AUTUMN END SEMESTER EXAMINATION-2019

School of Computer Engineering

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
DEEMED TO BE UNIVERSITY, BHUBANESWAR-24

Software Engineering
[IT-3003]

Q. 1

(a) Differentiate between exploratory style and modern style of software development.

Ans: The difference between exploratory style and modern style of software development is as below:

1. Emphasis has shifted from error correction to error prevention. Modern practices emphasize on detection of errors as close to their point of introduction as possible. In exploratory style, errors are detected only during testing

2. In exploratory style, coding is synonymous with program development. Now, coding is considered only a small part of program development effort.

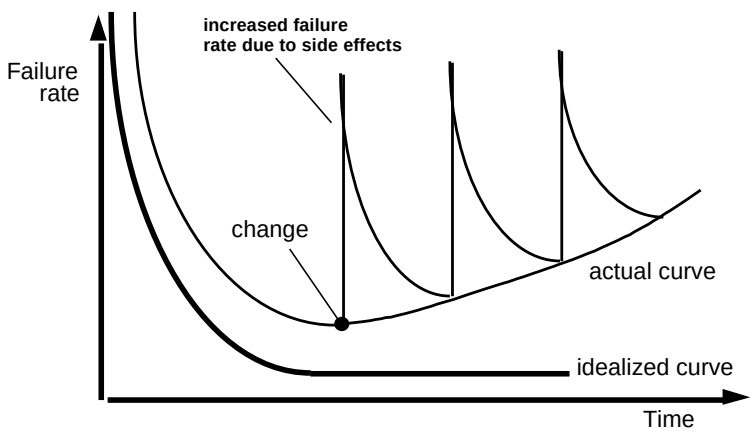
3. Exploratory style does not provide good visibility of design and code, whereas modern practices gives more visibility to this aspect.

4. Exploratory style lacks in documentation as it is mostly based on individual heroics and intuition. On the other hand, modern practices provides good documentation.

(b) What do you understand by the phrase "software doesn't wear out".

Ans: The phrase "Software doesn't wear out" refers to the ability of the software product to work as expected without getting affected by the environmental factors. Initially, some undetected errors may cause failures but gradually the failure rate gets flattened with the rectification of initial undetected errors. From above, it is clear that software does not wear out. However, it does deteriorate due to the maintenance activities carried out on it in its life-cycle. The ideal failure rate curve of a software is shown as below:

Wear vs. Deterioration



The software also changed because of many modifications in design or coding during maintenance phase. After some years, the software becomes so complex and dirty, that instead of modification, a new software is built.

(c) List the properties of a good SRS.

Ans: The properties of good SRS is as below:

1. It should be concise and unambiguous, easy to change,well-structured. consistent, complete. Traceable, verifiable
2. It should only specify what the system must do and not how it will do.

(d) Explain the three types of testing conducted during the system testing.

Ans: System testing consists of following activities:

- **Alpha Testing:** It is performed by the developers at their location.
- **Beta Testing:** It is performed by a friendly set of users, known as dummy users, at the developers site only. The dummies are not the actual users; rather they behave like actual users. Dummies are not the part of the development team but they belong to the development company. These dummies use use the product for a specified time frame after which they submit their feedback. If the feedback is satisfactory then only software release docs are prepared and meeting with the client is fixed.
- **Acceptance Testing:** This is done by the real users i.e. Clients. They will test the product using their SRS and will either ACCEPT or REJECT the product, based on their test results.

(e) What do you mean by software re-usability?

Ans: In computer science and software engineering, re-usability is the use of existing assets in some form within the software product development process; these assets are products and by-products of the software development life cycle and include code, software components, test suites, designs and documentation.

(f) "Development time is sublinear function of product size": Based on the statement identify the type of COCOMO model and justify the statement.

Ans: The above statement is applicable in basic COCOMO. This is because the development time increases but moderately, with problem size. This is because parallel activities can be carried out simultaneously by the engineers. It reduces the time to complete the project.

(g) State different UI (User Interface) design styles.

Ans: There are different ways of interacting with computer systems which have evolved over the years. The widely used types of user interface are as below:

1. Command Line
2. Graphical User Interface (GUI)
3. Menu driven
4. Form based

(h) What do you understand by risk leverage?. How it is calculated?

Ans: Risk Reduction Leverage (RRL) is defined as the difference between the Risk Exposure before and after the reduction activity divided by the cost of that activity. It can be calculated as below:

risk leverage = (risk exposure before reduction – risk exposure after reduction) / (cost of reduction.

(i) Differentiate between verification and validation in software testing.

Ans:

Verification	Validation
1. **Verification** is a static practice of verifying documents, design, code and program.	1. **Validation** is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is the process of checking “Are we developing right way”	3.It is the process of checking “Did we develop right thing”
4. **Verification** uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. **Validation** uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.	6. **Validation** is carried out with the involvement of testing team.

(j) State the use of SaaS (Software as a Service) in software development.

Ans:

1. SaaS is a software delivery model and involves customers to pay for any software per unit time of usage, with the price reflecting the market place supply and demand.

2. SaaS shifts the “ownership” of the software from the customer to a service provider. Software owner provides maintenance, daily technical operations, and support for the software.

