measure for size.
4. Basically we use two methods for estimating the size. They are :
   i. Lines of code (LOC)
   ii. Function point (FP)

Better technique : Function point.

Reason : One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification whereas in LOC, the size can be accurately determined only after the product has fully been developed.

Que 3.28. Describe lines of code (LOC), the size estimation method.

Answer

1. The simplest measure of problem size is lines of code.
2. This metric measures the number of source instructions required to solve a problem.
3. While counting the number of source instructions, line used for commenting the code and the headers lines are ignored.
4. In order to estimate the LOC measures at the beginning of a project, project managers divide the problem into modules, and each module into submodules, and so on until the sizes of the different leaf-level modules can be approximately predicted.
5. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.
6. For example, in the function shown in Fig. 3.28.1, if LOC is simply a count of the number of lines then Fig. 3.28.1 given below contains 10 LOC.

```
1.  main ( )
2.  {
3.  float c, f ;
4.  printf ("Enter temperature in degree centigrade : \n") ;
5.  scanf ("%f", &c) ;
6.  f = (c * 9/5) + 32 ;
7.  printf ("%f is temperature in degree Farenheit :" f) ;
8.  getch ( ) ;
9.  return 0 ;
10. }
```

Fig. 3.28.1.

Que 3.29. Explain the Halstead's software science.

AKTU 2012-13, Marks 10

OR

Give Halstead's software science measures for :

i. **Program length**
ii. **Program volume**
iii. **Program level**
iv. **Efforts**
v. **Language level**

Also, give its advantages and disadvantages.

Answer

Halstead's software science measure :

Halstead's software science is an analytical technique to measure size, development effort and development cost of software products. Halstead uses a few primitive programs parameters to develop the expression for the overall program length, potential minimum volume, actual volume, language level, effort and development time.

1. **Program length** : The length of the program is defined as the total usage of all operators and operands in the program so length,

$$N = N_1 + N_2$$

where

N_1 = total usage of all of the operators appearing in that implementation

N_2 = total usage of all of the operands appearing in that implementation.

2. **Program volume** : This is for the size of any implementation of any algorithm. The number of bits required to encode the program measured is known as volume and given by,

$$V = N \log_2 n$$

where n represents different identifiers uniquely

and $n = n_1 + n_2$

then $V = N \log_2 (n_1 + n_2)$

3. **Program level :** It measures the level of abstraction provided by the programming language.

The program level is given by,

$$L = \frac{V^*}{L}$$

Here, V^* = Potential minimum volume

4. **Efforts or programming effort :** The effort required to implement the program and can be obtained as,

$$\text{Efforts } (E) = \frac{V}{L} = \frac{\text{Program volume}}{\text{Program level}}$$

But $$L = \frac{V^*}{V}$$

Then $$E = \frac{V.V}{V^*} = \frac{V^2}{V^*}$$

5. **Language level :** Language level implies that higher the level of a language, the less effort it takes to develop a program using that language, *i.e.*, language level is inversely proportional to the effort to develop a program.

$$L \alpha \frac{1}{E}$$

6. **Potential volume :** It is a metric for denoting the corresponding parameters in an algorithm's shortest possible form. Neither operators nor operands can require repetition.

$$V^* = (n_1^* + n_2^*) \log_2 (n_1^* + n_2^*)$$

Advantages of Halstead metrics :

1. Do not require in-depth analysis of programming structure.
2. Predicts rate of error.
3. Predicts maintenance effort.
4. Useful in scheduling and reporting projects.
5. Measure overall quality of programs.

Disadvantages of Halstead metrics :

1. It depends on the completed code.
2. It has little or no use of predictive estimating model. But McCabe's model is more signed to application at the design level.

Que 3.30. **What do you understand by token count ? Consider a program having :**

i. **Number of distinct operator : 12**
ii. **Number of operands : 5**
iii. **Total number of operator occurrences : 20**
iv. **Total number of operand occurrences : 15**

Calculate the different Halstead software metrics for above programs. **AKTU 2013-14, Marks 10**

Answer

Token count : Token count is defined as size of program *i.e.*, number of tokens, where a token is a lexical token (keyword, arithmetic operator, constants, symbol), as recognized by compiler.

Numerical :

$$\text{Program length} = N = N_1 + N_2 = 20 + 15$$
$$N = 35$$
$$\text{Program vocabulary} = \eta = \eta_1 + \eta_2$$
$$\eta = 12 + 05$$
$$\eta = 17$$
$$\text{Volume of program} = V = N^* \log_2 \eta$$
$$V = 25^* \log 17 = 25^* 4.08 = 102$$
$$\text{Potential volume of program} = V^* = (2 + \eta_2^*) \log_2 (2 + \eta_2^*)$$
$$= (2 + 12) \log(2 + 12)$$
$$= 14 \log 14 = 14 \times 3.80 = 53.2$$
$$\text{Program level} = L = V^*/V = 53.2/102 = 0.54$$

Que 3.31. **For the following 'C' program estimate the Halstead's length and volume measures. Compare Halstead length and volume measures of size with LOC measure.**

```
// Program to calculate GCD of two number
int compute-gcd(x, y)
{
  int x, y;
  while (x != y)
        if (x > y) then x = x - y;
        else y = y - x;
  return x;
}
```

AKTU 2014-15, Marks 10

Answer

Table 3.31.1.

Operator	Occurrence	Operands	Occurrence
Compute-gcd	1	x	8
=	3	y	7
!	1		
while	1		
>	1		
–	2		
return	1		
if	1		
[	1		
]	1		
( )	3		
;	4		
int	2		
else	1		
then	1		
$n_1 = 15$	$N_1 = 24$	$n_2 = 2$	$N_2 = 15$

The length, $N = N_1 + N_2 = 24 + 15 = 39$

The volume, $V = N \log_2 (n_1 + n_2) = 39 \log_2 17$
$= 159.41$

Que 3.32. **Describe the function point based measures and metrics.**

Answer

1. Function points is used in the estimation of software development cost which is the most important potential use of function point data.
2. Function point may be used to compute the following important metrics :
   Productivity = FP / persons-months
   Quality = Defects / FP
   Cost = Rupees / FP
   Documentation = Pages of documentation per FP
3. The five functional units as shown in Table 3.32.1 are ranked according to their complexity, *i.e.*, low, average, or high, using a set of prescriptive standards.
4. Organizations that use function point method develop criteria for determining whether a particular entry is low, average or high.
5. After classifying each of the five function types, the unadjusted function points (UFP) are calculated using predefined weights for each function type as given in Table 3.32.2.

Table 3.32.1.

Functional units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Outputs (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal Logical Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

Table 3.32.2.

Functional Units	Count	Complexity	Complexity Total	Functional Unit Totals
External Inputs (EIs)		Low × 3 = Average × 4 = High × 6 =		
External Outputs (EOs)		Low × 4 = Average × 5 = High × 7 =		
External Inquiries (EQs)		Low × 3 = Average × 4 = High × 6 =		
Internal Logical Files (ILFs)		Low × 7 = Average × 10 = High × 15 =		
External Interface Files (EIFs)		Low × 5 = Average × 7 = High × 10 =		
Total unadjusted function point count =				

6. The weighting factors are identified as per Table 3.32.1 for all functional units and multiplied with the functional units accordingly.
7. The procedure for the calculation of UFP in mathematical form is given below

$$UFP = \sum_{i=1}^{3} \sum_{j=1}^{3} Z_{ij} * W_{ij}$$

where i indicates the row and j indicates the column of Table 3.32.1.
W_{ij} : It is the entry of the i^{th} row and j^{th} column of the Table 3.32.1.

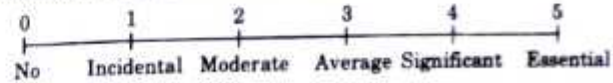
Z_{ij} : It is the count of the number of functional units of type i that have been classified as having the complexity corresponding to column j.

8. The final number of function points is arrived at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity.
9. In this, adjustment factor allows the UFP count to be modified by at most 35%. The final adjusted FP count is obtained by using the following relationship,

$$FP = UFP * CAF$$

where CAF is complexity adjustment factor and is equal to $(0.65 + 0.01 \times \Sigma F_i)$.

10. The F_i (i = 1 to 14) are the degrees of influence and can be calculated according to following sequence :
Rate each factor on a scale of 0 to 5.



1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing heavily utilized operational environment ?
6. Does the system require online data entry ?
7. Does the online data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated online ?
9. Is the inputs, outputs, files or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by user ?

Que 3.33. **Consider a project with the following functional units :**

Number of user inputs = 50
Number of user outputs = 40
Number of user enquiries = 35
Number of user files = 6
Number of external interfaces = 4

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Answer

$$UFP = \sum_{i=1}^{5}\sum_{j=1}^{3} Z_{ij} * W_{ij}$$

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7$$

$$= 200 + 200 + 140 + 60 + 28 = 628$$

$$CAF = (0.65 + 0.01 \Sigma Fi)$$

$$= (0.65 + 0.01 (14 \times 3)) = 0.65 + 0.42 = 1.07$$

$$FP = UFP \times CAF$$

$$= 628 \times 1.07 = 672$$

Que 3.34. Consider a project with the following parameters :

i. **External inputs :**
   a. **10 with low complexity**
   b. **15 with average complexity**
   c. **17 with high complexity**
ii. **External outputs :**
   a. **6 with low complexity**
   b. **13 with high complexity**
iii. **External inquiries :**
   a. **3 with low complexity**
   b. **4 with average complexity**
   c. **2 with high complexity**
iv. **Internal logical files :**
   a. **2 with average complexity**
   b. **1 with high complexity**
v. **External interface files :**
   a. **9 with low complexity**

In addition to above, system requires

i. **Significant data communication**
ii. **Performance is very critical**
iii. **Designed code may be moderately reusable**
iv. **System is not designed for multiple installations in different organizations.**

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Answer

To calculate the function point firstly, we have to calculate the unadjusted functional point counts as follows :

Functional Units	Count	Complexity		Complexity Total	Functional Unit Totals
External Inputs (ELs)	10	Low × 3	=	30	
	15	Average × 4	=	60	
	17	High × 6	=	102	192
External Outputs (EOs)	6	Low × 4	=	24	
	0	Average × 5	=	0	
	13	High × 7	=	91	115
External Inquiries (EQs)	3	Low × 3	=	9	
	4	Average × 4	=	16	
	2	High × 6	=	12	37
Internal Logical Files (ILFs)	0	Low × 7	=	0	
	2	Average × 10	=	20	
	1	High × 15	=	15	35
External Interface Files (EIFs)	9	Low × 5	=	45	
	0	Average × 7	=	0	
	0	High × 10	=	0	45

Total unadjusted function point count = 424

The factors given previously can be calculated as :

$$\sum_{i=1}^{14} F_i = 3 + 4 + 3 + 5 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 2 + 3 + 0 + 3 = 41$$

$$CAF = (0.65 + 0.01 * \Sigma Fi)$$

$$= (0.65 + 0.01 * 41)$$

$$= 1.06$$

$$FP = UFP * CAF$$

$$= 424 * 1.06$$

$$= 449.44$$

Que 3.35. What do you mean by cyclomatic complexity ? Enlist the properties of cyclomatic complexity.

OR

Write short note on cyclomatic complexity measures.

AKTU 2013-14, Marks 05

Answer

1. The cyclomatic complexity is also known as structural complexity because it gives internal view of the code.

2. This approach is used to find the number of independent paths through a program.
3. This provides us the upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once and every condition has been executed on its true and false side.
4. The complexity measure is defined in terms of independent paths that when taken in combination will generate every possible path.
5. An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
6. McCabe's Cyclomatic metric, $V(G)$ of a graph G with n vertices, e edges, and P connected components is $V(G) = e - n + 2P$.
7. Given a program we will associate with it a directed graph that has unique entry and exit nodes. Each node in the graph corresponds to a block of code in the program where the flow is sequential and the arcs correspond to branches taken in the program.
8. This graph is classically known as flow graph and it is assumed that each node can be reached by the entry node and each node can reach the exit node.
9. For example, a flow graph shown in Fig. 3.35.1 below with entry node 'a' and exit node 'f'.

The value of cyclomatic complexity can be calculated as

$$V(G) = 9 - 6 + 2 = 5$$

Here $e = 9$, $n = 6$ and $P = 1$

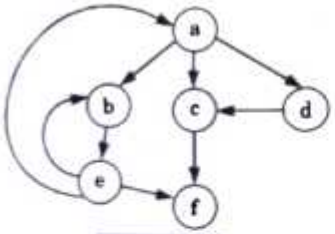


Fig. 3.35.1.

