

Q.2a. Elaborate the technical and interpersonal skills required for a system analyst.

Ans 2(a) System Analyst

For a System Development role, an individual must possess specific skills to effectively carry out the job but these do not necessarily comprise extensive computer coding ability. However, a system analyst **should be thoroughly familiar with the system processes and the business processes.** The analyst should also thoroughly familiarize himself/herself with the software systems being used for the software development process.

A System Analyst's skills can be divided into two categories:

- Interpersonal Skills
- Technical Skills.

Here are a few critical skills which are essential for any person to take up a role as System Analyst.

Interpersonal Skills	Technical Skills
<input type="checkbox"/> Communication - should possess good articulating and speaking skills, including knowledge and proficiency in the language used to develop the product. (e.g.: English.) <input type="checkbox"/> Understanding - should have a good understanding of the company objectives and goals and a good understanding of the subject of his area of work. <input type="checkbox"/> Dynamic personality - should be a dynamic personality ready to accept new challenges. Should be pro-active personality than a reactive. <input type="checkbox"/> Team Player - should be a good team player and know the team dynamics.	<input type="checkbox"/> Problem solving - should be able to solve problems, suggest alternate solutions and be able to take up challenges. <input type="checkbox"/> Domain expert - should have adequate knowledge of the system and be proficient in it. <input type="checkbox"/> Inquisitive Mind - should be knowing the what, when, why, where, who and how a system works. <input type="checkbox"/> Providing Support - should be able to provide support to the users when required.

(6 marks)

b. Give example of the type of system models that you might create during the analysis

process.

Ans 2(b) Examples of the types of system models that might be created during the analysis process are:

- *A data-flow model:* Data-flow models show how data is processed at different stages in the system.
- *A composition model:* A composition or aggregation model shows how entities in the system are composed of other entities.
- *An architectural model:* Architectural models show the principal sub-systems that make up a system.
- *A classification model:* Object class/inheritance diagrams show how entities have common characteristics.

A stimulus-response model: A stimulus-response model, or state transition diagram, shows how the system reacts to internal and external events. (4 marks)

c. Describe Key process areas of Capability Maturity Model (CMM).

Ans 2(c) Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief.



Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process. The key process areas at Level 2 focus on the project's concerns related to establishing basic project management controls. They are Requirements Management, Project Planning, Project Tracking and Oversight, Subcontract Management, Quality Assurance, and Configuration Management.

The key process areas at Level 3 address both project and organizational issues, as the organization establishes an infrastructure that institutionalizes effective engineering and management processes across all projects. They are Organization Process Focus, Organization Process Definition, Training Program, Integrated Management, Product Engineering, Intergroup Coordination, and Peer Reviews.

The key process areas at Level 4 focus on establishing a quantitative understanding of both the process and the work products being built. They are Quantitative Process Management and Quality Management.

The key process areas at Level 5 cover the issues that both the organization and the projects must address to implement continual, measurable software process improvement. They are Defect Prevention, Technology Change Management, and Process Change Management. Each key process area is described in terms of the key practices that contribute to satisfying its goals. The key practices describe the infrastructure and activities that contribute most to the effective implementation and institutionalization of the key process area.

(6 marks)

Q.3a. What do you understand by requirement elicitation? Discuss any two techniques in detail. (8)

Ans 3(a): Requirement Elicitation: It is the activity that helps to understand the problem to be solved. Requirements are gathered by asking question, writing down the answers, asking other questions, etc. Hence, a requirement gathering is the most communications intensive activity of the software development. Requirements elicitation requires the collaboration of several groups of participants who have different background. The two important techniques of requirement elicitation are explained below:

Initiating the Process The most commonly used requirements elicitation technique is to conduct a meeting or interview. The first meeting between a software engineer (the analyst) and the customer can be likened to the awkwardness of a first date between two adolescents. Neither person knows what to say or ask; both are worried that what they do say will be misinterpreted; both are thinking about where it might lead (both likely have radically different expectations here); both want to get the thing over with, but at the same time, both

want it to be a success. The analyst must start by asking *context-free questions*. That is, a set of questions that will lead to a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of the first encounter itself. The first set of context-free questions focuses on the customer, the overall goals, and the benefits. For example, the analyst might ask:

- Who is behind the request for this work?
- Who will use the solution?
- What will be the economic benefit of a successful solution?
- Is there another source for the solution that you need? These questions help to

identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development. The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice his or her perceptions about a solution:

- How would you characterize "good" output that would be generated by a successful solution?