3. Services are provided to the client on amount of usage basis.

4. The cost of providing software services are reduced as more and more customers subscribe to the service.

Ans 2 (A)

2	(a)	Explain the concept of Evolutionary Model? Give an example of a software application which you will develop using the evolutionary model and why?
	(b)	Explain the functional, non functional and constraints of a SRS document along with an example.

Ans:

2.a Evolution model explanation - 2

Diagram - 1

Application and why -1

Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. Evolutionary model (aka successive versions or incremental model):

The system is broken down into several modules which can be incrementally implemented and delivered.

-First develop the core modules of the system.

The initial product skeleton is refined into increasing levels of capability

by adding new functionalities in successive versions.

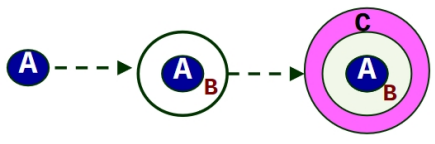
- Successive version of the products are

functioning systems capable of performing some useful work.

- A new release may include new functionality:

also existing functionality in the current release might have been enhanced.

The number of chunks is huge and is the number of deliveries made to the customer. The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

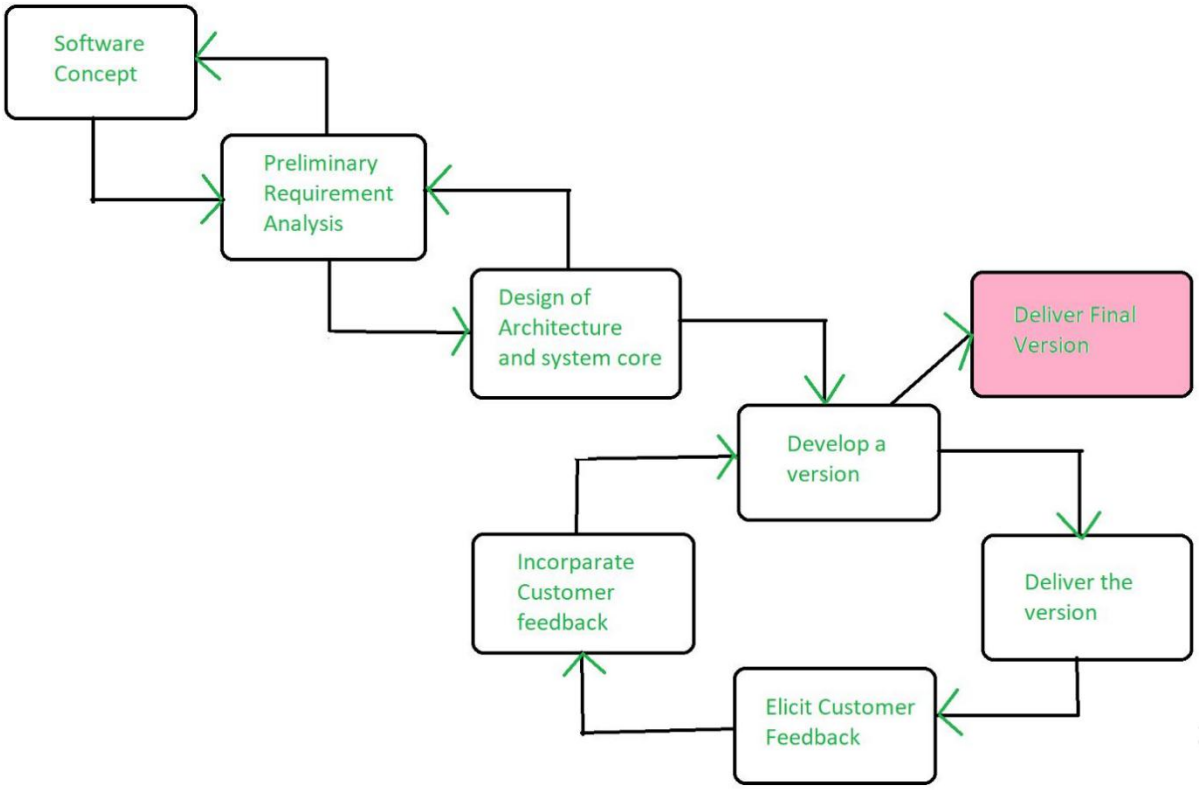


Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.

It is better for software products that have their feature sets redefined during development because of user feedback and other factors. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.

All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.



Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.

4. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- Helps finding exact user requirements: much before fully working system is developed.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

Ans 2 (B) In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data

manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional software requirements help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform.

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

Example of Functional Requirements

- The software automatically validates customers against the ABC Contact Management System
- The Sales system should allow users to record customers sales
- The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
- Only Managerial level employees have the right to view revenue data.
- The software system should be integrated with banking API
- The software system should pass Section 508 accessibility requirement.

Examples of Non-functional requirements

Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audit

In user management page in library software

SRS-001: The system shall display the user account information including

user ID, last and first name, and user position, privilege.

SRS-002: The system shall use a graphic user interface which allows

librarians to choice actions including removing, changing and adding user

account and account information..

3.(a) Compute the function point value for a project with the following domain characteristics:

1. No. of user inputs = 24
2. No. of user outputs = 65
3. No. of user inquiries = 12
4. No. of files = 12
5. No. Of external interfaces = 4

Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

Ans.