There will be following five independent paths for the above flow graph :

path 1 : a c f
path 2 : a b e f
path 3 : a d c f
path 4 : a b e a c f or a b e a b e f
path 5 : a b e b e f

Properties of cyclomatic complexity :

1. $V(G) >= 1$
2. $V(G)$ is the maximum number of independent paths in graph G.
3. Inserting and deleting functional statements to G does not affect $V(G)$.

4. G has only one path if and only if $V(G) = 1$.
5. Inserting a new row in G increases $V(G)$ by unity.
6. $V(G)$ depends only on the decision structure of G.

Que 3.36. Consider a flow graph given below, calculate its cyclomatic complexity.

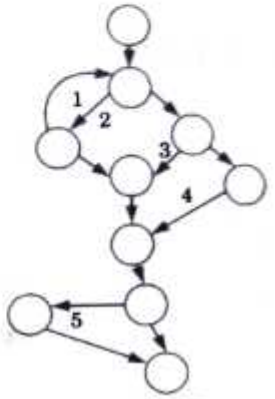


Fig. 3.36.1.

OR

What do you mean by cyclomatic complexity of a software ? How it is computed ? Discuss with example. AKTU 2012-13, Marks 10

Answer

Cyclomatic complexity : Refer Q. 3.35, Page 3–31C, Unit-1.

Example :

Cyclomatic complexity can be calculated by any of the three methods.

i. $$V(G) = e - n + 2P$$
$$= 13 - 10 + 2 = 5$$

ii. $$V(G) = n + 1$$
$$= 4 + 1 = 5$$

iii. $$V(G) = \text{number of regions}$$
$$= 5$$

Therefore, complexity value of a flow graph is 5.

Que 3.37. Draw a flow graph, arrive at the cyclomatic complexity and find the set of linearly independent paths for the following program :

```
void F(int key, int T[ ], int size, boolean found, int L)
    {
    int bot, top, mid;
    bot = 0;
    top = size-1;
    L = (top+bot)/2;
    if(T[L] == key) found = true;
    else
    found = false;
    while(bot <== top && !found)
    {
    mid = (top+bot)/2;
    if(T[mid] == key)
    {
    found = true; L = mid;
    }
    else if(T[mid] < key)
    bot = mid + 1 ;
    }
    }
```

AKTU 2015-16, Marks 10

Answer

Number of edges (e) = 20

Number of nodes (n) = 17

i. $V(G) = e - n + 2P = 20 - 17 + 2(1) = 5$

where P is number of connected components.

ii. $V(G) = p + 1, 4 + 1 = 5$

where p is number of predicate (decision) nodes.

iii. V(G) = number of regions = 5

Therefore, complexity value of flow graph is 5.

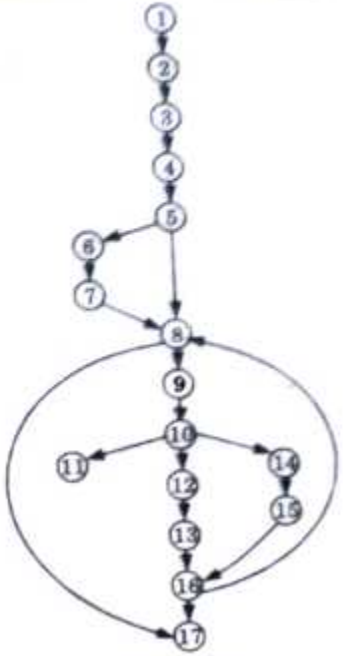


Fig. 3.27.1.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Explain the term software design along with design principles.

Ans. Refer Q. 3.1.

Q. 2. What do you mean by coupling ?

Ans. Refer Q. 3.10.

Q. 3. Discuss the concept of cohesion.

Ans. Refer Q. 3.11.

Q. 4. Differentiate between object oriented design and function oriented design.

Ans. Refer Q. 3.21.

Q. 5. Explain software metrics with its types.

Ans. Refer Q. 3.24.

Q. 6. Discuss Halstead's software science.

Ans. Refer Q. 3.29.

Q. 7. Write a short note on function point based measures and metrics.

Ans. Refer Q. 3.32.

Q. 8. How cyclomatic complexity is measured ?

Ans. Refer Q. 3.35.

4 UNIT

Software Testing

PART-1

Testing Objectives, Unit Testing, Integration Testing, Acceptance Testing, Regression Testing, Testing for Functionality and Testing for Performance.

CONCEPT OUTLINE : PART-1

- Software testing is the process of executing a program to locate an error.
- Software testing objectives are :
  - i. Software quality improvement
  - ii. Verification and validation
  - iii. Software reliability estimation
- There are certain levels of testing which are as follows :
  - i. Unit testing
  - ii. Integration testing
  - iii. System testing
  - iv. Acceptance testing

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. What is software testing ? Discuss the objectives of it.

OR

Write short note on software testing. AKTU 2013-14, Marks 05

Answer

1. Software testing is the process of executing a program with the intention of finding errors in the code.
2. It is a process to verify that whether a system or its parts is satisfying specified requirements or not.
3. Testing is the fundamental of software success.
4. Testing is not a distinct phase in system development life cycle but is applicable throughout all phases *i.e.*, design development and maintenance phase.

Following are objectives of software testing :

1. **Software quality improvement :**
   a. Software quality means the conformance to the specified software design requirements.

   b. The minimum requirement of quality means performing as required under specified circumstances.
   c. Software testing is not only used to remove bugs, but also to find out design defect by the programmer.
2. **Verification and validation :**
   a. Verification means to test that we are building the product in right way *i.e.*, we are using the correct procedure for the development of software so that it can meet the user requirements.
   b. Whereas validation is the process which checks that whether we are building the right product or not.
3. **Software reliability estimation :**
   a. Software reliability has important relationship with many aspects of software development.
   b. Its objective is to discover the residual designing errors before delivery to the customer.
   c. The failure data during the testing process are taken down in order to estimate the software reliability.

Que 4.2. What are the principles of software testing ?

Answer

Principles of software testing :

1. All tests should be traceable to customer requirements.
2. Testing time and resources are limited, so avoid redundant test.
3. It is impossible to test everything.
4. Use effective resources to test.
5. Test should be planned long before testing begins *i.e.*, after requirement phase.

Que 4.3. Does fault necessarily lead to failures ? Justify your answer with an example. AKTU 2013-14, Marks 05

Answer

1. No, it is not necessary that fault lead to failure.
2. A system is fault-tolerant, if faults in its hardware or software components will not necessarily lead to system failure.
3. This property is usually obtained by using redundancy in time (for example, repeated computation), space (for example, replicated functional units), or additional functions (for example, failure detectors).
4. Fault-tolerant distributed systems are a classic application area for a wide variety of fault tolerance techniques.

5. Distributed systems can even tolerate fires, floods, and other disasters, as redundant components may reside in different buildings or cities.
6. On the other hand, fault tolerance techniques in distributed systems are hard to implement, because the subsystems are only loosely connected by unreliable links which may subject to failure by themselves. This proves fault does not necessarily lead to system failure.

Que 4.4. **What do you understand by unit / module testing ? Explain.**

Answer

1. Unit testing is a level of software testing where individual units/ components of software are tested.
2. Unit testing focuses verification effort on the smallest unit of software design, the software component or module.
3. In unit testing, individual components are tested to ensure that they are working properly in the same manner as required.
4. In this process, a module (software component) is taken and executed in isolation from the rest of software product. That's why it is also called "software component testing".
5. It is the lowest level of testing of an application.
6. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing.
7. The unit testing is white box oriented, and the step can be conducted in parallel for multiple components.
8. Unit testing is typically conducted by development team and programmer who coded the unit.

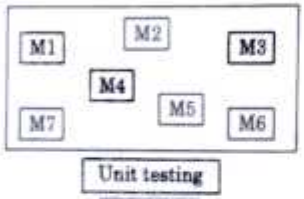


Fig. 4.4.1.

Que 4.5. **What is integration testing ? What are various software integration techniques ?**

Answer

Integration testing :

1. **Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.**

2. It occurs after unit testing and before validation testing.
3. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

The various software integration techniques are as follows :

1. **Top-down integration testing :**
   i. In this approach, testing process starts with testing the top most module/component in the hierarchy and move downwards.
   ii. This process continues until all components are integrated and then whole system has been completely tested.
   iii. Top-down integration testing approach requires the use of program stubs to simulate the effect of lower-level routines that are called by the routines under test.
   iv. A pure top-down integration does not require any driver routines.

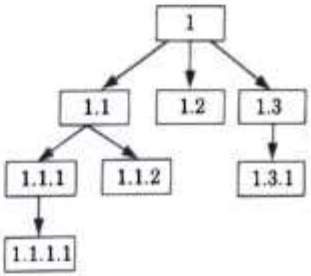


Fig. 4.5.1. Top-down integration.

2. **Bottom-up integration testing :**
   i. In this method, each subsystem (component) at the lower level of system hierarchy is tested individually first.
   ii. Then the next component who calls the previously tested ones to be tested.
   iii. This approach is followed repeatedly until all components are included in the testing.
   iv. The primary purpose to test each subsystem is to test interface among various models making up the subsystem.
   v. In this, both control and data interface are tested.
   vi. In pure bottom-up testing, no stubs are required, and only test drivers are required.
   vii. Bottom-up integration testing is mainly used when the performance and machine interface of the system is the main concern.

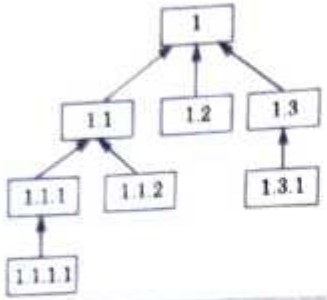


Fig. 4.5.2. Bottom-up integration testing.

3. **Big-bang testing :**
   i. It is a non-incremental integration approach.
   ii. It is the simplest testing approach, where all modules making a complete system are integrated and tested as a whole and disorder usually gives unpredictable results.
   iii. The set of errors encountered here are very difficult to localize as it may potentially belong to any of the module being integrated.
   iv. And once these errors are corrected, new ones appear and the process continues in an endless loop.

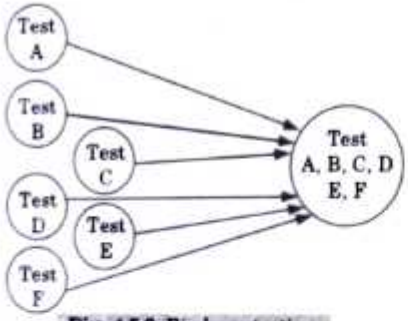


Fig. 4.5.3. Big-bang testing.

4. **Sandwich integration testing :**
   i. The combination of top-down and bottom-up integration testing techniques is called sandwich integration.
   ii. This system is viewed as three layers just like a sandwich.
   iii. An upper layer of sandwich use top-down integration, the lower layer of sandwich integration use bottom-up integration, and the middle layer called target layer use the testing on the basis of system characteristics and structure of the component hierarchy.

Que 4.6. What is acceptance testing ? Why acceptance testing is needed ?

Answer

1. Acceptance testing is a level of software testing where a system is tested for acceptability.
2. Acceptance testing, a testing technique performed to determine whether or not the software system has met the requirement specifications.
3. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Need of acceptance testing : Acceptance testing is needed due to following reasons :

1. Developers code software based on requirements document which is their "own" understanding of the requirements and may not actually be what the client needs from the software.
2. Requirement changes during the course of the project may not be communicated effectively to the developers.

Que 4.7. Define stress testing. Why is stress testing applicable to only certain types of systems ?

Answer

1. Stress testing is designed to determine the behaviour of the software under abnormal situations.
2. In this testing, test cases are designed to execute the system in such a way that abnormal conditions arise.
3. Some examples of test cases that may be designed for stress testing are :
   i. Test cases that generate interrupts at a much higher rate than the average rate.
   ii. Test cases that demand excessive use of memory as well as other resources.
   iii. Test cases that cause 'thrashing' by causing excessive disc accessing.
4. IEEE defines stress testing as 'testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.' Stress testing is applicable to certain type of systems which generally work below their maximum capability, however, are stressed for some peak hours only.

Que 4.8. What is security testing ?

Answer

1. It is a type of non-functional testing.
2. Security testing is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.
3. Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended.

There are four main focus areas to be considered in security testing (especially for web sites/applications) :

1. **Network security** : This involves looking for vulnerabilities in the network infrastructure (resources and policies).
2. **System software security** : This involves assessing weaknesses in the various software (operating system, database system, and other software) the application depends on.
3. **Client-side application security** : This deals with ensuring that the client (browser or any such tool) cannot be manipulated.
4. **Server-side application security** : This involves making sure that the server code and its technologies are robust enough to fend off any intrusion.

Que 4.9. What is meant by system testing ?

