BCA(III) – 17/20

2020

Time: 3 hours

Full Marks: 80

Candidates are required to give their answers in

their own words as far as practicable.

The figures in the margin indicate full marks.

Answer from both the Sections as directed.

Section – A

1. Choose the correct answer of the following:

2 x 10 = 20

(a) The first step in Software Development Life Cycle (SDLC) is:
- (i) Preliminary Investigation and Analysis
- (ii) System Design
- (iii) System Testing
- (iv) Coding

(b) The detailed study of existing system is referred to as:
- (i) System Planning
- (ii) System Analysis
- (iii) Feasibility Study
- (iv) Design DFD

(c) Risk analysis of a project is done in:
- (i) System Analysis
- (ii) Feasibility Study
- (iii) Testing Phase
- (iv) Maintenance Phase

(d) Which one of the following is the most important phase of SDLC?
- (i) Requirement Analysis
- (ii) Design
- (iii) Testing
- (iv) Coding

(e) Which step of SDLC performs cost/benefit analysis?
- (i) Feasibility Study
- (ii) Analysis
- (iii) Design
- (iv) None of these

(f) Alpha and Beta testing are the forms of:
- (i) Acceptance testing
- (ii) Integration testing
- (iii) Unit testing
- (iv) System testing

(g) Spiral Model was developed by:
   (i) [✓ Barry Bohem]   (ii) Roger Pressman
   (iii) Victor Bisli   (iv) None of them

(h) The worst type of coupling is:
   (i) Data coupling
   (ii) Control coupling
   (iii) Stamp coupling
   (iv) [✓ Content coupling]

(i) Which one of the following is performed by user?
   (i) [✓ Acceptance testing]
   (ii) Unit testing
   (iii) Compatibility testing
   (iv) None of these

(j) Who writes Software Requirement Specification Document (RSD)?
   (i) [✓ System Developer]
   (ii) System Tester
   (iii) System Analyst
   (iv) None of these

Section- B

Answer any **four** questions of the following:

15 x 4 = 60

2. Explain the term 'SDLC'. What are the various phases of SDLC? Explain in detail.

Ans: SDLC is the acronym of the Software Development Life Cycle.

It is also called the Software Development Process.SDLC is a framework defining tasks performed at each step in the software development process. SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace, and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process. Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop, and test high-quality software. The SDLC aims to produce high-quality software that meets or exceeds customer expectations, reaches completion within time and cost estimates.

SDLC Phases

#1) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectations. Any ambiguities must be resolved in this phase only.

The business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

#2) Design

In this phase, the requirements gathered in the SRS document are used as an input, and software architecture that is used for implementing system development is derived.

#3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

#4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed. Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer to SRS documents to make sure that the software is as per the customer's standard.

#5) Deployment

Once the product is tested, it is deployed in the production environment, or the first UAT (User Acceptance testing) is done depending on the customer's expectation. In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

#6) Maintenance

After the deployment of a product in the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care of by the developers.

3. What do you understand by Requirement Engineering? What are its various phases? Explain in detail.

Ans: **Requirement Engineering** is the process of defining, documenting, and maintaining the requirements. It is a process of gathering and defining the service provided by the system. Requirement engineering is also known as requirement analysis. The goal of requirements engineering is to develop and maintain a sophisticated and descriptive 'System Requirements Specification' document.

<u>The various phases of requirement engineering:</u>

1.Requirements Elicitation:

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of the same type, standards, and other stakeholders of the project.

The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Elicitation does not produce formal models of the requirements understood. Instead, it widens the knowledge domain of the analyst and thus helps in providing input to the next stage.

2.Requirements specification:

This activity is used to produce formal software requirements models. All the requirements including the functional as well as the non-functional requirements and constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

3.Requirements verification and validation:

Verification: It refers to the set of tasks that ensure that software correctly implements a specific function.

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

If requirements are not validated, errors in the requirements definitions would propagate to the successive stages resulting in a lot of modification and rework.

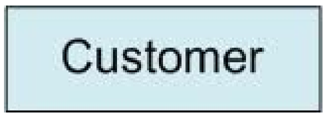
4.Requirements management:

Requirement management is the process of analyzing, documenting, tracking, prioritizing, and agreeing on requirements and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end-users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

4. What do you understand by Data Flow Diagram (DFD)? What are the Symbols used in it and what are the rules for constructing DFD?
Ans: **DFD** is the abbreviation for **Data Flow Diagram**. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart. Data Flow Diagrams can be represented in several ways. The DFD belongs to structured-analysis modeling tools. Data Flow diagrams are very popular because they help us to visualize the major steps and data involved in software-system processes.