Masurement Parameter	Count		Weighing Parameter (Average)
No. of user inputs	24	*	4 = 96
No. of user outputs	65	*	5 = 325
No. of user inquiries	12	*	4 = 48
No. of files	12	*	10 = 120
No. of external interfaces	4	*	7 = 28
Count Total			**617**

So sum of all Fi (1 to 14) = 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = **43**

FP = Count-total * [0.65 + 0.01 * ∑(Fi)]

= 617 * [0.65 + 0.01 * 43]

= 617 * 1.08

= 666

3.(b)	Describe the role of COCOMO Model in software engineering. Explain the three categories of software product development.

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

- **For each of the three product categories:**
  - From size estimation (in KLOC), Boehm provides equations to predict:
    - **project duration in months**
    - **effort in programmer-months**
- Gives only an approximate estimation:
  - **Effort = a1 (KLOC)^{a2}**
  - **Tdev = b1 (Effort)^{b2}**
    - KLOC is the estimated kilo lines of source code,
    - a1,a2,b1,b2 are constants for different categories of software products,
    - Tdev is the estimated time to develop the software in months,
    - Effort estimation is obtained in terms of person months (PMs).

Divides software product developments into 3 categories:

Organic: Relatively small groups. working to develop well-understood applications.

Semidetached: Project team consists of a mixture of experienced and inexperienced staff.

Embedded: The software is strongly coupled to complex hardware, or real-time systems.

Development Effort Estimation

- Organic : **Effort** = 2.4 $(KLOC)^{1.05}$ PM
- Semi-detached: **Effort** = 3.0$(KLOC)^{1.12}$ PM
- Embedded: **Effort** = 3.6 $(KLOC)^{1.20}$PM

Development Time Estimation

- Organic: **Tdev** = 2.5 $(Effort)^{0.38}$ Months
- Semi-detached: **Tdev** = 2.5 $(Effort)^{0.35}$ Months

Embedded: **Tdev** = 2.5 $(Effort)^{0.32}$ Months

4. **(a)** Using the table below, draw the network diagram and answer the following questions:

Activity	Predecessor	Estimate in weeks
Start	-	0
C	Start	6
B	Start	4
P	Start	3
A	C, B, P	7
U	P	4
T	A	2
R	A	3
N	U	6
End	T, R, N	0

1. How many paths are in the network, and what are they?
2. What is the critical path and its duration?
3. What is the float on activity U?
4. What is the impact to the project if activity B takes three weeks longer than planned?
5. Do the forward and backward pass using original estimates.

1. How many paths are in the network, and what are they?

There are seven paths. See the details on the next page.

2. What is the critical path and its duration?

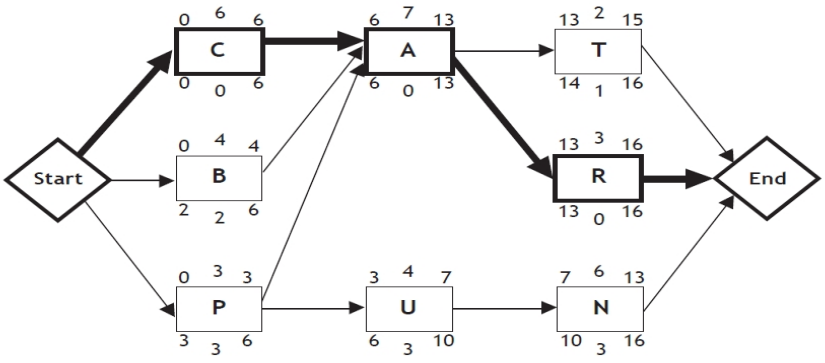
Start, C, A, R, End 16 weeks

3. What is the float on activity U?

3 weeks

4. What is the impact to the project if activity B takes three weeks longer than planned?

The critical path will shift to Start, B, A, R, End. It will delay the project end by one week.



Paths	
Start, C, A, T, End	15
Start, C, A, R, End	16
Start, B, A, T, End	13
Start, B, A, R, End	14
Start, P, A, T, End	12
Start, P, A, R, End	13
Start, P, U, N, End	13

4 (b) The activities of software design is as follows:

1) Preliminary (or high-level) design – A problem is decomposed into a set of modules, Control relationships among the modules are identified, and also interfaces are identified, The outcome of high-level design is called the program structure or the software architecture

2) Detailed design: Once the high-level design is complete, detailed design is undertaken, during design phase each module is examined carefully to design its data structures and the algorithms, the outcome is usually documented in the form of a MSPEC (module specification document) 2marks