- What problem(s) will this solution address?
- Can you show me (or describe) the environment in which the solution will be used?
- Will special performance issues or constraints affect the way the solution is approached? The final set of questions called as *meta-questions* focuses on the effectiveness of the meeting.

- Are you the right person to answer these questions? Are your answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

Facilitated Application Specification Techniques Too often, customers and software engineers have an unconscious "us and them" mind-set. Rather than working as a team to identify and refine requirements, each constituency defines its own "territory" and communicates through a series of memos, formal position papers, documents, and question and answer sessions. History has shown that this approach doesn't work very well. Misunderstandings abound, important information is omitted, and a successful working relationship is never established. It is with these problems in mind that a number of independent investigators have developed a team-oriented approach to requirements gathering that is applied during early stages of analysis and specification. Called *facilitated application specification techniques* (FAST), this approach encourages the creation of a joint team of customers and developers who work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements. FAST has been used predominantly by the information systems community, but the technique offers potential for improved communication in applications of all kinds. Many different approaches to FAST have been proposed. Each makes use of a slightly different scenario, but all apply some variation on the following basic guidelines:

- A meeting is conducted at a neutral site and attended by both software engineers and customers.
- Rules for preparation and participation are established.
- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.

- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used.
- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal. (8 marks)

b. Consider the program given below

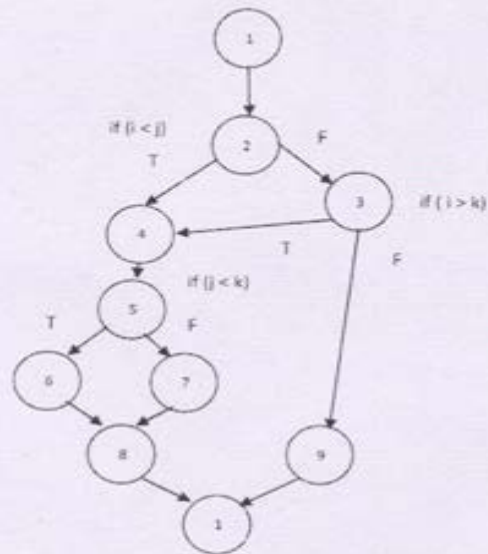
(8)

```
void main()
{
  int i,j,k;
  readln (i,j,k);
  if ((i < j) || (i > k))
  {
    writeln("then part");
    if (j < k)
      writeln (" j less then k");
    else writeln ( " j not less then k");
  }
  else writeln("else Part");
}
```

- Draw the flow graph
- Determine the cyclomatic complexity
- Arrive at all the independent paths

Ans 3(b): Ans:

```
void main()
{
  1 int i,j,k;
  2 readln (i,j,k);
  3 if( (i < j) || ( i > k) )
  {
  4 writeln("then part");
  5 if (j < k)
  6 writeln ("j less then k");
  7 else writeln ( " j not less then k");
  8 }
  9 else writeln( "else Part");
}
```



(2 marks)

(ii) Cyclomatic complexity = $E - N + 2 = 12 - 10 + 2 = 4$

(2 marks)

(iii) The four independent paths are

(4 marks)

Path1 : 1 2 3 9 10

Path2 : 1 2 4 5 7 8 10

Path3 : 1 2 4 5 6 8 10

Path4 : 1 2 3 4 5 7 8 10

Q.4a. List the benefits of prototyping. Differentiate between the objectives of evolutionary and throw-away prototyping.

Ans 4a. Benefits of prototyping:

- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified
- A working system is available early in the process
- The prototype may serve as a basis for deriving a system specification
- The system can support user training and system testing

The objective of *evolutionary prototyping* is to deliver a working system to end-users. The development starts with those requirements which are best understood.