There are four basic symbols to represent a data flow diagram.
1. External entity

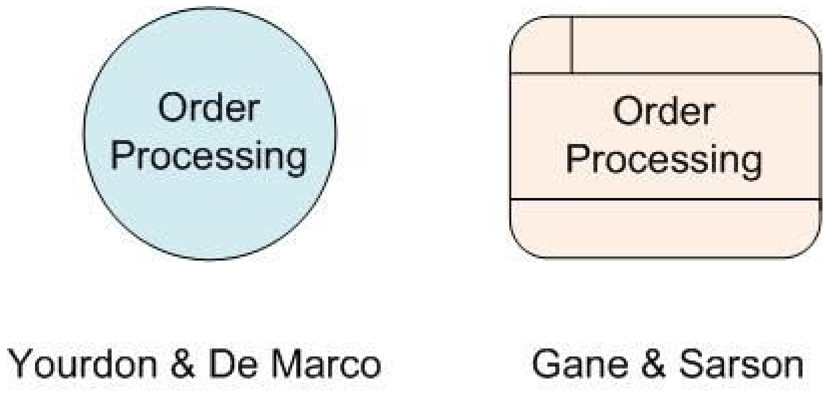
External entities are objects outside the system with which the system communicates. These are sources and destinations of the system inputs and outputs. They are also known as terminators, sinks, sources, or actors.



2. Process

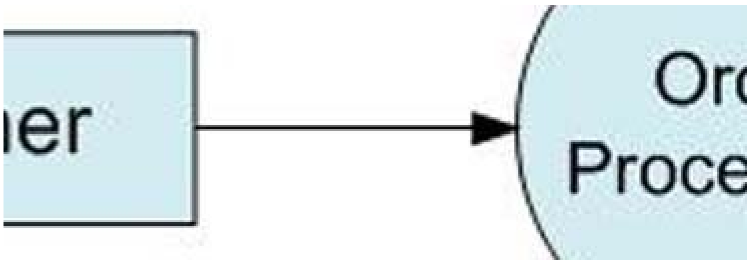
A process receives input data and process output data with a different form or content. Every process has a name that identifies the function it performs. The process can be as simple as collecting input data and saving it in the database or it may be as complex as producing monthly sales reports of any particular product in any selected region.

The symbol of the process is a circle or rectangle with rounded corners.



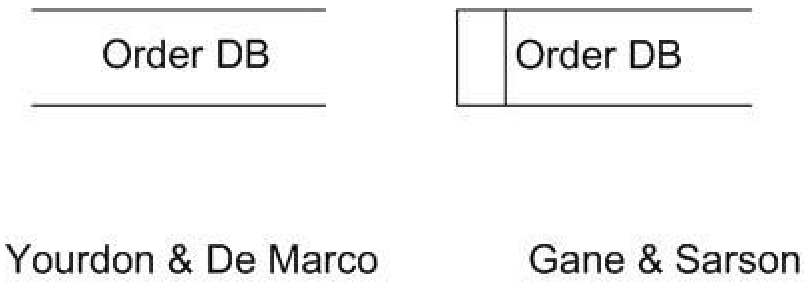
3. Data flow

Data flow is the path for data to move from one part of the system to another. It may be a single data element or set of the data elements. The symbol of data flow is the arrow. The arrow shows the flow direction.



4. Datastore

Datastore are repositories of data in the system. They are sometimes also referred to as files. Each data store receives a simple label such as Orders.



Rules for constructing DFD

- The name of the entity should be easy and understandable without any extra assistance(like comments).
- The processes should be numbered or put in an ordered list to be referred to easily.
- The DFD should maintain consistency across all the DFD levels.
- A single DFD can have maximum processes up to 9 and a minimum of 3 processes.