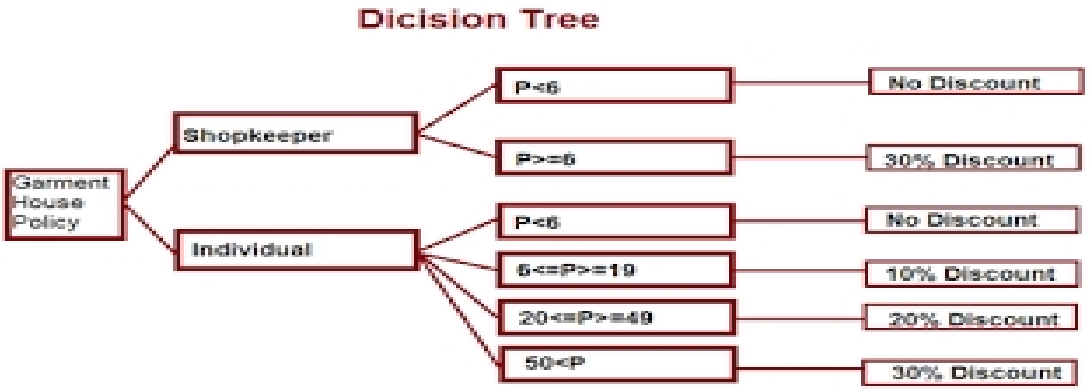
Four good characteristics of software Design are –

1. Correctness, Efficiency, Understandability (A good design should be modular and layered), Maintainability

[EXPLANATION OF ALL THE CHARACTERISTICS IS IMPORTANT] 2marks

5.	(a)	A Garment House announces its trade discount policy as follows: If a customer is from shop and does purchase garments more that 6 pair a flat discount of 30% would be provided. If a customer is individual and does purchase garment of 6-19 pair 10% discount would be provided, 20% on discount for 20-49 pair and 30% discount on more than

		equal to 50 pair. In no other case, a discount would be provided. Draw a decision tree and table for above policies.



2marks

Decision Table

Condition	R-1	R-2	R-3	R-4	Else
Shopkeeper	Y	N	N	N	
p>=6	Y				
6<=p>=19		Y			
20<=p<=49			Y		
p>=50				Y	
ACTION:					
30%	X			X	
10%		X			
20%			X		
No Discount					X

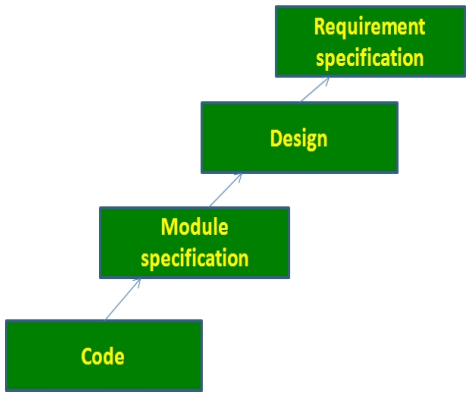
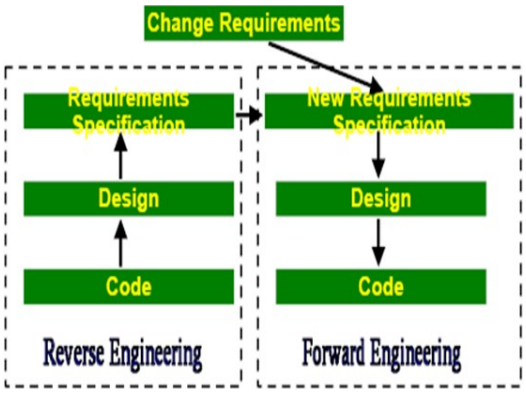
2marks

5 (b)

Cohesion is a measure of the functional strength of a module, whereas the coupling between two modules is a measure of the degree of interaction between the two modules. 1mark

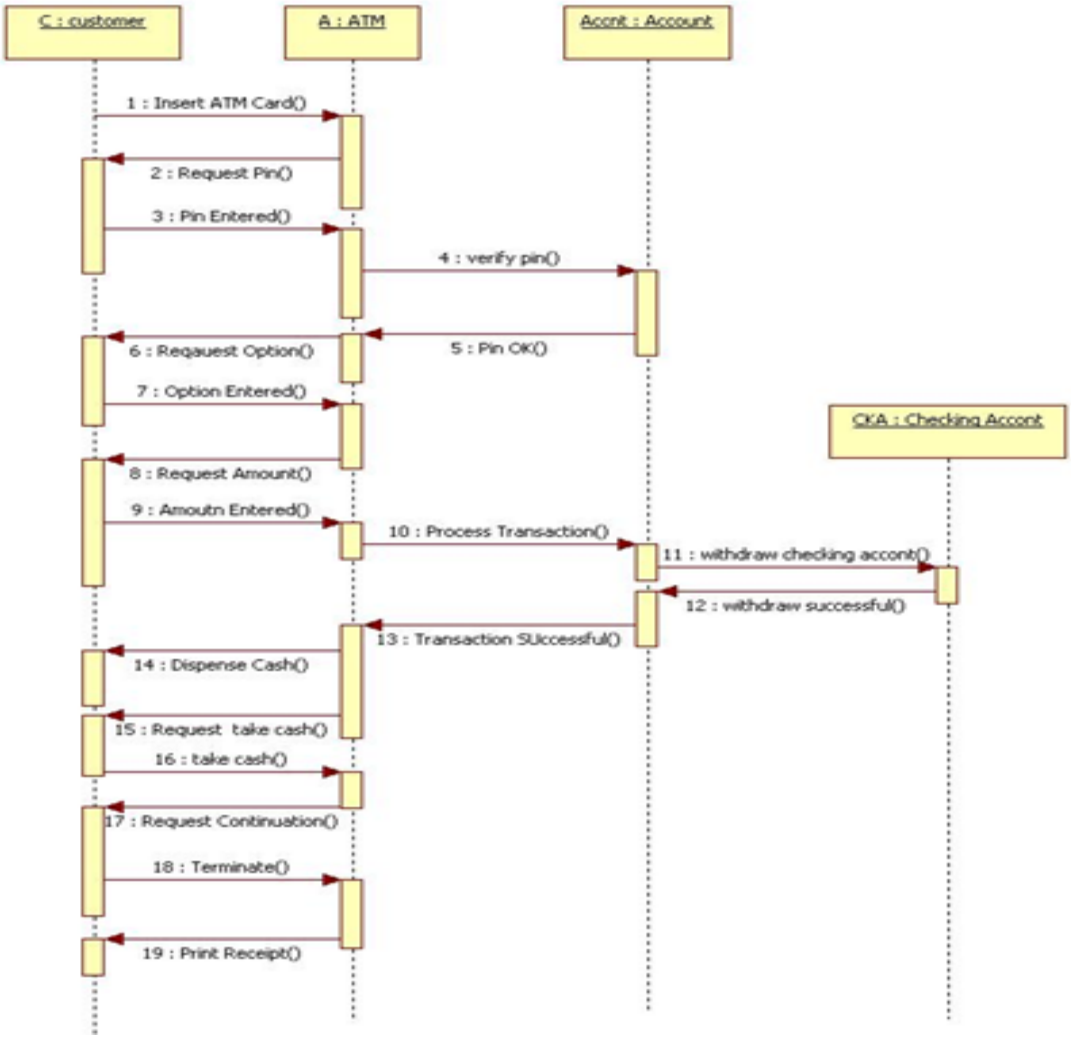
Different levels of Cohesiveness is as follows: 1) Coincidental cohesion, Logical cohesion, Temporal cohesion, Procedural cohesion, Communicational cohesion, Sequential cohesion, Functional cohesion [coincidental cohesion is the worst type of cohesion][BRIEF EXPLANATION OF ALL CLASSES] 3marks