The objective of *throw-away prototyping* is to validate or derive the system requirements.

The prototyping process starts with those requirements which are poorly understood.

(6 marks)

b. Compute function point value for a project with the following domain characteristics:

No. of I/P = 30

No. of O/P = 62

No. of user Inquiries = 24

No. of files = 8

No. of external interfaces = 2

Assume that all the complexity adjustment values are average. Assume that 14 algorithms have been counted.

Ans 4(b): We know

$UFP = \sum W_{ij} Z_{ij}$ where $j=2$ because all weighting factors are average.

$$= 30 \cdot 4 + 62 \cdot 5 + 24 \cdot 4 + 8 \cdot 10 + 2 \cdot 7$$

$$= 120 + 310 + 96 + 80 + 14$$

$$= 620$$

$$CAF = (0.65 + 0.01 \sum F_i)$$

$$= 0.65 + 0.01(14 \cdot 3)$$

$$= 0.65 + 0.42$$

$$= 1.07$$

$$nFP = UFP \cdot CAF$$

$$= 620 \cdot 1.07$$

$$= 663.4 \approx 663$$

(6 marks)

Formula - 2
Correct computation - 2
Correct answers - 2

c.Explain the general principles of user interface design.

Ans 4(c): structures the principle. Your design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with your overall user interface architecture.

1. **The simplicity principle.** Your design should make simple, common tasks simple to do, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

2. **The visibility principle.** Your design should keep all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with too many alternatives or confuse them with unneeded information.

3. **The feedback principle.** Your design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

4. **The tolerance principle.** Your design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonable.

5. **The reuse principle.** Your design should reuse internal and external components and behaviours, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember. **(4 marks)**

Q.5 a. What is meant by design patterns? What are the advantages of using design patterns? (4)

Ans 5(a): Design patterns are reusable solutions to problems that recur in many applications. A pattern serves as a guide for creating a "good" design. Patterns are based on sound common sense and the application of fundamental design principles. These are created by people who spot repeating themes across designs. The pattern solutions are typically described in terms of class and interaction diagrams. Examples of design patterns are expert pattern, creator pattern, controller pattern etc.

Design patterns are very useful in creating good software design solutions. In addition to providing the model of a good solution, design patterns include a clear specification of the problem, and also explain the circumstances in which the solution would and would not work. Thus, a design pattern has four important parts:

- The problem.
- The context in which the problem occurs.
- The solution.
- The context within which the solution works.

(4 marks)

b. Discuss the important characteristics of distributed approach to system development?

Ans 5(b): The important characteristics of distributed approach to system development are as follow:

- **Resource sharing:** A distributed system allows the sharing of hardware and software resources – such as disks, printers, files, and compilers – that are associated with computers on a network.
- **Openness:** Distributed systems are normally open systems, which mean they are designed around standard protocols that allow equipment and software from different vendors to be combined.
- **Concurrency:** In a distributed system, several processes may operate at the same time on separate computers on the network. These processes may (but need not) communicate with each other during their normal operation.
- **Scalability:** In principle at least, distributed systems are scalable in that the capabilities of the system can be increased by adding new resources to cope with new demands on the system. In practice, the network linking the individual computers in the system may limit the system scalability. If many new computers are added, then the network capacity may be inadequate.
- **Fault tolerance:** The availability of several computers and the potential for replicating information means that distributed systems can be tolerant of some hardware and software failures. In most distributed systems, a degraded service can be provided when failures occur; complete loss of service only tends to occur when there is a network failure. (6 marks)

c. What is difference between module coupling and module cohesion? List different types of coupling and cohesion.