Answer

a. At this level the entire software system is tested.

b. The reference document for this purpose is the requirements document, and the objective is to see if the software meets its requirements.

c. This technique involves testing of a product by verifying its working with the original requirements specification document.

d. Following types of testing are used at this stage :

1. **Functional testing :**
   a. It is concerned with functionality rather than implementation of the program.
   b. In functional testing the structure of the program is not considered.
   c. Test cases are decided solely on the basis of the requirements or specifications of the program or module, and the intervals of the module of the program are not considered for selection of test cases.
   d. Due to its nature, functional testing is often called "black box testing".

2. **Performance testing :**
   a. It deals with quality related issues like security test, reliability test etc.
   b. System testing is done at the developers and before the product is given to customer for use.
   c. System tests are designed to validate a fully developed system with a view to assuring that it meets its requirements.

Que 4.10. What are the objectives of software testing ? Discuss the purpose of integration testing. How is integration testing done ?

AKTU 2012-13, Marks 10

Answer

Objectives of software testing : Refer Q. 4.1, Page 4–2C, Unit-4.

Purpose : Main purpose of integration testing is to identify the functional, requirement and performance level bugs.

How is integration testing done : Refer Q. 4.5, Page 4–4C, Unit-4.

Que 4.11. What is regression testing ? When we need regression testing ?

Answer

1. Regression testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
2. Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
3. This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that old code still works once the new code changes are done.

Need of regression testing :

Regression testing is required when there is a :

1. Change in requirements and code is modified according to the requirement.
2. New feature is added to the software.
3. Defect fixing.
4. Performance issue fixing.

Que 4.12. Define software testing. Explain various level of testing.

Answer

Software testing : Refer Q. 4.1, Page 4–2C, Unit-4.

Various levels of testing : Fig. 4.12.1 shows various levels of testing :

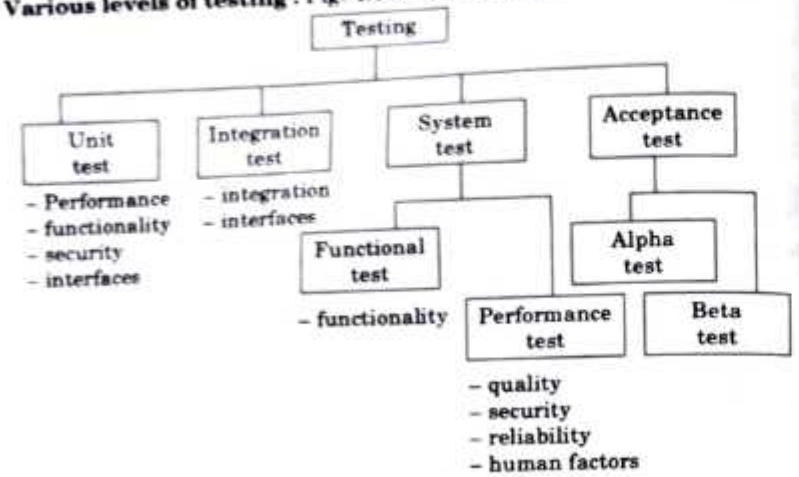


Fig. 4.12.1. Levels of testing.

1. **Unit testing :** Refer Q. 4.4, Page 4–4C, Unit-4.
2. **Integration testing :** Refer Q. 4.5, Page 4–4C, Unit-4.
3. **System testing :** Refer Q. 4.9, Page 4–8C, Unit-4.
4. **Acceptance testing :** Refer Q. 4.6, Page 4–7C, Unit-4.

Que 4.13. What are the attributes of a good software test ? Why is regression testing an important part of any integration testing procedure ? **AKTU 2013-14, Marks 05**

Answer

Following are the attributes of a good software test :

a. Has a high probability of finding an error.
b. Non-redundant.
c. Should be capable of uncovering a whole class of errors.
d. Should not be too simple or too complex.

Regression testing is an important part of any integration testing procedure as because :

1. It allows for re-execution of the systems after changes have been made to ensure that the system still has integrity.
2. Side effects associated with these changes may cause problems with functions that were previously working.
3. Regression testing allows for tests that have already been conducted to ensure that changes have not propagated unintended side effects.

Que 4.14. What is software testing ? What are the attributes of "good" test ? Distinguish among error, fault and failure.

Answer

Software testing : Refer Q. 4.1, Page 4–2C, Unit-4.

Attributes of good test : Refer Q. 4.13, Page 4–10C, Unit-4.

Difference between error, fault and failure :

S. No.	Error	Fault	Failure
1.	Error is the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.	It is incorrect step, process or data definition in a computer program which causes the program to behave in an unintended or unanticipated manner. It is the result of the error.	Failure is the inability of a system or component to perform its required function within the specified performance requirement.
2.	Error refers to a missing or wrong person action resulting in certain fault being injected into software.	Fault refers to an underlying condition within software that causes failure to happen.	A software failure occurs if the behaviour of the software is different from the specified behaviour. Failures may be caused due to functional or performance reasons.
3.	According to the IEEE, "Error is the human mistake that caused fault."	According to IEEE, "Fault is the discrepancy in code that causes a failure."	According to IEEE, "Failure occurs when the external behaviour is incorrect."
4.	Errors also include error sources such as human mis-understandings, dissensions, misinterpretation and so on.	Software fault is a hidden programming error.	A failure is produced only when there is a fault in the system. However, presence of a fault does not guarantee a failure.

PART-2

Top-Down and Bottom-Up Testing Strategies : Test Drivers and Test Stubs, Structural Testing (White Box Testing), Functional Testing (Black Box Testing), Test Data Suite Preparation, Alpha and Beta Testing of Products.

CONCEPT OUTLINE : PART-2

- Test strategies are :
  i. Test drivers
  ii. Test stubs
- White box testing is also known as structural testing. It has the knowledge of the internal structure of the code which can be used to find number of test cases require to generate a given level of test coverage.
- Black box testing is a testing which is based on the functionality of the program.
- In alpha testing, to solve the problem, customer is called at the developer site to test the system.
- Beta testing of software is done at the user site in real world environment.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.15. **What are drivers and stub modules in context of integration and unit testing of software product ? Why are stubs and drivers modules required ?** **AKTU 2015-16, Marks 10**

Answer

Drivers and stub modules in context of integration testing :

1. **Drivers :**
   a. Test drivers are used during bottom-up integration testing in order to simulate the behaviour of the upper level modules that are not yet integrated.
   b. Test drivers are the modules that act as temporary replacement for a calling module and give the same output as that of the actual product.

   c. Drivers are also used when the software needs to interact with an external system and are usually complex than stubs.

Driver-flow diagram :

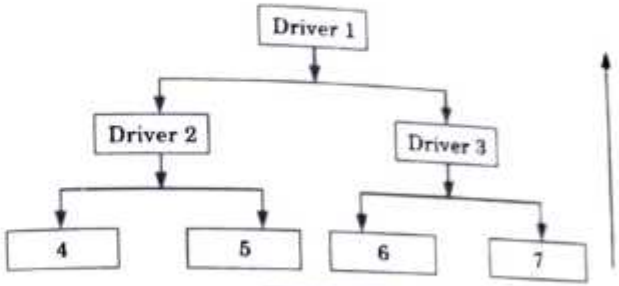


Fig. 4.15.1.

The above diagrams clearly states that modules 4, 5, 6 and 7 are available for integration, whereas, above modules are still under development that cannot be integrated at this point of time. Hence, drivers are used to test the modules.

2. **Stubs :**
   a. Stubs are used during top-down integration testing, in order to simulate the behaviour of the lower-level modules that are not yet integrated. Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.
   b. Stubs are also used when the software needs to interact with an external system.

Stub-flow diagram :

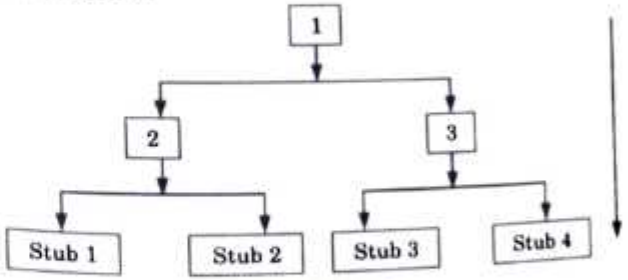


Fig. 4.15.2.

The above diagram clearly states that modules 1, 2 and 3 are available for integration, whereas, below modules are still under development that cannot be integrated at this point of time. Hence, stubs are used to test the modules.

Drivers and stubs in context of unit testing :

1. In order to test a single module, we need a complete environment to provide all relevant code that is necessary for execution of the module. Besides the module under test, the following are needed to test the module :
   a. The procedures belonging to other modules that the module under test calls.
   b. Non-local data structures that the module accesses.
   c. A procedure to call the functions of the module under test with appropriate parameters.
2. Modules required to provide the necessary environment are usually not available until they too have been unit tested. In this context, stubs and drivers are designed to provide the complete environment for a module so that testing can be carried out.
3. The role of stub and driver modules is pictorially shown in Fig. 4.15.3.

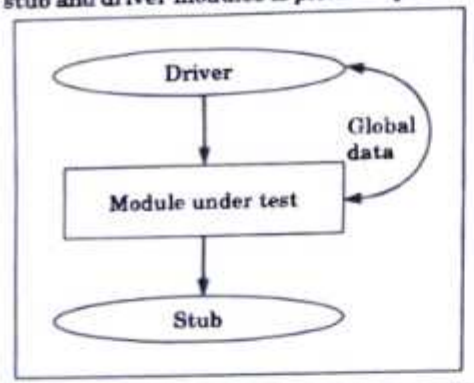


Fig. 4.15.3. Unit testing with the help of driver and stub modules.

Stub :

A stub procedure is a dummy procedure that has the same Input / Output parameters as the given procedure, but has a highly simplified behaviour. For example, a stub procedure may produce the expected behaviour using a simple table look up mechanism.

Driver :

A driver module should contain the non-local data structures accessed by the module under test. Additionally, it should also have the code to call the different functions of the module under test with appropriate parameter values for testing.

Que 4.16. What is software testing ? Describe white box testing in detail.

Answer

Software testing : Refer Q. 4.1, Page 4–2C, Unit-4.

White box testing :

1. White box testing is a software testing approach that examines the program structure and derives test data from the program logic.
2. In this approach, test group must have complete knowledge about the internal structure of the software.
3. Structural testing is usually applied to relatively small program units such as subroutine or operations associated with objects.
4. The white box testing allows to peep (preview) inside the box, it basically focuses on internal knowledge of the software to guide the selection of test data.
5. The white box testing is also known by name of glass box testing, structural testing, clear box testing, open-box testing, logic driven testing and path-oriented testing.
6. Test cases can be derived that :
   a. Guarantee that all independent paths within a module have been exercise at least once.
   b. Exercise all logical decisions on their true and false sides.
   c. Execute all loops at their boundaries and within their operational bounds.
   d. Exercise internal data structures to assure their validity.
7. In white box testing, the test cases are selected on the basis of examination of the code rather than specification. The white box testing is illustrated in Fig. 4.16.1.

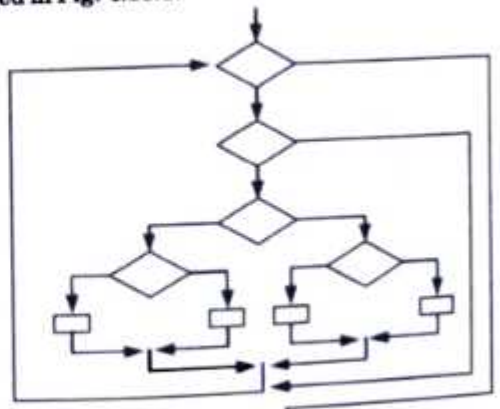


Fig. 4.16.1.

Advantages of white box testing :

1. Helps to optimize the code.
2. Reveals errors in hidden code.

Disadvantages of white box testing :

1. It is very expensive.
2. It is time consuming.

Que 4.17. Explain white box testing techniques.

Answer

1. In the white box testing the code coverage analysis is main part.
2. Code coverage analysis helps to identify the gaps in a test case suite. It allows us to find the area in the code which is not executed by given set of test cases. Upon identifying the gaps in the test case suite we can add the respective test case. So it helps to improve the quality of the software application.

Following are white box testing techniques :

1. **Statement coverage :**
   a. This white box testing technique tries to cover 100% statement coverage of the code, it means while testing every possible statement in the code is executed at least once.
   b. Tools : To test the statement coverage the Cantata++ can be used as white box testing tool.
2. **Decision coverage :**
   a. This white box testing technique try to cover 100% decision coverage of the code, it means while testing every possible decision conditions like if-else, for loop and other conditional loops in the code is executed at least once.
   b. Tools : To cover the decision coverage testing in the code the TCAT-PATH is used.
3. **Condition coverage :**
   a. This white box testing tries to cover 100% condition coverage of the code, it means while testing every possible conditions in the code is executed at least once.
4. **Decision/condition coverage :**
   a. This mixed type of white box testing technique try to cover 100% decision/condition coverage of the code, it means while testing the every possible decisions/conditions in the code is executed at least once.
5. **Multiple condition coverage :**
   a. In this testing, we ensure that each entry point of the system is executed once.

   b. In the actual development process developers make use of the combination of techniques those are suitable for their software application.