6(a) [2+2]

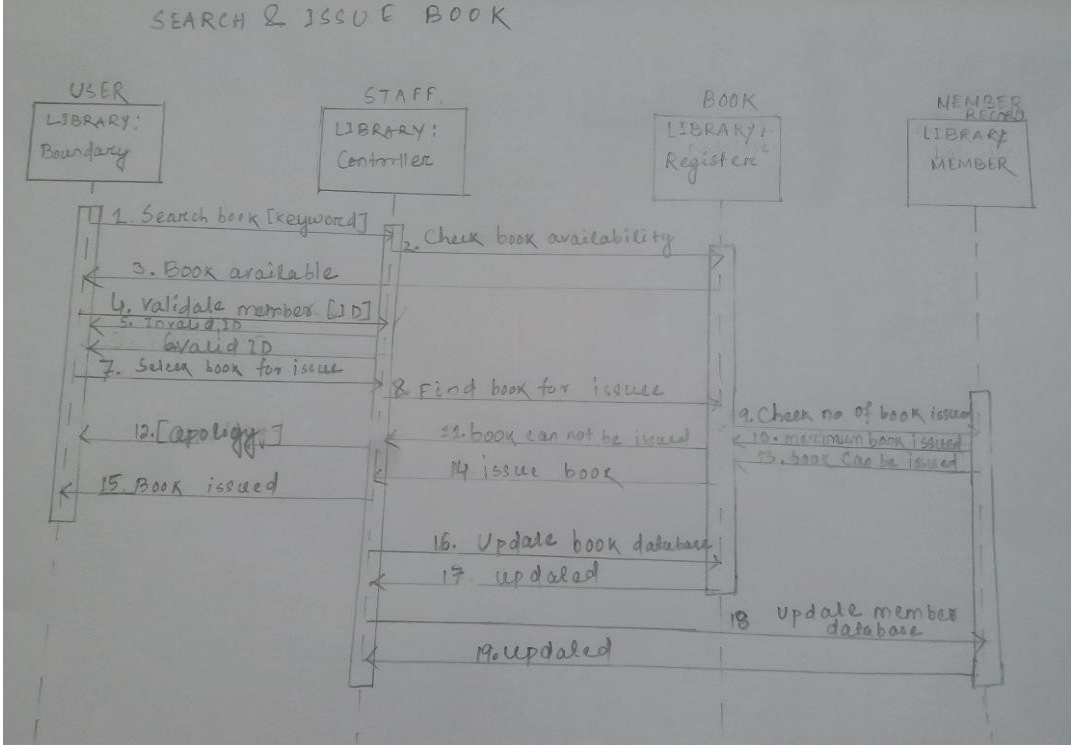
REVERSE ENGINEERING	REEENGINEERING
1. It means to restart the whole 2. It is simply going back to the basics. 3.It is started by identifying the system's component , followed by an investigation into the relationship among those components. 4. 	1. It means to make the previous product again to get better output. 2. It is the process through which the existing flow is modified. 3. It is the investigation and redesign of individual component. 4. 