Ans 5(c): Cohesion is the property of a single module and can be described as glue that keeps the data elements within a single module together. While defining, we must aim for high cohesion. Different types of cohesion are:

- Coincidental Cohesion
- Logical Cohesion
- Temporal Cohesion
- Communicational Cohesion
- Sequential Cohesion
- Functional Cohesion
- Procedural Cohesion

Coupling on the other hand is the measure of dependence among modules. A designer must try for minimum coupling. Different types of coupling are:

- Content Coupling
- Common Coupling
- Control Coupling
- Stamp Coupling
- Data Coupling

With some details of each..... (6 marks)

Q.6a. Discuss the benefits and problems of software reuse.

Ans6 (a): Benefits of reuse Increased dependability Reused software, that has been tried and tested in working systems, should be more dependable than new software. The initial use of the software reveals any design and implementation faults. These are then fixed, thus reducing the number of failures when the software is reused.

1. **Reduced process risk** If software exists, there is less uncertainty in the costs of reusing that software than in the costs of development. This is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub-systems are reused.
2. **Effective use of specialists** Instead of application specialists doing the same work on different projects, these specialists can develop reusable software that encapsulates their knowledge.
3. **Standards compliance** Some standards, such as user interface standards, can be implemented as a set of standard reusable components. For example, if menus in a user interfaces are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability as users are less likely to make mistakes when presented with a familiar interface.
4. **Accelerated development** Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time should be reduced.

Reuse problems

1. **Increased maintenance Costs** If the source code of a reused software system or component is not available then maintenance costs may be increased as the reused elements of the system may become increasingly incompatible with system changes.
2. **Lack of tool support** CASE toolsets may not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account.
3. **Not-invented-here Syndrome** some software engineers sometimes prefer to re-write components as they believe that they can improve on the reusable component. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.
4. **Creating and maintaining a component library** Populating a reusable component library and ensuring the software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature..

(6 marks)

b. Explain :**(i) Reverse Engineering****(ii) Re-Engineering**

Ans 6(b): Ans i) REVERSE ENGINEERING:-It is a process of analyzing software with a view to understanding its design and specification.

- In this, source code and executable code are the input.
- It may be part of a re-engineering process but may also be used to respecify a system for re-implementation.
- Builds a program data base and generates information from this.
- Program understanding tools (browsers, cross reference generates, etc.) may be used in this process.
- Design and specification may be reverse re-engineer to:-
 - a) Serve as input to SRS for program replacement.
 - b) Be available to help program maintenance.

Reverse Engineering often precedes Re-Engineering but is sometimes worthwhile in its own right. The design and specification of a system may be reverse engineered so that they can be an input to the requirements specification process for the system replacement. The design and specification may be reverse engineered to support program maintenance. (5 marks)

ii) RE-ENGINEERING:- It is re-organizing and modifying existing system to make them more maintainable. It involves:-

- Source code translation.
- Reverse engineering.
- Program structure development.
- Program modularization.
- Data re-engineering.

Restructuring or re-writing part or all of the legacy system without changing its functionality.

Legacy system is a system that is hard to maintain. So it involves:-

- 1) Re-documenting the system.
- 2) Organizing and re-structuring the system.
- 3) Modifying and upgrading structure and value of the system data.
- 4) Input to a re-engineering process is a legacy system and output is a structure modularized version of the same program. So re-engineering involves adding effort to make them easier to maintain. The system may be restructured or redocumented.

When to Re-Engineer?

- When the system changes are mostly confined to part of the system then re-engineer that part.
- When hardware or software support becomes obsolete.
- When tools to support re-structuring are available.

Advantages of Re-Engineering:-

- 1) Reduced risk – there is a high risk in new software development.

There may be development problems, staffing problems and specification problems.

- 2) Reduced cost – the cost of re-engineering is often significantly less than the cost of developing new software.

Re-Engineering cost factors:-

- 1) The quality of the software to be re-engineered.
- 2) The tool support available for re-engineering.
- 3) The extent of the data conversion, which is required.
- 4) The availability of expert staff for re-engineering.

(5 marks)

Q.7 a. What is ripple effect? How does it affect the stability of a program?

Ans 7(a) The **ripple effect** is a term used to describe a situation where, like the ever expanding ripples across water when an object is dropped into it, an effect from an initial state can be followed outwards incrementally. Examples can be found in economics where an individual's reduction in spending reduces the incomes of others and their ability to spend. In sociology it can be observed how social interactions can affect situations not directly related to the initial interaction. and in charitable activities where information can be disseminated and passed from community to community to broaden its impact.