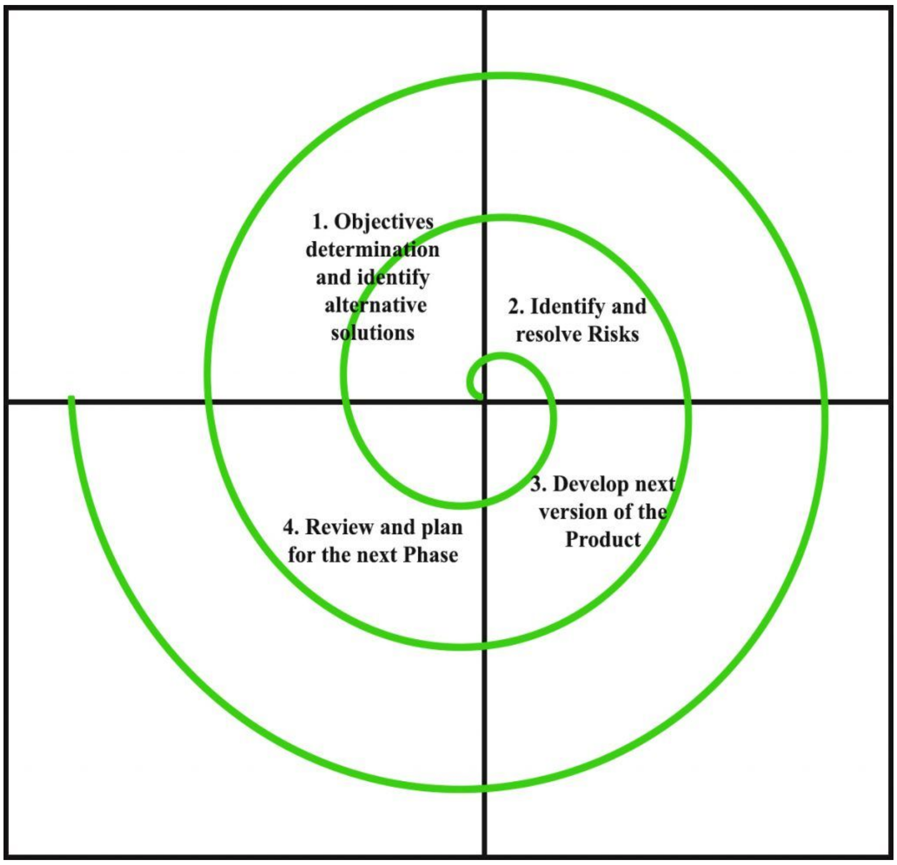
5. What do you understand by Modularity? What are the advantages of a Modular System? Discuss briefly.
Ans: The module simply means the software components that have been created by dividing the software. The software is divided into various components that work together to form a single functioning item but sometimes they can perform as a complete function if not connected with each other. This process of creating software modules is known as **Modularity** in software engineering.
It simply measures the degree to which these components are made up than can be combined. Some of the projects or software designs are very complex that it’s not easy to understand their working and functioning. In such cases, modularity is a key weapon that helps in reducing the complexity of such software or projects.

Advantages of a Modular System:-

- Allow each module to be written separately without having to know the code of other modules.

- Faster development time due to parts being split up across team members and problems handled in chunks.
- Modules allow for greater flexibility of the overall program.
- Individual modules are highly reusable, this Cuts down costs and reduces the time taken on future projects.
- Bugs are less frequent and easier to detect as they will exist in the module they originate from Shorter programs.

6. Draw and explain in various phases of Spiral Model. What are its advantages?
Ans:


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

Four phases of the spiral model:-

1. **Objective determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.
3. **Develop the next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Advantages of Spiral Model:

1. Software is produced early in the software life cycle.
2. Risk handling is one of the important advantages of the Spiral model, it is the best development model to follow due to the risk analysis and risk handling at every phase.
3. Flexibility in requirements. In this model, we can easily change requirements at later phases and can be incorporated accurately. Also, additional functionality can be added at a later date.
4. It is good for large and complex projects.
5. It is good for customer satisfaction. We can involve customers in the development of products in the early phase of software development. Also, software is produced early in the software life cycle.
6. Strong approval and documentation control.
7. It is suitable for high-risk projects, where business needs may be unstable. A highly customized product can be developed using this.

7. Explain the difference between Functional and Structural Testing.
Ans:

Functional Testing	**Structural Testing**
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure of the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred to as outer or external software testing.	It is the inner or internal software testing.
It is a functional test of the software.	It is a structural test of the software.
This testing can be initiated on the basis of the requirement specifications document.	This type of testing of software is started after a detailed design document.

No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called clear box testing.
It is the least time-consuming.	It is the most time-consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	**Example:** by input to check and verify loops

8. Discuss Cohesion and Coupling. Explain their various types. Explain what happens when two modules have high coupling.
Ans: **Cohesion** is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. Good software design will have high cohesion.
Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for another element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update records in the database and send them to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.

- **Temporal Cohesion:** The elements are related by their timing involved. In a module connected with temporal cohesion, all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at the same time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related. The elements have no conceptual relationship other than the location in the source code. It is accidental and the worst form of cohesion. Ex- print the next line and reverse the characters of a string in a single component.

Coupling is a measure of the degree of interdependence between the modules. Good software will have low coupling.

Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data access, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

If two modules are concerned with the high coupling, it means their interdependence will be very high. Any changes applied to the single module will affect the functionality of the other module. The greater the degree of change, the greater will be its effect on the other. As the dependence is higher, such modifications will affect modules in a negative manner, and in turn, the maintainability of the project is decreased. This will further decrease the reusability factor of individual modules and lead to unsophisticated software. So, it is always desirable to have inter-connection & interdependence among modules.