6(b) Definition of UML. [1+3]

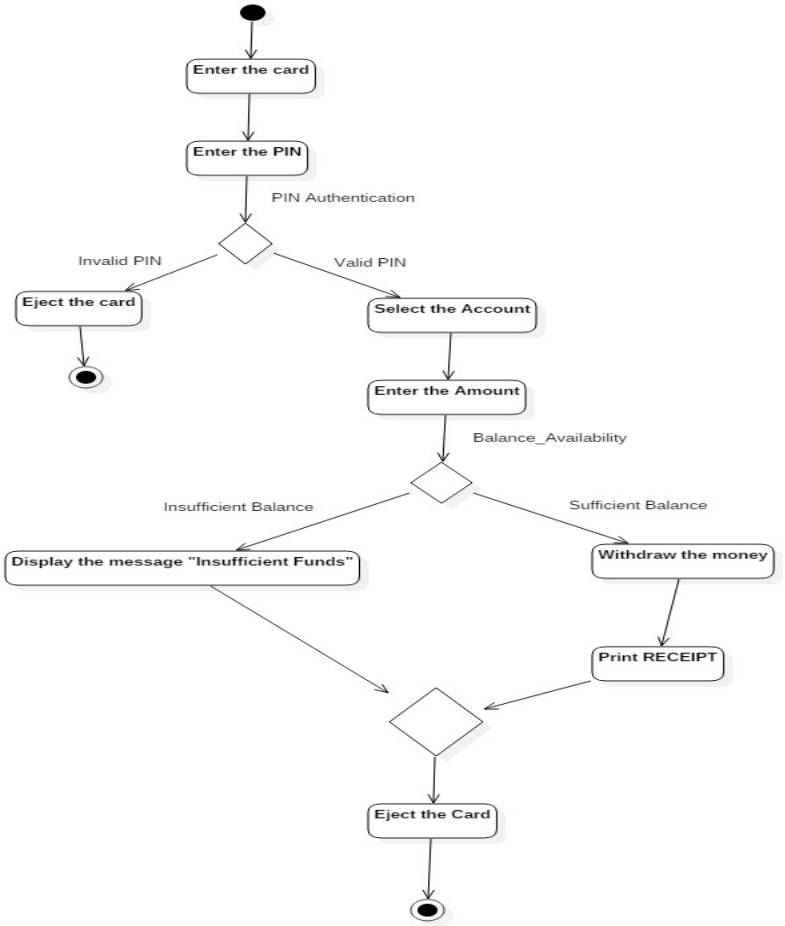
UML Sequence Diagram



Note: The diagrams here are sample ones. Any other diagram relevant to the given question may be evaluated accordingly.



UML Activity Diagram



Q 7 (a)What is white-box testing? Explain at least three white box testing methodologies with suitable examples.

ANS: White- box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. It is an important type of Unit testing.

Three White-box Testing (It may vary according to students)

1)Statement coverage:

In a programming language, a statement is nothing but the line of code or instruction for the computer to understand and act accordingly. A statement becomes an executable statement when it gets compiled and converted into the object code and performs the action when the program is in a running mode.

Hence *"Statement Coverage"*, as the name itself suggests, it is the method of validating whether each and every line of the code is executed at least once.

2) Branch Coverage:

"Branch" in a programming language is like the "IF statements". An IF statement has two branches: T**rue and False**.

So in Branch coverage (also called Decision coverage), we validate whether each branch is executed at least once.

In case of an "IF statement", there will be two test conditions:

- One to validate the true branch and,
- Other to validate the false branch.

Hence, in theory, Branch Coverage is a testing method which is when executed ensures that each and every branch from each decision point is executed.

3) Condition coverage:

Condition coverage is also known as Predicate Coverage in which each one of the Boolean expression have been evaluated to both **TRUE** and **FALSE**.

Example: 1

```
int computeGCD(int x,int y){
1 while(x !=y){
2 if(x>y)
3 x=x-y
4 else y=y-x;
5 }
 6 return x;
}
```

Example: 2

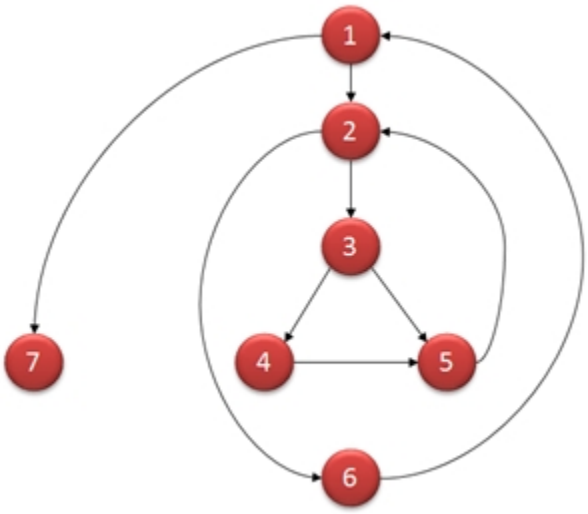
if (A || B) {do something} else {do something else}"

1.Statement coverage will be achieved by generating the test cases where the conditional expression of the while statement needs to be made true and the conditional expression of the if statement needs to be made both true and false. The test set will be {(x=3,y=3),(x=4,y=3),(x=3,y=4)}. All statement will be executed once.(according to example 1)

2. Branch coverage will be achieved by the test suite {x=3,y=3),(x=3,y=2),(x=4,y=3),(x=3,y=4)}(According to example 1)

3.Condition Coverage will be achieved when tested with [0 1], [1 0], then A and B will both have been evaluated to 0 and 1(According to Example 2)

7 (b)	Explain the concept of Control flow graph? Draw the control graph and calculate cyclomatic complexity for the following program segment.
	I=0;
	n=4;
	while (i<n-1) do{
	j= I + 1;
	while (j<n) do{
	if A[i] < A[j]
	swap(A[i], A[j]);}
	i=i+1;
	}



V(G) = 9 - 7 + 2 = 4

V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)

8(a)

Definition of Software Reliability [1 mark]

IEEE Definition:

"The ability of a system or component to perform its required functions under stated conditions for a specified period of time."