In software, the effect of a modification may not be local to the modification, but may also affect other portions of the program. There is a ripple effect from the location of the modification to the other parts of the programs that are affected by the modification. One aspect of the ripple effect concerns the performance of the program. The primary attribute affecting the ripple effect as a consequence of a program modification is the stability of the program. Program stability is defined as the resistance to the amplification of changes in the program.

(4 marks)

b.Explain fault-tolerant architecture with suitable diagram.

Ans Page 506 of Text Book. Unit 6

c.Write a brief note on the following estimation techniques:

- (i) **Algorithmic cost modelling**
- (ii) **Expert judgement**
- (iii) **Estimation by analogy**

Ans7(c) Algorithmic: A formulaic approach based on historical cost information and which is generally based on the size of the software. (2 marks)

Expert Judgement: One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached. (2 marks)

Estimation by analogy: The cost of a project is computed by comparing the project to a similar project in the same application domain. (2 marks)

Q.8 a. Explain various types of debugging techniques used in Software testing.

Ans 8(a): Ans. Debugging is the activity of locating and correcting errors. Various debugging techniques are:-

1) Core dumps:-A printout of all registers and relevant memory location is obtained and studies. All dumps should be well documented and retained for possible use on subsequent problems.

Advantages:-

1. The complete contents of a memory at a crucial instant of time are obtained for study.
2. Can be cost effective if used to explore validity of a well formulated error hypothesis.

Disadvantages:-

1. Require some CPU time, significant input time, and much analysis time.
2. Wasteful if used indiscriminately.
3. Hexadecimal numbers are cumbersome to interpret and it is difficult to determine the address of source language variables.

2) Traces:-Printout contains only certain memory and register contents and printing is conditional on some event occurring. Typical conditionings are entry, exit, or use of-

- 1) A particular subroutine, statement, macro, or database;
- 2) Communication with a terminal, printer, disk, or other peripheral;
- 3) The value of a variable or expression; and
- 4) Timed actuations in certain real time system.

A special problem with trace programs is that the conditions are entered in the source language and any changes require a recompilation.

3) Print statements:-The standard print statement in the language being used is sprinkled throughout the program to output values of key variables.

Advantages:-

- 1) This is a simple way to test whether a particular variable changes, as it should after a particular event.

2) A sequence of print statements portrays the dynamics of variable changes.

Disadvantages:-

- 1) They are cumbersome to use on large programs.
- 2) If used indiscriminately they can produce copious data to be analyzed much of which is superfluous.
- 4) Debugging programs:-A program which runs concurrently with the program under test and provides commands to examine memory and registers, stop execution of a program at a particular point, search for references to particular constants, variables, registers.

Advantages:-

- 1) Terminal oriented real time program.
- 2) Considerable flexibility to examine dynamics of operation.

Disadvantages:-

- 1) Generally works on a machine language program.
- 2) Higher-level language versions must work with interpreters.
- 3) More commonly used on microcomputers than large computers.

(6 marks)

b. What are the advantages of using testing tools? Explain in detail different type of testing tools.

Ans 8(b): Ans: The advantages of testing tools are:

- They improve the productivity and quality of software development.
- Help in identification of errors which are difficult and time consuming to find manually.
- Reduce the testing time.
- Help in running large volumes of test unattended for 24 hours.
- Automatic generation of test cases.
- Automated the regression testing.
- Improve the productivity of the testers.

Some of the important testing tools are:

- (a) Test Case Generators: These tools generate test cases from SRS, program or test design languages. They use certain rules called test design techniques to generate test cases.
- (b) Capture/ Playback and Test harness tools: These tools automate the re running of manual test by recording and replaying the test scripts. The recorded test scripts can also be edited as per need. They can be either intrusive or non intrusive type. Intrusive tools along with software under test reside on the same machine.
- (c) Coverage Analysis Tools: These tools ensure that the software is tested and helps the tester to find the parts which are not covered.
- (d) Test Comparators: These tools compare the results of software under test with the expected results and generate the report.
- (e) Memory Testing Tools: These tools are used to test memory related problems, such as using uninitialized memory location, accessing memory locations which are out of range etc.
- (f) Simulators: These tools are used to simulate the hardware/ software with which software under test is going to interact.
- (g) Test database: It is a sample of database which is being manipulated by software under test.