Que 4.18. What is black box testing ? List out advantages and disadvantages of black box testing.

Answer

1. Black box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or working.
2. In black box testing, the tester do not know the internal structure of module, he tests only for input/output behaviour.
3. It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
4. The selection of test cases is based on the specification of the test object without knowledge of its inner structure.
5. It is also important to design test cases that demonstrate the behaviour of the test object on erroneous input data.
6. The black box testing is also known by the name of functional testing, exterior testing, specification testing, data-driven testing and input/output driven testing.

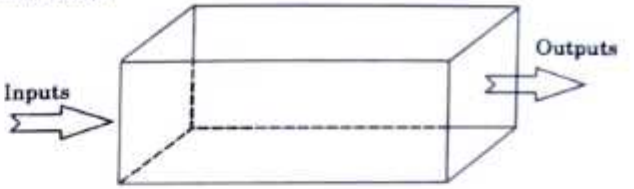


Fig. 4.18.1. Black box testing.

7. Test case should be derived which :
   a. Reduce the number of additional test cases that must be designed to achieve reasonable testing.
   b. Tell us something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

Advantages of black box testing :

1. The test is unbiased.
2. Test is done from the point of view of user not the designer.
3. Test cases can be designed as soon as specification is completed.

Disadvantages of black box testing :

1. The test can be redundant if software designer has already run a test case.

2. The test cases are difficult to design.

Que 4.19. **Design a black box test suite for a program that computes the intersection point of two straight lines and displays the result as "Parallel lines"/ "Intersecting lines"/ "Coincident lines". It reads two integer pairs ($m1$, $c1$) and ($m2$, $c2$) defining the two straight lines of the form $y = mx + c$. The lines are parallel if $m1 = m2$, $c1 \neq c2$; intersecting if $m2 \neq m2$; and coincident if $m1 = m2$, $c1 = c2$.**

Answer

The equivalence classes are the following :

i. Parallel lines ($m1 = m2, c1 \neq c2$)

ii. Intersecting lines ($m1 \neq m2$)

iii. Coincident lines ($m1 = m2, c1 = c2$)

Now, selecting one representative value from each equivalence class, we get the required equivalence class test suite.

If conditions are true then :

Case 1 :

$m1 = 5, c1 = 6$ and $m2 = 5, c2 = 50$;

the lines are parallel.

Case 2 :

$m1 = 5, c1 = 7$ and $m2 = 5, c2 = 7$;

the lines are coincident.

Case 3 :

$m1 = 5, c1 = 6$ and $m2 = 8, c2 = 6$;

the lines are intersecting.

Que 4.20. **Explain the different techniques used in black box testing.**

Answer

Following are the techniques used for black box testing :

i. Equivalence class partitioning :

1. Equivalence class partitioning is a method which divides the input data of a software unit into partitions of equivalence classes from which test cases can be derived.
2. Equivalence classes are usually formed for input domains and are classified into a valid equivalence class and one or more invalid equivalence classes.
3. The idea is to partition the input domain of the system into a finite number of equivalence classes such that each member of the class would behave in a similar fashion.

4. The techniques increase the efficiency of software testing as the numbers of input states are drastically reduced.
5. This techniques involves two steps :
   a. Identification of equivalence classes.
   b. Generating the test cases.

ii. Boundary value analysis :

1. Boundary value analysis is a technique in which test are designed to include representatives of boundary values in a range.
2. In boundary value analysis, we choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence classes.
3. Boundary values for each equivalence class, including the equivalence classes of the output, should be covered.
4. Boundary value test cases are also called "extreme cases".
5. Hence, we can say that a boundary value test case is a set of input data that lies on the edge of boundary of a class of input data or that generates output that lies at the boundary of a class of output data.
6. Guidelines for boundary value analysis are :
   a. If an input condition specifies a range bounded by values a and b, test cases should be designed with the values a and b and just above and just below a and b.
   b. If an input condition specifies a number of input value, test cases should be developed that exercise the minimum and maximum number. Values just above and just below minimum and maximum are also tested.

By applying above guidelines, the boundary testing will be more complete thereby having a higher likelihood for error detection.

iii. Cause-effect graphing :

1. Cause-effect graphing is a technique that aids in selecting combinations of input conditions in a systematic way, such that the number of test cases does not become unmanageably large.
2. This technique starts with identifying causes and effects of the system under testing.
3. A cause is a distinct input condition, and an effect is a distinct output condition.
4. Each condition forms a node in the cause-effect graph.
5. The condition should be states such that they can be set to either true or false.

In steps cause effect testing are as follows :

1. For a model, identify input conditions (causes) and actions (effect).
2. Develop a cause-effect graph.
3. Transform cause-effect graph into a decision table.
4. Convert decision table rules to test cases. Each column of decision table represents a test case.

Que 4.21. Consider a program that input two integers having values in range (10, 250) and classifies them as even or odd. For this program generate :

i. Test cases using boundary values analysis

ii. Equivalence class testing **AKTU 2013-14, Marks 10**

Answer

i. Test cases using boundary value analysis :

1. It selects test cases at the boundaries of different equivalence classes.
2. For a function that classifies the integer as even and odd in the range of 10 and 250, test cases must include the values {11, 170, 249}.

ii. Test cases using equivalence class testing :

There are three equivalence classes.

One valid and two invalid equivalence classes are defined as follows :

Valid :

1. Set of integers in the range of 10 and 250.

Invalid :

1. The set of integers less than 10.
2. The set of integers larger than 250.

Que 4.22. Write the difference between black box testing and white box testing.

Answer

S. No.	Black box testing	White box testing
1.	Black box testing is the software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2.	This type of testing is carried out by testers.	This type of testing is carried out by software developers.
3.	Implementation knowledge is not required to carry out black box testing.	Implementation knowledge is required to carry out white box testing.
4.	Programming knowledge is not required to carry out black box testing.	Programming knowledge is required to carry out white box testing.
5.	The main aim of this testing is to check what functionality is performed by the system under test.	The main aim of white box testing is to check how system is performing.
6.	Testing is applicable on higher levels of testing like system testing, acceptance testing.	Testing is applicable on lower level of testing like unit testing, integration testing.
7.	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.

Que 4.23. What do you understand by test data suite preparation ?

Answer

1. A test suite is the accumulation of test cases that will run in a test progression until some stopping criteria are met.
2. Test suite preparation involves the construction and allocation of test cases in some systematic way based on the specific testing techniques used. For example, when coverage-based testing is planned, the specific coverage criteria would dictate the allocation of test cases.

3. There is also another way to receive a test suite. It is with the help of reuse of test cases for earlier versions of the equal product. Such type of software testing is generally considered as regression testing.
4. Commonly all the test cases should form an integrated suite, regardless of their background, in what way they are obtained and what models are used.
5. It happens so that the test suite may evolve over time and its creation may overlap with the actual testing.

Que 4.24. Why test cases are designed and should be the criteria for designing test cases ?

Answer

1. Test cases are designed to test both system and components. In test cases, input and predicted outputs are designed.
2. They are designed to discover program defects and to show that the new system meets its requirements.
3. The feature of the system or component which needs.
4. The testing team has to test several features while testing the test designs.

Criteria for designing test case :

1. **Functionality based test case design :**
   a. Sometimes, the team has to test the functionality of the system.
   b. So, it designs test cases to meet this requirement.
   c. The testing team designs such test cases with the help of domain and subject matter experts.
2. **Partition based test case design :**
   a. Sometimes, the test team notes that input given to a program and output so generated can be categorized into groups.
   b. In that case testing team categorizes the input and output as generated into groups.
   c. Here, case testing team partitions the input and output sets and pickup input from all the partitions. It saves the time and efforts of the teams.
   d. So, instead of testing the program for all possible values of input the team focuses on a few values chosen from each partition. This is known as partition based test case design.

   e. Test cases can be designed by picking a few values from both the partitions.
3. **Structure based test case design :**
   a. The programmers have to make sure that all the instructions of the program are executed at least once then in such situations, the programmers, as they know internal structure of the program, they have to design such kind of test cases.

Que 4.25. Explain alpha and beta testing.

Answer

Alpha testing :

1. Alpha testing is a type of acceptance testing; performed to identify all possible issues/bugs before releasing the product to everyday users or public.
2. The focus of this testing is to simulate real users by using black box and white box techniques.
3. The aim is to carry out the tasks that a typical user might perform. Alpha testing is carried out in a lab environment and usually the testers are internal employees of the organization.
4. To put it as simple as possible, this kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing.

Beta testing :

1. Beta testing is the procedure of subjecting software to test by real clients in the real surrounding before its release.
2. Beta testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external user acceptance testing.
3. Beta version of the software is released to a limited number of end-users to obtain feedback on the product quality.
4. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.
5. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of beta testing.
6. This testing helps to tests the product in real time environment.

Que 4.26. What is debugging ? Discuss various debugging approaches with examples.

Answer

Debugging is the process of identifying and resolving errors in a program code.

Debugging approaches :

1. **Brute force method :**
   a. This is the most common method of debugging but is the least efficient method.
   b. In this approach, the program is loaded with print statements to print the intermediate values with the hope that some of the printed values will help to identify the statement in error.
   c. This approach becomes more systematic with the use of a symbolic debugger (also called a source code debugger), because values of different variables and break points can be easily checked, and break points and watch points can be easily set to test the values of variables effortlessly.
2. **Backtracking :**
   a. In this approach, beginning from the statement at which an error symptom has been observed, the source code is traced backwards until the error is discovered.
   b. However as the number of source lines to be traced back increases, the number of backward paths increases and becomes unmanageably large, thus limiting the use of this approach.
3. **Cause elimination method :**
   a. In this approach, a list of causes which could possibly have contributed to the error symptom is developed and tests are conducted to eliminate each cause.
   b. A related technique of identification of the error from the error symptom is the software fault free analysis.

Que 4.27. Differentiate between testing and debugging.

Answer

S. No.	Testing	Debugging
1.	Starts with known conditions, user predefined procedures, predictable outcomes.	Unknown initial condition, end cannot be predicted.
2.	S uld be planned, designed, scheduled.	Cannot be constrained.
3.	Is a demonstration of errors/ apparent correctness.	Is a deductive process.

4.	Proves a programmer's failure.	Vindicates a programmer.
5.	Should strive to be predictable, dull, constrained, rigid.	Demands intuitive leaps, conjectures, experimentation, freedom.
6.	Much can be done without design knowledge.	Impossible without design knowledge.
7.	Can be done by outsider.	Must be done by insider.
8.	Theory of testing is available.	Only recently theorists have started.
9.	Much of the test design and execution can be automated.	Automation is not possible.

PART-3

Static Testing Strategies, Formal Technical Review (Peer Reviews), Walkthrough, Code Inspection, Compliance with Design and Coding Standards.

CONCEPT OUTLINE : PART-3

- Formal technical review is also known as peer review. It is a software quality assurance activity performed by software engineers.
- In walkthrough session, the material being examined is presented by a reviewer and evaluated by a team of reviewers.
- The goal of code inspection is to identify and remove bugs before testing the code.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.28. What is software review and formal technical review ?

Answer

Software review :

1. Software review is an effective way of filtering errors in a software product.

2. Reviews conducted at each of these phases *i.e.*, analysis, design, coding, and testing reveal areas of improvement in the product.
3. Reviews also indicate those areas that do not need any improvement.
4. We can use software reviews to achieve consistency and uniformity across products.
5. Reviews also make the task of product creation more manageable.
6. Some of the most common software review techniques are :
   a. Inspection
   b. Walkthrough
   c. Formal technical reviews (FTR)
   d. Code reviews
   e. Pair programming

FTR :

a. A formal technical review is software quality assurance activity performed by software engineers.

b. The objectives of FTR are :
   1. To uncover errors in function, logic or implementation for representation of software.
   2. To verify that software under review meets its requirements.
   3. To ensure that the software has been represented according to predefined standards.
   4. To achieve software that is developed in a uniform manner.
   5. To make projects more manageable.

c. In addition, the FTR serves as a training ground, enabling junior engineers to observe different approach to software analysis, design and implementation.

d. The FTR also serves to promote backup and continuity because a number of people became familiar with parts of the software that they may not have otherwise seen.