According to ANSI

Software Reliability is defined as the probability of failure – free software operation for a specified period of time in a specified environment.

Reliability of a software product:

Essentially denotes its trustworthiness or dependability.

Probability of the product working "correctly" over a given period of time.

Probability of Failure on Demand (POFOD)

Probability of Failure on Demand (POFOD)

POFOD = 0.001

For one in every 1000 requests the service fails per time u

Software reliability metrics: [2 marks]

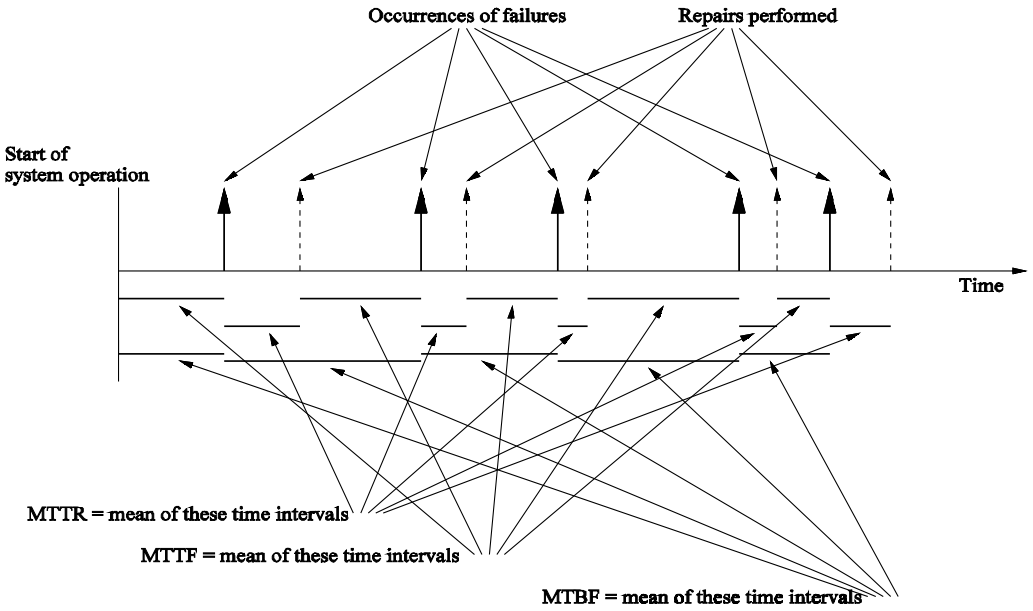
Mean Time Between Failures (MTBF)

Mean Time to Failure (MTTF)

Mean Time to Repair (MTTR)

Mean Time To Failure (MTTF)

MTBF: Mean Time Between Failures (= MTTF + MTTR)



MTBF: Mean Time Between Failures (= MTTF + MTTR)

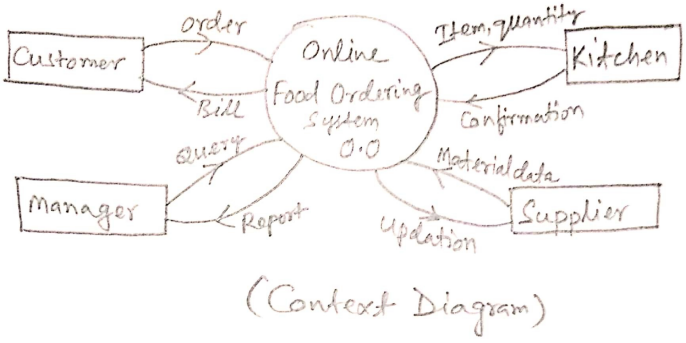
Reliability Growth Modelling: [1 mark]

A reliability growth model is a model of how software reliability grows as errors are detected and repaired. A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained.

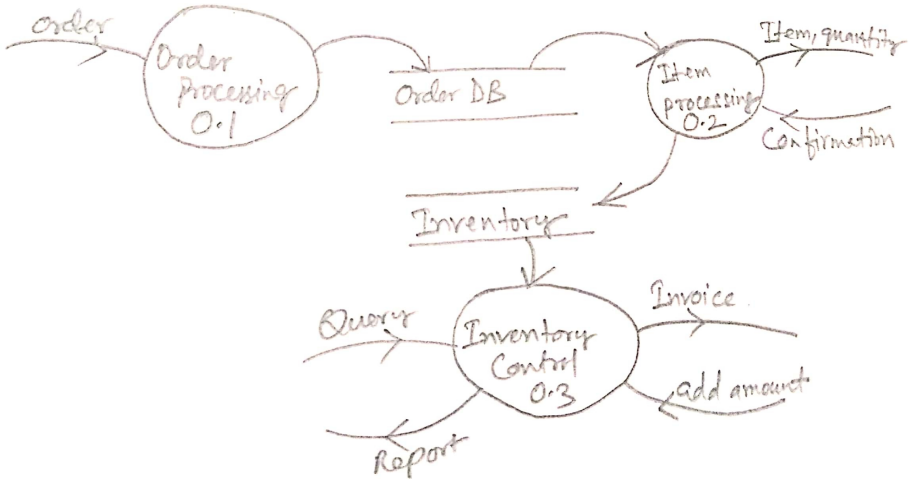
8(b) Context diagram [1 mark]