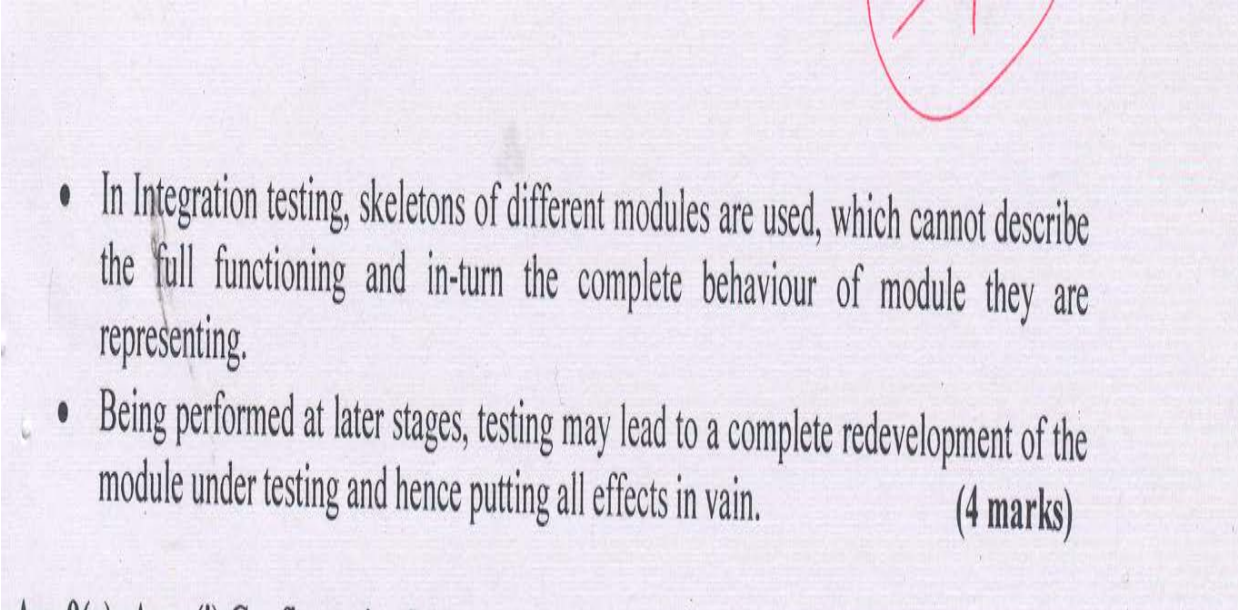
(6 marks)

c. Explain some of the limitations of testing.

Ans 8(c): Ans: Though testing is an important part of system development and leads to a valid, verified and efficient system, it also faces some limitation in its scope.

Following are some of such limitations.

- It is very difficult to trace out logical errors through Testing.
- Stress testing or load tests are not the realistic options and hence, it cannot be defined that how application or module will be reacting at heavy data loads.

- 
- In Integration testing, skeletons of different modules are used, which cannot describe the full functioning and in-turn the complete behaviour of module they are representing.
 - Being performed at later stages, testing may lead to a complete redevelopment of the module under testing and hence putting all effects in vain. (4 marks)

Q.9 a. Write short notes on:

2)

- (i) Configuration Management**
- (ii) Decision Table**

Ans 9(a): Ans: (i) Configuration Management:

Due to several reasons, software changes during its life cycle. As a result of the changes made, multiple versions of the software exist at one time. These changes must be managed, controlled and documented properly in order to have reliable systems.

Configuration management helps the developers to manage these changes systematically by applying procedures and standards and by using automated tools.