Que 4.29. What is a formal technical review and why is one conducted ? Outline the steps required to conduct a successful FTR.

AKTU 2013-14, Marks 05

Answer

Formal technical review : Refer Q. 4.28, Page 4–25C, Unit-4.

Steps in FTR :

1. **The review meeting :**
   a. Every review meeting should be conducted by considering the following constraints :

   i. Involvement of people
   ii. Advance preparation
   iii. Short duration

   b. Rather than attempting to review the entire design walkthrough are conducted for modules or for small group of modules.

   c. The focus of the FTR is on work product (a software component to be reviewed). The review meeting is attended by the review leader, all reviewers and the producer.

   d. One of the reviewers become recorder who records all the important issues raised during the review. When errors are discovered, the recorder notes each error.

   e. At the end of the review, the attendees decide whether to accept the product or not, with or without modification.

2. **Review reporting and record keeping :**
   a. During the FTR, the reviewer actively records all the issues that have been raised.
   b. At the end of meeting, these all raised issues are consolidated and review issues list are prepared.
   c. Finally, formal technical review summary report is produced.

3. **Review guidelines :**
   a. Guidelines for the conducting of formal technical review must be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed.

Que 4.30. What is mean by "Formal Technical Review" ? Should it access both programming style as well as correctness of software ? Give reasons.

AKTU 2014-15, Marks 05

AKTU 2016-17, Marks 10

Answer

FTR : Refer Q. 4.28, Page 4–25C, Unit-4.

No, formal technical review should focus on correctness only.

Reason :

1. Assessing style is tricky and can lead to bad feelings if a reviewer is not careful when he/she makes comments concerning style.
2. If the producer gets the feeling that the reviewer is saying, "Do it like I do", It is likely that some resentment will arise.

Que 4.31. What is walkthrough ? When we can perform the walkthrough and what are the objectives of walkthrough ?

Answer

1. Walkthrough is an activity in which author describes and explains the work product in an informal meeting to his peers or supervisor to get feedback.
2. Here validity of the proposed solution for work product is checked.
3. Structured walkthrough technique is very useful technique to analyze a product for its effectiveness.
4. In design phase of the product, the purpose of walkthrough is to find out as many possible problems in product design while the design is on paper.
5. It is cheaper to make changes when the design is on paper rather than at the time of conversion.
6. Generally walkthrough can be done at any of the following stages of product development :
   i. At the time of deciding schedule for different phases
   ii. At the time of problem specification
   iii. Designing data structure
   iv. Program designing
   v. Preparing documentation and user manual
   vi. Coding
   vii. Test plan, data and result
   viii. Maintenance changes
7. So, the walkthrough can start in early stage of software development as design and planning, long before the testing begins.
8. It is a static method of quality assurance. Walkthrough are informal meetings but with purpose.

A walkthrough has two broad objectives :

1. To gain feedback about the technical quality or content of the document.
2. To familiarize the audience with the content.

Que 4.32. What do you mean by code inspection ? Which types of errors can be removed by code inspection ?

Answer

1. Code inspection is a process of examining the code of program for identification of certain errors which are not identifiable by code walkthrough.
2. Code inspection is done to find out some common types of errors caused due to misunderstanding and improper programming.

3. The inspection of a work product is done by a group of peers, who first inspect the product privately and then get together in a formal meeting to discuss potential defects found by individuals and to detect more defects.
4. An inspection can be held for any technical product, which may include requirements specifications, system design document, detailed design, code, and test plan.
5. During identifying errors through code inspection, the standard of coding is also checked.

Following programming errors can be removed during code inspection :

1. Use of un-initialized variables.
2. Jumps within loop (use of transfer control statements (goto) to transfer execution control of program in between the loop *i.e.*, for loop, while-loop, do-while loop etc).
3. Non-terminating loops.
4. Mismatched assignment.
5. Array indices out of bounds (size of array is not initialized).
6. Improper storage allocation and de-allocation.

Que 4.33. Why code review is considered to be a more efficient way to remove errors from code ? Briefly discuss about code inspection and code walkthrough. **AKTU 2012-13, Marks 10**

Answer

Reason : The reason behind why code review is a much more cost-effective strategy to eliminate errors from code as compared to testing is that reviews directly detect errors. On the other hand, testing only helps detect failures and significant effort needs to be spent in debugging to locate the error.

Code inspection : Refer Q. 4.32, Page 4–28C, Unit-4.

Code walkthrough : Refer Q. 4.31, Page 4–28C, Unit-4.

Que 4.34. What do you understand by the cleanroom strategy ?

Answer

Cleanroom strategy :

1. The cleanroom strategy is a software development process intended to produce software with a certifiable level of reliability.
2. The focus of the cleanroom strategy is on defect prevention, rather than defect removal.

The cleanroom approach to software development is based on five key strategies :

1. **Formal specification** : The software to be developed is formally specified. A state-transition model which shows system responses to stimuli is used to express the specification.
2. **Incremental development** : The software is partitioned into increments which are developed and validated separately using the cleanroom process. These increments are specified, with customer input, at an early stage in the process.
3. **Structured programming** : Only a limited number of control and data abstraction constructs are used. The program development process is a process of stepwise refinement of the specification. A limited number of constructs are used and the aim is to apply correctness-preserving transformations to the specification to create the program code.
4. **Static verification :** The developed software is statically verified using rigorous software inspections. There is no unit or module testing process for code components.
5. **Statistical testing of the system :** The integrated software increment is tested statistically to determine its reliability. These statistical tests are based on an operational profile which is developed in parallel with the system specification.

Que 4.35. Describe coding standard in brief.

Answer

Coding standards are a set of conventions that the programmer follows to standardize their computer code to some degree and to make the overall program easier to read and understand.

Representative coding standards :

1. **Rules for limiting the use of globals :** These rules list what types of data can be declared global and what can not, with a view to limit the data that needs to be defined with global scope.
2. **Standard headers to precede the code of different modules :** The information contained in the headers of different modules should be standard for an organization. The exact format in which the header information is organized in the header is specified. The following is an example of header format :
   i. Name of the module
   ii. Date on which the module was created
   iii. Author's name
   iv. Modification history
   v. Synopsis of the module
   vi. Different functions supported in the module, along with their input/output parameters
   vii. Global variables accessed/modified by the module

3. **Naming conventions for global variables, local variables, and constant identifiers :** A naming convention is that variables are named using mixed case lettering. Global variable names would always start with a capital letter (for example, GlobalData) and local variable names start with small letters (for example, localData). Constant names should be formed using capital letters only (for example, CONSTDATA).
4. **Conventions regarding error return values and exception handling mechanisms :** The way error conditions are reported by different functions in a program should be standard within an organization. For example, all functions while encountering an error condition should either return a 0 or 1 consistently, independent of which programmer has written the code.

Que 4.36. What is difference between coding standards and coding guidelines ? Why are these considered important in software development organization ? Write down five important coding standards and guidelines that you would recommend.

AKTU 2014-15, Marks 10

Answer

Difference :

S.No.	Coding standards	Coding guidelines
1.	Coding standards are adherence to a well-defined and common style of coding.	Coding guidelines are a set of best practices.
2.	They are mandatory to be followed.	They are not mandatory.

Importance :

1. A coding standard and guidelines give a uniform appearance to the code written by different engineer.
2. They facilitate code understanding.
3. They promote good programming practices.

Coding standards : Refer Q. 4.35, Page 4–30C, Unit-4.

Coding guidelines :

The following are some representative coding guidelines that are recommended by many software development organizations :

1. Do not use a coding style that is too clever or too difficult to understand.
2. Avoid obscure side effects.
3. Do not use an identifier for multiple purposes.
4. The code should be well-documented.

5. The length of any function should not exceed 10 source lines.
6. Do not use GOTO statements.

Que 4.37. What is coding standard ? Also, discuss the different types of code reviews.

Answer

Coding standard : Refer Q. 4.35, Page 4–30C, Unit-4.

Types of code review :

Reviews can be classified as follows :

a. Formal reviews :

1. Formal reviews are conducted at the end of each life cycle phase.
2. They are held when the author believes that the product is error free.
3. A formal review follows a published process. The findings of the review are documented.
4. Formal reviews include the SRS review, the software design documentation review, and the software test readiness review.
5. Results of formal reviews are then documented.

b. Informal reviews :

1. Informal reviews are conducted on an as-needed basis.
2. They may be held at any time, serving the purpose of a brainstorming session and following no set agenda.
3. No documentation of the meeting is necessary.
4. The developer chooses the reviewers and provides the material to be reviewed.
5. The material may be as informal as a computer listing or handwritten documentation.
6. Informal reviews are conducted depending upon the need and their results are not documented.
7. The purpose of informal reviews is just the exchange of information.

Que 4.38. Why the testing is important in the software development life cycle ? Explain black box and white box testing in detail ?

Answer

Software testing is very important in SDLC because of the following reasons :

1. Software testing points out the defects and errors that were made during the development phases.
2. It is very important to ensure the quality of the product. Quality product delivered to the customers helps in gaining their confidence.

3. Testing is necessary in order to provide the facilities to the customers like the delivery of software application which requires lower maintenance cost, is more accurate, consistent and reliable.
4. Software testing is required for an effective performance of software application or product.
5. Software testing ensures that the application should not result into any failures because cost of fixing is larger in the later stages of the development.

Black box testing : Refer Q. 4.18, Page 4–17C, Unit-4.

White box testing : Refer Q. 4.16, Page 4–15C, Unit-4.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Discuss the objectives and principles of software testing.
Ans. Refer Q. 4.1 and Q. 4.2.

Q. 2. Define software testing. Explain various level of testing.
Ans. Refer Q. 4.12.

Q. 3. Explain the concept of white box testing and black box testing.
Ans. Refer Q. 4.16 and Q. 4.18.

Q. 4. Differentiate between testing and debugging.
Ans. Refer Q. 4.27.

Q. 5. Describe the following :
i. **FTR**
ii. **Walkthrough**
iii. **Code inspection**

Ans.
i. Refer Q. 4.28.
ii. Refer Q. 4.31.
iii. Refer Q. 4.32.

Q. 6. Write short note on coding standard and coding guideline.
Ans. Refer Q. 4.35 and Q. 4.36.

5 UNIT

Software Maintenance and Software Project Management

PART-1

Software as an Evolutionary Entity, Need for Maintenance, Categories of Maintenance : Prevention, Corrective and Perfective Maintenance, Cost of Maintenance, Software Re-engineering, Reverse Engineering.

CONCEPT OUTLINE : PART-1

- Software maintenance can be defined as a set of activities undertaken on a software system following its release for operational use.
- Categories of maintenance are :
  i. Perfective maintenance
  ii. Adaptive maintenance
  iii. Corrective maintenance
- Maintenance costs vary widely from one application to another but on average, they seem to be between two to four times of the development cost for large embedded software systems.
- Reverse engineering is the process of analysing software with the objective of recovering its design and specification.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.1. Describe the term software evolution. Explain the different laws used for software evolution.

Answer

1. Software evolution is the term which refers to the process of developing software initially, then repeatedly updating it for various reasons.
2. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.
3. The cost and impact of these changes are accessed to see how much system is affected by the change and how much it might cost to implement the change.
4. If the proposed changes are accepted, a new release of the system is planned. During release planning, all proposed changes (fault repair, adaptation and new functionality) are considered.

5. A decision is then made on which changes to implement in the next version of the system. The process of change implementation is an iteration of the development process where the revisions to the system are designed, implemented and tested.

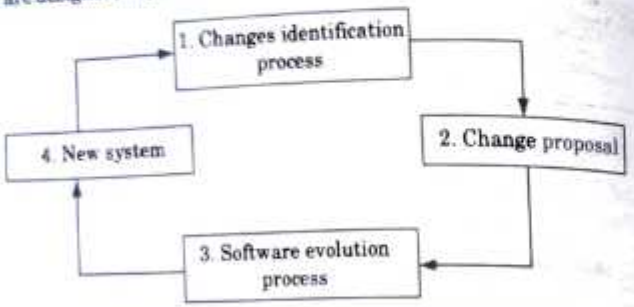


Fig. 5.1.1. Change identification and evolution process.

Laws used for software evolution are :

1. **Law of continuing change :** This law state that any software system that represent some real world reality undergoes continuous change or become progressively less useful in that environment.
2. **Law of increasing complexity :** As an evolving program changes, its structure become more complex unless effective efforts are made to avoid this phenomenon.
3. **Law of large program evolution :** Program evolution is a self-regulating process and measurement of system attribute such as size, time between releases, number of reported errors etc., reveals statistically significant trends and variances.
4. **Law of conservation of organization stability :** Over the lifetime of a program, the rate of development of that program is approximately constant and independent of the resource devoted to system development.
5. **Law of conservation of familiarity :** This law states that during the active lifetime of the program, changes made in successive release are almost constant.