Though multiple versions of the software exist in the tool repository, only one official version of set of project components exist called baseline.

The different components of the baseline are called configuration items. Some examples of configuration items are project plan, SRS, Design document, test plans, user manuals etc. A group of people constitute Configuration Control Board (CCB) which controls the changes to be made to the software. Whenever changes are to be made the following steps are followed:

- (i) Submit the change request along with details to CCB
- (ii) CCB accesses the change request after proper evaluation.
- (iii) Depending upon the results, the request is either accepted or rejected or can be deferred for the future assessment.
- (iv) If accepted, proper plan is prepared to implement the change.
- (v) Once changes are made, after validating by the quality personnel, all configuration items are updated.

Some popular configuration management tools are Clear CASE, Visual Source Safe etc.

(4 marks)

(ii) Decision Table: When the process logic for a process involves multiple conditions and is very complicated, it is not advisable to use structured English.

Instead decision tables are used. Main parts of the table are:

1. Condition Stubs
2. Action Stubs
3. Roles

Condition stubs list all the conditions relevant to the decision. Action part lists all the actions that will take place for a valid set of conditions. Finally rules parts of the table specify the set of conditions that will trigger a particular action. A sample decision table is shown below:

	Rule 1	Rule 2	Rule 3
Condition 1	X	X	
Condition 2		X	
Condition 3	X		X
Condition 4			X
Action 1	X		
Action 2		X	
Action 3			X

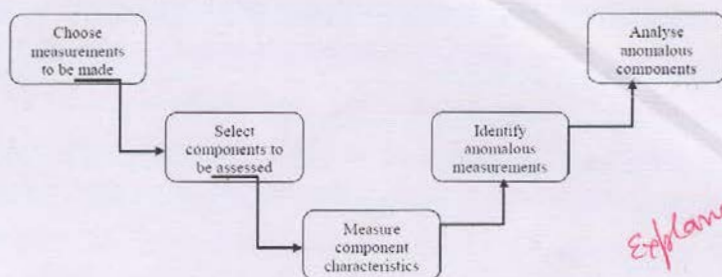
From this table it is clear that if conditions 1 and 3 are satisfied, Action 1 will be triggered.

(4 marks)

b. With the help of a figure, explain the key stages of software measurement process which is a part of a quality control process.

Ans 9(b): A software measurement process that may be part of a quality process is shown in figure below. Each of the components of the system is analysed separately, and the values of the metric compared both with each other and, perhaps, with historical measurement data collected on previous projects. Anomalous measurements should be used to focus the quality assurance effort on components that may have quality problems. The key stages in this process are:

- **Choose measurements to be made:** The questions that the measurement is intended to answer should be formulated and the measurements required to answer these questions defined. Measurements that are not directly relevant to these questions need not be collected. Basili's GQM (Goal-Question-Metric) paradigm is a good approach to use when deciding what data is to be collected.
- **Select components to be assessed:** It may not be necessary or desirable to assess metric values for all of the components in a software system. In some cases, you can select a representative selection of components for measurement. In others, components that are particularly critical, such as core components that are in almost constant use, should be assessed.



The process of product measurement

Explanation of key stages - 5
Diagram - 3

- **Measure component characteristics:** The selected components are measured and the associated metric values computed. This normally involves processing the component representation (design, code, etc.) using an automated data collection tool. This tool may be specially written or may already be incorporated in CASE tools that are used in an organisation.
- **Identify anomalous measurements:** Once the component measurements have been made, you should compare them to each other and to previous measurements that have been recorded in a measurement database. You should look for unusually high or low values for each metric, as these suggest that there could be problems with the component exhibiting these values.
- **Analyse anomalous components:** Once components that have anomalous values for particular metrics have been identified, you should examine these components to decide whether the anomalous metric values mean that the quality of the component is compromised. An anomalous metric value for complexity does not necessarily mean a poor quality component. There may be some other reason for the high value and it may not mean that there are component quality problems.

(8 marks)

Text Book

1. Software Engineering, Ian Sommerville, 7th edition, Pearson Education, 2004 (TB-I)