Que 5.2. What is software maintenance ? What is the need of the software maintenance ? AKTU 2014-15, Marks 05

Answer

1. According to the standard for software maintenance-1992, software maintenance is modification of software product after delivery to correct faults, to improve performance of other attributes or to adopt the product to a modified environment.
2. Software maintenance is the process of changing system after it has been delivered to the customer.

3. The cost of software maintenance is quite high.
4. Software maintenance accounts more than 65% cost of software development cost.

Need for software maintenance :

The software needs maintenance due to following reasons :

i. Over a period of time, software's original requirements may change to reflect the customer's need.

ii. Errors undetected during software development may be found during the use and require correction.

iii. With time, new technologies are introduced such as new hardware, operating system etc. The software therefore must be modified to adopt the new modified environment.

Que 5.3. What is software maintenance ? Explain any two models of software maintenance. AKTU 2013-14, Marks 05

Answer

Software maintenance : Refer Q. 5.2, Page 5–3C, Unit-5.

Models of software maintenance :

1. **Quick-fix model :**
   a. This is an adhoc approach used for maintaining the software system.
   b. The objective of this model is to identify the problem and then fix it as quickly as possible.

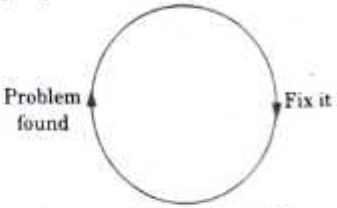


Fig. 5.3.1. Quick fix model.

   c. The advantage is that it performs its work quickly and at a low cost.
   d. This model is an approach to modify the software code with little consideration for its impact on the overall structure of the software system.
2. **Iterative enhancement model :**
   a. Iterative enhancement model considers the changes made to the system are iterative in nature.
   b. This model incorporates changes in the software based on the analysis of the existing system.

c. It assumes complete documentation of the software is available in the beginning.

d. Moreover, it attempts to control complexity and tries to maintain good design.

e. Iterative enhancement model is divided into three stages :

i. Analysis of software system

ii. Classification of requested modifications

iii. Implementation of requested modifications

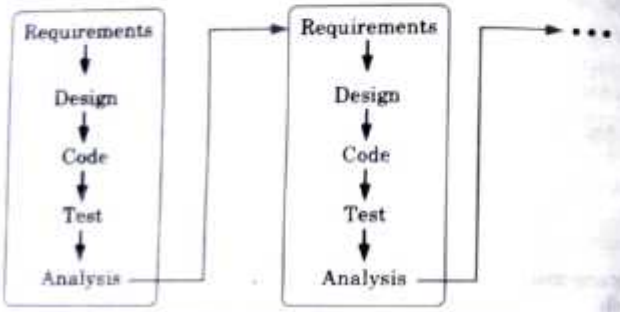


Fig. 5.3.2. Iterative enhancement model.

Que 5.4. What is software maintenance ? Discuss different types of maintenance of that a software product might need.

Answer

Software maintenance can be categorized into following four types :

1. **Adaptive maintenance :**
   a. This type of maintenance concerns external changes.
   b. Even if the software is error free, it is possible that the environment in which the software system works will often change.
2. **Corrective maintenance :**
   a. Corrective maintenance refers to modifications initiated by defects in the software.
   b. This type of maintenance is also called bug fixing.
3. **Perfective maintenance :**
   a. Perfective maintenance is an activity that we undertake to improve the maintainability, performance and other attributes of our applications.
   b. It includes all changes like insertions, deletions, modifications, extension and enhancement made to the application to meet evolving user need.

4. **Preventive maintenance :**
   a. This type of maintenance is done to anticipate future problems and to improve the maintainability using techniques like documenting, commenting, or even re-implementing some part of software using modern software engineering tools and techniques.

Que 5.5. Why is software maintenance required ? Discuss with examples. **AKTU 2015-16, Marks 10**

Answer

There are three types of maintenance and following are the reasons for why they are required along with their examples :

1. **Perfective maintenance :** It is required for improving the performance of the software.

   Examples :
   a. Configuring the network management software to improve performance such as improving access times to data, speed at which reports are produced, etc.
   b. Software may need to be modified to improve the user interface upon feedback from users who are finding it more difficult to use than it needs to be.
   c. Developing online tutorials and more help screens to help new staff learn the software.
   d. The software provider provides upgrades which will improve the performance of the software.
2. **Corrective maintenance :** It is required for correcting bugs in the software which were not discovered during testing.

   Example :
   a. A piece of software may crash when being used with another piece of software.
   b. A piece of software may crash when used with a particular item of hardware.
   c. Software may present a security risk which needs correction.
   d. Problems with reports not being printed out properly.
3. **Adaptive maintenance :** It is required when software may need to be changed owing to the changing needs of the business or organization.

   Example :
   a. Software may need altering so that it is more flexible in supplying the managers with information which was not envisaged at the time of development.
   b. Changes to values such as the percentage rate of VAT or changes to income tax rates will result in changes to the software.

c. The organisation expands so the software needs to be altered so it is able to cope with an increased number of users.

d. Adapting the software to work with newly developed operating systems software or new hardware.

e. A virus threat/hacker threat means that the software will need to be adapted to protect against this.

Que 5.6. Which category of software maintenance consumes maximum effort and why ?

Answer

1. Perfective maintenance consumes maximum effort. Since this type of maintenance requires perfecting the software by implementing requirements.
2. It also means maintaining the functionality of system by improving its structure and performance.
3. This type of maintenance is necessary when the system requirements change in response to organizational or business change.
4. The scale of change required for this type of maintenance is greater than other types of maintenance.

Que 5.7. Explain cost and efforts of software maintenance.

Answer

1. The cost of system maintenance represents a large proportion of the budget of most organizations that use software system.
2. More than 65% of software lifecycle cost is expended in the maintenance activities.
3. Cost of software maintenance can be control by postponing the development opportunity of software maintenance but this will cause following intangible cost :
   i. Customer dissatisfaction when requests for repair or modification cannot be addressed in a timely manner.
   ii. Reduction in overall software quality as a result of changes that introduce hidden errors in maintained software.
4. Efforts expended on maintenance may be divided into productivity activities (for example analysis and evaluation, design and modification, coding). The following expression provides a module of maintenance efforts.

Where, $M = P + K(C - D)$

M = Total efforts expended on maintenance.
P = Productive effort.
K = An empirical constant.

C = A measures of complexity that can be attributed a lack of good design and documentation.
D = A measures of degree of familiarity with the software.

Que 5.8. Explain software re-engineering and the steps involved in re-engineering process.

Answer

Software re-engineering :

1. Software re-engineering is the examination and alteration of a system to reconstitute it in a new form.
2. The principles of re-engineering when applied to software development processes, is called software re-engineering.
3. If affects positively at software cost, quality, service to the customer and speed of delivery.
4. In software re-engineering we are improving the software to make it efficient and effective.

Steps of software re-engineering process :

1. **Inventory analysis :**
   a. Every software organization should have an inventory of all applications.
   b. The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description (for example, size, age, business, and criticality) of every active application.
   c. By sorting this information according to business criticality, longevity, current maintainability, and other locally important criteria, candidates for re-engineering appear.
   d. Resources can then be allocated to candidate applications for re-engineering work.
2. **Document restructuring :**
   a. Documentation of a system either explains how it operates or how to use it.
   b. Documentation must be updated.
   c. It may not be necessary to fully re-document an application. Rather, those portions of the system that are currently undergoing change are fully documented. Over time, a collection of useful and relevant documentation will evolve.
   d. The system is business critical and must be fully re-documented

3. **Reverse engineering :**
   a. Reverse engineering is a process of design recovery. Reverse engineering tools extract data, architectural, and procedural design information from an existing program.

4. **Code restructuring :**
   a. To accomplish code restructuring, the source code is analyzed using a restructuring tool. Violations of structured programming constructs are noted, and code is then restructured.
   b. The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced. Internal code documentation is updated.

5. **Data restructuring :**
   a. Data restructuring begins with a reverse engineering activity.
   b. Current data architecture is dissected, and necessary data models are defined.
   c. Data objects and attributes are identified, and existing data structures are reviewed for quality.

6. **Forward engineering :**
   a. Forward engineering, also called renovation or reclamation not only recovers design information from existing software, but uses this information to alter or reconstitute the existing system in an effort to improve its overall quality.

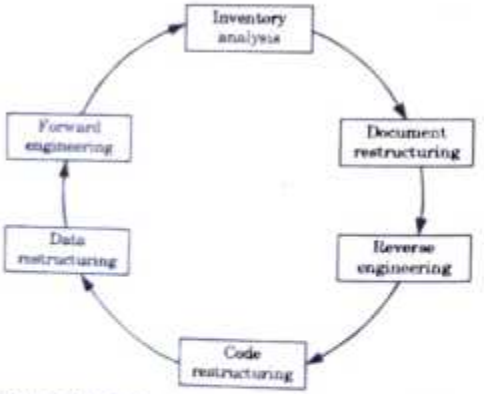


Fig. 5.8.1. Software re-engineering process model.

Que 5.9. What are the benefits of re-engineering phases ? Explain.

Answer

Benefits of re-engineering phases :

1. **Costs :** Evidence from a number of project suggest that re-engineering of an existing software system costs significantly less than new system development.
2. **Lower risks :** Software re-engineering is based on incremental improvement of systems, rather than radical system replacement. The risk of losing critical software knowledge is drastically reduced.
3. **Better use of existing staff :** The skill of existing staff can be better utilized by re-engineering.
4. **Incremental development :** The development is not carried out as a whole. It is in stages.

Que 5.10. What are legacy systems ? Why do they require re-engineering ? Describe briefly the steps required for re-engineering a software product.

Answer

Legacy system :

1. Legacy systems are old systems that must still be maintained. These legacy systems may have been the best that technology had to offer.

Reason for requirement of re-engineering :

1. Most of the legacy systems may be poorly structured and their documentation may be either out of date or non-existent.
2. The developers of these systems have left the organization; there may be no one in the organization who really understands these systems in detail.
3. These systems were also not designed for change. Another problem is the non-availability of requirements, design and test cases. So, the legacy system requires software re-engineering.

Steps of re-engineering process : Refer Q. 5.8, Page 5–8C, Unit-5.

Que 5.11. Write short notes on software reverse engineering.

OR

Write short notes on software re-engineering and software reverse engineering.

Answer

Software re-engineering : Refer Q. 5.8, Page 5–8C, Unit-5.

Software reverse engineering :

1. Software reverse engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code.
2. The purpose of reverse engineering is to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Steps of software reverse engineering :

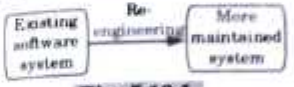
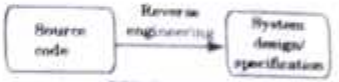
1. **Collecting information :** This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.
2. **Examining the information :** The information collected in step 1 is studied so as to get familiar with the system.
3. **Extracting the structure :** This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.
4. **Recording the functionality :** During this step processing details of each module of the structure charts are recorded using structured language like decision tables etc.
5. **Recording data flow :** From the information extracted at step 4 and step 5, set of data flow diagrams are derived to show flow of data among the processes.
6. **Recording control flow :** High level control structure of the software is recorded.
7. **Review extracted design :** Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.
8. **Generate documentation :** Finally in this step the complete documentation including SRS, design document, history, overview etc., is recorded for future use.

Que 5.12. What is difference between re-engineering and reverse engineering ? Explain different steps of re-engineering.

AKTU 2013-14, Marks 05

Answer

The differences between re-engineering and reverse engineering are as follows :

S. No.	Re-engineering	Reverse engineering
1.	Software re-engineering is concerned with taking existing legacy systems and re-implementing them to make them more maintainable.  **Fig. 5.12.1.**	Reverse engineering is a process of design recovery.  **Fig. 5.12.2.**
2.	The objective of re-engineering is to produce a new more maintainable system.	The objective of reverse engineering is to derive the design or specification of a system from its source code.

Steps of re-engineering : Refer Q. 5.8, Page 5–8C, Unit-5.

PART-2

Software Configuration Management Activities, Change Control Process, Software Version Control, An Overview of Case Tools.

CONCEPT OUTLINE : PART-2

- Software configuration management is designed to manage an evolving software system or product and then maintain it in the post development phase throughout its life cycle.
- Four major SCM activities are :
  - i. Configuration identification
  - ii. Change control
  - iii. Software configuration status reporting
  - iv. Audits and reviews
- CASE tools are software programs that are designed to assist human programmers with the complexity of the processes and the artifacts of software engineering.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.13. Write short note on software configuration management.

AKTU 2012-13, Marks 10

Answer

1. Software Configuration Management (SCM) can be defined as a process of defining and implementing a standard configuration, which results into the primary benefits such as easier setup and maintenance, less down-time, better integration with enterprise management, and more efficient and reliable backups and also maximize productivity by minimizing mistakes.
2. SCM is used to track and manage the emerging product and its versions.
2. SCM ensures that all people involved in the software process know what is being designed, developed, built, tested, and delivered.
3. Through SCM, the design requirements can be traced to the final software product.

Objectives of software configuration standards :

1. **Remote system administration :**
   a. The configuration standard should include necessary software and/or privileges for remote system administration tools.
   b. These remote tools can be used to check the version of virus protection, check machine configuration, or offer remote help-desk functionality.
2. **Reduced user down-time :**
   a. A great advantage of using a standard configuration is that system becomes completely interchangeable resulting in reduced user down-time.
3. **Reliable data backups :**
   a. Using a standard directory for user data allows backup systems to selectively backup a small portion of a machine, thus, greatly reducing the network traffic and tape usage for backup systems.
   b. Also, should a catastrophic failure occur, the data directory could be restored to a new machine with little time and effort.
4. **Easy workstation setup :**
   a. Any sort of standardized configuration streamlines the process of setting up the system and ensures that vital components are available.
   b. If multiple machines are being setup according to a standard setup, most of the setup and configuration can be automated.
5. **Multi-user support :**
   a. Although, it is common for users to share a workstation, the system configuration is designed to allow multiple users to use the same workstation without interfering each other's work.

Que 5.14. Illustrate the various SCM activities.

Answer

1. **Identification of configuration components :**
   a. This process requires us to freeze the baseline product.
   b. The product baseline further can be viewed and split into requirement driver design (RDD) baseline, design baseline and document baseline.
   c. The baseline product needs to be broken down into smaller components.
   d. These identifies components are the basic units for modification and control; hence, they are numbered and tagged for identification, access and control.
2. **Change management control (CMC) process :**
   a. CMC is triggered through a change requirement from user, customer or from developer.
   b. Configuration management group has to describe its utility and other technical and commercial implications and then make the decision of accepting the CR for implementation.
   c. All accepted CRs are then priorities and scheduled for effecting the change in a planned manner.
3. **Product status accounting and management :**
   a. The baseline product and baseline components undergo a change and assume new status.
   b. The role of product status accounting is to ensure that the product in its latest status is functional in all respects and all concerned are informed.

Que 5.15. What is change control process ? Explain.

Answer

1. Change control is a systematic approach to managing all changes made to a product or system.
2. Change control process ensures that the changes to the system are controlled and that their effect on the system can be predicted.
3. Change control process comes into effect when the software and associated documentation are delivered to configuration management change request form which should record the recommendations regarding the change.
4. The recommendations may include assessment of the proposed change, the estimated costs and how the change should be implemented.

5. This form is submitted to a Change Control Authority (CCA), which decides whether or not the change is to be accepted. If change is approved by the CCA, it is applied to the software.
6. The revised software is revalidated by the Software Quality Assurance (SQA) team to ensure that the change has not adversely affected other parts of the software.
7. The changed software is handed over to the software configuration team and is incorporated in a new version of the system.

Que 5.16. What is software version control process ? Explain the activities of version control process.

Answer

1. Version control combines procedure and a tool that manages different versions of configuration items that are created during the software engineering process.
2. A detailed record of every version of the software must be kept.
3. This comprises the name of each source code component, including the variations and revisions.

Version control activity is divided into four main sub-activities :

1. **Identifying new version :**
   a. A software configuration items (SCI) will receives a new version number when there has been a change to its established baseline.
   b. Each previous version will be stored in a corresponding directory such as version0, version1, version2 etc.
2. **Numbering scheme :**
   a. The numbering scheme will have the following format : Version X.Y.Z.
   b. The first letter (X) represents the entire SCI. Therefore changes made to the entire configuration items or changes large enough to warranty a completely new release of items will causes the first digit to increase.
   c. The second letter (Y) represents a component of SCI. The digit will sequentially increase if a change is made to a component, or small changes to multiple components.
   d. The third letter (Z) represents a section of components of SCI. This number will only be possible if component of an SCI can be broken down into individual sections.
3. **Visibility :**
   a. The version number will be visible either in a frame or below the title.

   b. The decision for this depends upon the group decision to code all the documents for a frame capable browser or allow for non frame capable browser.
   c. In either case, number will always be made available.
4. **Tracking :** The best way to keep track of the different versions is with a version evolution graph. As shown in Fig. 5.16.1 given below :

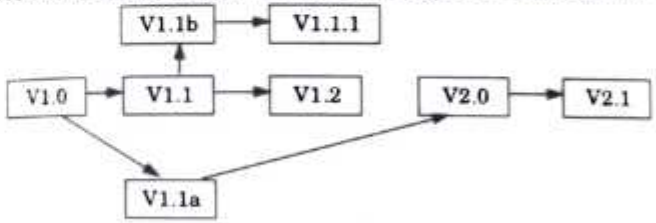


Fig. 5.16.1. Version evolution graph.

Que 5.17. Write a short note on CASE tools.

AKTU 2013-14, Marks 05

OR

Give and explain **the architecture of CASE environment.**

Answer

1. CASE tools are software programs that are designed to assist human programmers with the complexity of the processes and the artifacts of software engineering.
2. The use of Computer Aided Software Engineering (CASE) tool reduces the effort of development, of achieving quality goals and managing change and configuration throughout the product life cycle.
3. Computer Aided Software Engineering (CASE) tools help the project manager, the software developer and other key personnel to improve their productivity in the development team.

Architecture of CASE environment :

The important component of a modern CASE environment are :

i. User interface
ii. Tool set
iii. Object management system (OMS)
iv. A repository

1. **User interface :** The user interface provided a consistent framework for accessing the different tools thus making it easier for the users to interact with the different tools and hence reducing the learning period.
2. **Object management system :** Different CASE tools represent the software product as a set of entities such as specification, design, test

data, project plan etc. The object management system maps these logical entities into the underlying storage management system (repository).

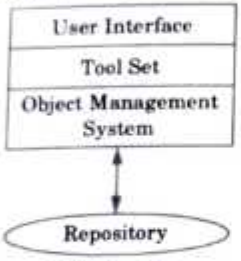


Fig. 5.17.1.

Repository :

1. The complete information about meta-models, methods, project artifacts, baseline, process models etc., is available in the repository.
2. This information can be shared by different team members efficiently thereby improving the project management as a whole.
3. While implementing, the repository can be either centralized or distributed. Normally this is implemented using a database or a file system.

Que 5.18 **What are benefits and limitations of CASE tools ? Explain.**

Answer

Benefits of CASE tools :

1. Improved productivity
2. Better documentation
3. Reduced lifetime maintenance
4. Improved accuracy
5. Opportunity to non-programmers
6. Intangible benefits

Limitations of CASE tools :

1. Cost
2. Learning curve
3. Tool mix : It is important to make an appropriate selection of tool mix to get cost advantage.

Que 5.19. **List out some most used CASE tools.**

Answer

Some popular CASE tools supporting different stages of software life cycle are :

1. **Software requirements tools :** A number of tools are proposed for modeling, tracing, and analyzing requirements.

   Examples are :

   a. Turbo-analyst
   b. Oracle's Designer/2000, Argo/UML Rational ROSE

2. **Software construction tools :** Software construction tools are the tools which are used to code and implement the software and hence transform the software requirements into working product. Broadly they can be classified as :

   a. Program editors
   b. Compilers
   c. Interpreters
   d. Debuggers

3. **Software maintenance tools :** Tools under this category are :

   a. **Comprehensive tools :** These tools are used to assist in human comprehension and visualization of the programs. Examples are Visual Studio, exref etc.

   b. **Re-engineering tools :** These are tools which allow the change of existing format of a program to a new format *i.e.*, a new language or new database or new technology in general.

4. **Configuration management tools :** These tools support version control and other activities related to configuration management *i.e.*, management and control of changes made to the documents. Examples are Clear Case, Change man etc.

5. **Project management tools :** Tools under this category automate size estimations, cost estimation, schedule estimation, risk management activities etc. Some of the tools are MS Project, Excel, COCOMO, FPA etc.

PART-3

Estimation of Various Parameters such as Cost, Efforts, Schedule/ Duration, Constructive Cost Models (COCOMO), Resource Allocation Models, Software Risk Analysis and Management.

CONCEPT OUTLINE : PART-3

- Productivity is defined as :

$$\text{Productivity} = \frac{\text{Size}}{\text{Effort}}$$

- Constructive cost model is the most widely used estimating models techniques. Boehm postulated that there are essentially three important classes of software products :
  i. Organic
  ii. Semidetached
  iii. Embedded
- Risk is a problem that could cause some loss or threaten the success of the software project, but which has not happened yet.
- Risk can be categorized into :
  i. Project risk
  ii. Product risk
  iii. Business risk

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.20. What do you mean by effort estimation ? Why effort estimation is necessary ?

Answer

1. Software development effort estimation is the process of predicting the most realistic amount of effort (expressed in terms of person-hours or money) required to develop or maintain software based on incomplete and uncertain input.
2. Based on the effort, the cost and time required to complete the development is estimated.
3. Effort will depend upon the productivity.
4. In software development, productivity is that of the software engineer and the effort is that of the term.
5. Productivity is defined as

$$\text{Productivity} = \frac{\text{Size}}{\text{Effort}}$$

Software effort estimation is important because of following reasons :

1. Organizations have proper control over project and they can plan systematically.
2. There is a clear understanding of the product.
3. Estimation also determines the project feasibility in terms of budget and time constraints.
4. It helps in identification of resources to be used during the project.
5. Estimation also helps management in taking decision for current as well as future projects.
6. Estimation helps in quantifying the impact of risks and guides the project manager to take suitable decision.

Que 5.21. What do you mean by activity networks ? Also, name different types of activity network ?

Answer

1. An activity network is a graphical representation that shows the different activities making up a project, their estimated durations, and interdependencies.
2. Each activity is represented by a rectangular node and the duration of the activity is shown alongside each task.
3. The technique can be broken down into three stages :
   a. **Planning :** It identifies tasks and estimate duration of times.
   b. **Scheduling :** Establish time table of start and finish times.
   c. **Analysis :** Establish the float and evaluate and revise as necessary.
4. There are three types of activity networks diagram. They are :
   a. Program Evaluation and Review Techniques (PERT)
   b. Critical Path Method (CPM)
   c. Gantt chart

Que 5.22. What do you understand by Project Evaluation Review Technique (PERT) ? Also, enlist the benefits and limitations of PERT.

Answer

1. A Project Evaluation and Review Technique (PERT) chart is a project management tool used to schedule, organize, and coordinate tasks within a project.
2. PERT can be both a cost and a time management system.
3. PERT is organized by events and activities or tasks.
4. PERT charts depict task, duration, and dependency information.

5. Each chart starts with an initiation node from which the first task, or tasks originates.
6. If multiple tasks begin at the same time, they are all started from the node or branch, or fork out from the starting point.
7. Each task is represented by a line, which states its name or other identifier, its duration, the number of people assigned to it, and in some cases the initials of the personnel assigned.
8. The other end of the task line is terminated by another node, which identifies the start of another task, or the beginning of any slack time, that is, waiting time between tasks.

Benefits of PERT :

1. The PERT network is continuously useful to project managers prior to and during a project.
2. The PERT network's graphical representation of the project's tasks helps to show the task interrelationships.
3. PERT controls time and costs during the project and also facilitates finding the right balance between completing a project on time and completing it within the budget.
4. It exposes all possible parallelism in the activities and thus helps in allocating resources.
5. It allows scheduling and simulation of alternative schedules.

Limitations of PERT :

1. The PERT network does not deal very well with task overlap.
2. PERT does not help in deciding which activities are necessary or how long each will take.

Que 5.23. What do you meant by Critical Path Method (CPM) ? What are the benefits and limitations of CPM ?

Answer

1. Critical Path Method is step-by-step project management technique for process planning that defines critical and non-critical path.
2. Critical Path Method (CPM) charts are similar to PERT charts and are sometimes known as PERT/CPM.
3. CPM acts as the basis both for preparation of a schedule, and of resource planning.
4. Critical path analysis is an effective and powerful method of assessing :
   a. What tasks must be carried out ?
   b. Where parallel activity can be performed ?
   c. The shortest time in which we can complete a project.
   d. Resources needed to execute a project.

   e. The sequence of activities, scheduling and timings involved.
   f. Task priorities.
   g. The most efficient way of shortening time on urgent projects.
5. During management of a project, it allows to monitor the achievements of project goals. It also helps to see where remedial action needs to be taken to get a project back on course.
6. In a CPM chart, the critical path is indicated.
7. The critical path determines the total duration of the project. If a task on the critical path is delayed, the final completion of the project will likely to be delayed.

Benefits of CPM :

a. It identifies the task that must be completed on time for the whole project to be completed on time.
b. It also identifies which tasks can be delayed for a while if resource needs to be reallocated to catch up on missed tasks.
c. CPM helps to minimize cost.

Limitations of CPM :

a. The relation of tasks to time is not immediate.
b. CPM charts are more difficult to understand.

Que 5.24. What is Gantt chart ? Discuss advantages and disadvantages of Gantt charts.

Answer

1. A Gantt chart is a special type of bar chart where each bar represents an activity.
2. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity.
3. Gantt charts are mainly used to allocate resources to activities.
4. Gantt charts are useful tools for planning and scheduling projects.
5. They allow to assess how long a project should take, determine the resources needed, and lay out the order in which tasks need to be carried out.
6. They are useful in managing the dependencies between tasks.
7. When a project is under way, Gantt charts are useful for monitoring its progress.
8. We can immediately see what should have been achieved at a point in time, and can therefore take remedial action to bring the project back on course.

Advantages of Gantt chart :

1. This is a simple and very inexpensive method and can be developed even by supervisory staff with some amount of training.
2. These charts clearly show the decided time and work schedules for every job.
3. Monitoring and control are easier and can be done within a minimum time frame and at the lowest cost.
4. These charts can be changed and updated quickly at a lower cost.

Disadvantages of Gantt chart :

1. They do not show job interrelationships and interdependence.
2. Cost implications cannot be shown.
3. With these charts, it is not possible to depict other alternatives for project completion.
4. The shape and form of Gantt charts can differ according to the nature of the requirement.

Que 5.25. Discuss the various categories of software development projects according to the constructive cost models (COCOMO) estimation model.

OR

Write a short note on Constructive Cost Models (COCOMO).

AKTU 2012-13, Marks 10

Answer

Various categories of software development projects according to COCOMO estimation models are :

1. Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity : organic, semidetached, and embedded.
2. Boehm's definitions of organic, semidetached, and embedded systems are as follows :
   a. **Organic :** We can consider a development project to be of organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.
   b. **Semidetached :** A development project can be considered to be of semidetached type, if the development team consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

   c. **Embedded :** A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if stringent regulations on the operational procedures exist.

COCOMO cost estimation models :

According to Boehm, software cost estimation should be done through following three stages :

1. **Basic COCOMO model :**
   a. The basic COCOMO model gives an approximate software development efforts and cost as function of program size expressed in estimated lines of code.
   b. The basic COCOMO estimation model is given by the following expressions :

      Effort $(E) = a^*(\text{KLOC})^b$

      Development Time $(T_{dev}) = c^*(E)^d$

      Where E is effort applied in person-month, T_{dev} is development time in months.
   c. The coefficients a, b, c, d are constant and can be calculated by given table :

Project	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

   d. When effort and development time are known, the average staff size to complete the project may calculated as :
      Average staff size $(SS) = E/T_{dev}$ persons
   e. When project size is known, the productivity level may be calculated as :
      Productivity $(P) = \text{KLOC}/E$
2. **Intermediate COCOMO :**
   a. The intermediate COCOMO model refines the initial estimate obtained through the basic COCOMO expressions by using a set of fifteen cost drivers (multipliers) based on various attributes of software development.
   b. Boehm requires the project manager to rate these fifteen different parameters for a particular project on a scale of one to three.
   c. Then depending on these ratings, he suggests appropriate cost driver values which should be multiplied with the initial estimate obtained using the basic COCOMO.

d. In general, the cost drivers can be grouped into four categories :
   1. Product attributes
   2. Computer attributes
   3. Personnel
   4. Project attribute

e. The intermediate COCOMO equations are :

Effort (E) = a*(KLOC)^b* EAF
Development Time (T_{dev}) = c*(E)^d
Where EAF (Effort Adjustment Factor) is multiplication of different types of cost drivers. E is an effort applied in person-month, T_{dev} is development time in months.

f. The coefficients a, b, c, d are constant and can be calculated by given table :

Project	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

3. Detailed/ complete COCOMO model :

a. A major shortcoming of both the basic and the intermediate COCOMO models is that they consider a software product as a single homogeneous entity.

b. However, most large systems are made up of several smaller subsystems. These subsystems may have widely different characteristics.

c. For example, some subsystems may be considered as organic type, some semidetached, and some embedded.

d. Not only that the inherent development complexity of the subsystems may be different, but also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development, and so on.

e. The complete COCOMO model considers these differences in the characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems.

f. The cost of each subsystem is estimated separately. This approach reduces the margin of an error in the final estimate.

COCOMO-II :

1. COCOMO-II is the revised version of the original COCOMO.

2. It provides a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.
3. COCOMO-II provides three detailed cost estimation models. These can be used to estimate project costs at different phases of the software.
4. As the project progresses, these models can be applied at different stages of the same project.
   a. **Application composition :** This model as the name suggests, can be used to estimate cost for prototyping, for example, to resolve user interface issues.
   b. **Early design :** This supports estimation of cost at the architectural design stage.
   c. **Post architecture stage :** This provides cost estimation during detailed design and coding stage.

Que 5.26. Using schematic diagram and suitable example show the order in which the following are estimated in the COCOMO estimation technique :

cost, effort, duration, size

AKTU 2014-15, Marks 10

Answer

COCOMO : Refer Q. 5.25, Page 5–23C, Unit-5.

Example :

Problem : Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.

Solution : As per the basic COCOMO estimation formula for organic software :

Effort = 2.4 * (32)1.05 = 91 PM
Nominal development time = 2.5 * (91)0.38 = 14 months
Cost required to develop the product = 14 * 15,000 = Rs. 2,10,000/-

Que 5.27. What do you mean by risk ? Explain.

Answer

1. A risk is any anticipated unfavourable event or circumstance that can occur while a project is underway.
2. A risk is a problem that might occur from the problems currently being faced by a company.
3. If a risk becomes true, the anticipated problem becomes a reality and is no more a risk.

4. If a risk becomes real, it can adversely affect the project and hamper the successful and timely completion of the project.
5. Software risk is a problem that could cause some loss or threaten the success of a software project, but which has not happened yet.
6. These potential problems might have an adverse impact on the cost, schedule, or technical success of the software project, the quality of software products, or project team morale.

Risks can be categorized as follows :

a. **Project risks :** Risks that threaten the project (or the project schedule).
b. **Product risks :** Risks that threaten the quality of the software developed.
c. **Business risks :** Risks that threaten the development (or client) organization.

Que 5.28. Write short note on software risk analysis and management.

AKTU 2012-13, Marks 10

OR

What do you mean by software risk ? Discuss the risk management activities during software development.

Answer

Risk management is the process of identifying risk, assessing risk and taking steps to reduce this to an acceptable level.

Software risk : Refer Q. 5.27, Page 5–26C, Unit-5.

Risk management is a very tedious task. It involves basically two steps :

1. Risk assessment
2. Risk control

1. **Risk assessment :** It is the process of examining a project and identifying areas of potential risk. The risk assessment consists of three activities :
   a. **Risk identification :**
      i. Risk identification is a systematic attempt to specify threats to the project plan. The purpose of risk identification is to develop a list of risk items called risk statement.
      ii. Risk identification can be facilitated with the help of a checklist of common risk areas for software projects or by examining the contents of an organizational database of previously identified risks and mitigation strategies (both successful and unsuccessful).
      iii. Risk identification is carried out as a team process using brainstorming. To assist the process a list of risk types can be used.

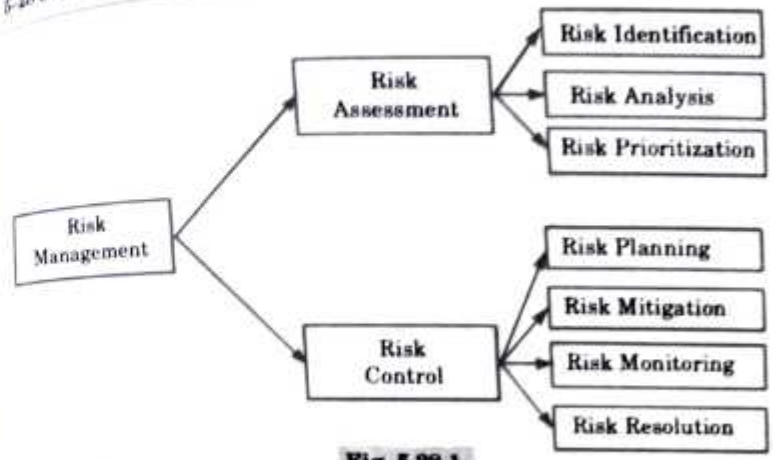


Fig. 5.28.1.

iv. The end product of this step of the process is a list of risks that could occur and affect the product, the process or the business.
v. Within the identification phase, several activities occur. The main activities are :
   1. Identify risks
   2. Define risk attributes
   3. Document
   4. Communicate

b. **Risk analysis :**
   i. When the risks have been identified, all items are analyzed using different criteria. The purpose of the risk analysis is to assess the loss probability and magnitude of each risk item.
   ii. The input is the risk statement and context developed in the identification phase.
   iii. The output of this phase is a risk list containing relative ranking of the risks and a further analysis of the description, probability, consequence and context.
   iv. The main activities in this phase are :
      1. Group similar risks
      2. Determine risk drivers
      3. Determine source of risks
      4. Estimate risk exposure
      5. Evaluate against criteria

c. **Risk prioritization :**
   i. Risk prioritization helps the project focus on its most severe risks by assessing the risk exposure.

ii. Exposure is the product of the probability of incurring a loss due to the risk and the potential magnitude of that loss.

iii. This prioritization can be done in a quantitative way, by estimating the probability (0.1–1.0) and relative loss, on a scale of 1 to 10.

iv. Multiplying these factors together provide an estimation of risk exposure due to each risk item, which can run from 0.1 to 10.

v. The higher the exposure, the more aggressively the risk should be tackled. It may be easier to simply estimate both probability and impact as High, Medium or Low.

vi. Those items having at least one dimension rated as high are the ones to worry about first.

2. **Risk control :** Risk control is the process of managing risks to achieve the desired outcomes. Risk control process involves the following activities :

a. Risk planning :

i. Risk planning is to identify strategies to deal with risk. These strategies fall into three categories :

1. Risk avoidance
2. Risk minimization
3. Risk contingency plans

b. Risk mitigation :

i. The risk mitigation is plan that would reduce or eliminate the highest risks.

ii. The mitigation plan includes a description of the actions that can be taken to mitigate the red rated risk and assigns a primary handler for the action.

c. Risk resolution :

i. Risk resolution is the execution of the plans for dealing with each risk.

ii. If the risk is at the watch list, a plan of how to resolve the risk already had taken place. The project manager has to respond to the trigger and execute the action plan.

iii. The project manager also needs to report progress against the plan and correct for deviation.

iv. The input to this phase is the lists of risk and its implementation. The output is risk action plan.

d. Risk monitoring :

i. Risk monitoring is continually reassessing of risks as the project proceeds and conditions change.

ii. For example, successful completion of beta testing means that the risk of the client organization rejecting the system is minimal, while large turnover in development staff usually increases project and product risks.

Que 5.29. What is risk management ? How are project risks different from technical risks ?

AKTU 2013-14, Marks 05

AKTU 2016-17, Marks 10

Answer

Risk management : Refer Q. 5.28, Page 5–27C, Unit-5.

Difference :

S. No.	Project risks	Technical risks
1.	Project risk is the cumulative effect of events that may prevent from achieving project and product goals.	Technical risk is that related to the discovery of new information, technology limits, and/or constraints that render previous solution assumptions invalid.
2.	These risks tend not to get much attention, but can have an equally negative effect on the project outcome if not addressed.	This is where the majority of the risk assessment is focused.

Que 5.30. What do you mean by risk management ? Explain how to select best risk reduction technique when there are many ways of reducing a risk.

AKTU 2014-15, Marks 10

Answer

Risk management : Refer Q. 5.28, Page 5–27C, Unit-5.

Selection of risk reduction technique :

1. To choose between the different strategies of reducing a risk, the project manager must consider the cost of handling the risk and the corresponding reduction of risk.
2. For this, we may compute the risk leverage of the different risks.
3. Risk leverage is the difference in risk exposure divided by the cost of reducing the risk.
4. More formally,

$$\text{Risk leverage} = \frac{\left(\begin{matrix}\text{Risk exposure}\\ \text{before reduction}\end{matrix}\right) - \left(\begin{matrix}\text{Risk exposure}\\ \text{after reduction}\end{matrix}\right)}{\text{Cost of reduction}}$$