Level 1 diagram [2 marks]

Level 2 diagram [1 mark



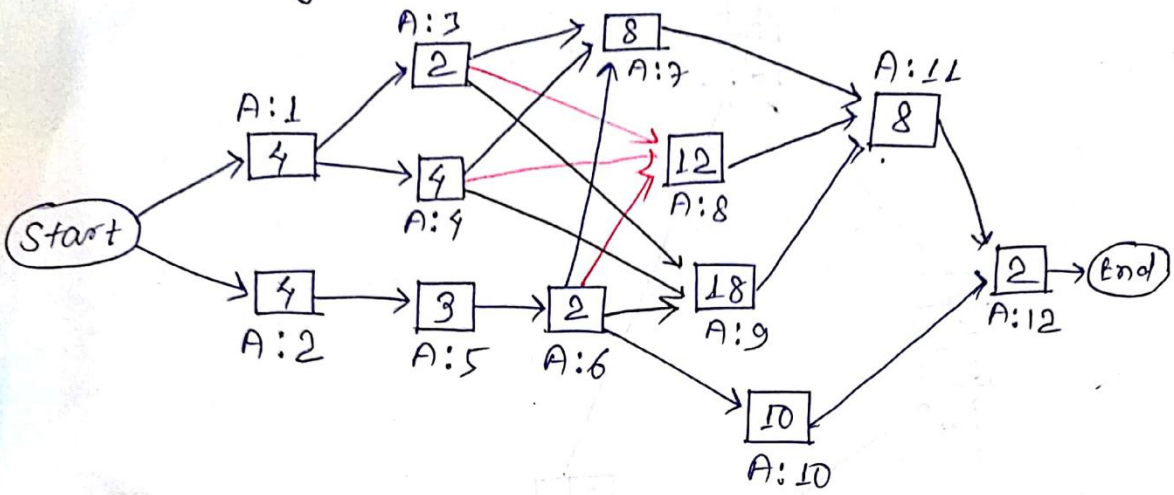
(Context Diagram)



(Level 1 DFD)

Q.4 (i) Solution for CPM problem

(i) Activity Network diagram



(ii) Critical path Identification

Following paths exist in the above diagram.

a) S — 1 — 3 — 7 — 11 — 12 — End (Duration:- 24 weeks)

b) S — 1 — 3 — 8 — 11 — 12 — End (Duration:- 28 weeks)

c) S — 1 — 3 — 9 — 11 — 12 — End (Duration:- 34 weeks)

d) S — 1 — 4 — 7 — 11 — 12 — End (Duration:- 28 weeks)

e) S — 1 — 4 — 8 — 11 — 12 — End (Duration:- 30 weeks)

f) S — 1 — 4 — 9 — 11 — 12 — End (Duration:- 36 weeks)

g) S — 2 — 5 — 6 — 7 — 11 — 12 — End (Duration:- 29 weeks)

h) S — 2 — 5 — 6 — 8 — 11 — 12 — End (Duration:- 31 week)

Critical path → i) S — 2 — 5 — 6 — 9 — 11 — 12 — End (Duration:- 37 weeks)

j) S — 2 — 5 — 6 — 10 — 12 — End (Duration:- 21 weeks)

∴ Critical Path :- S — 2 — 5 — 6 — 9 — 11 — 12 — End; since it is the longest duration path.

(iii) Determination of ES, EF, LS, and LF

a) Early start (ES) and Early finish (EF)

A: 1

ES = 1 (as it is the first task of the project).

EF = Activity Duration + ES − 1

= 4 + 1 − 1 = 4

A: 2

ES = 1 (as it does not have any predecessor)

EF = Act. Duration + ES − 1

= 4 + 1 − 1 = 4

A: 3

ES = EF of Predecessor Act. + 1

= 4 + 1 = 5

EF = Act. Duration + ES − 1

= 2 + 5 − 1 = 6

A: 4

ES = EF of Predecessor Act. + 1

= 4 + 1 = 5

EF = Act. Duration + ES − 1

= 4 + 5 − 1 = 8

A: 5

ES = EF of Predecessor + 1 = 4 + 1 = 5

EF = Act. Duration + ES − 1 = 3 + 5 − 1 = 7

A:6 ES = EF of predecessor + 1 = 7 + 1 = 8

EF = Act. Duration + ES − 1 = 2 + 8 − 1 = 9

A-7 ES = EF of predecessor + 1 = 9 + 1 = 10

{Out of 3 predecessors, choose that has maximum EF time}.

EF = Act. Duration + ES − 1 = 8 + 10 − 1 = 17

A-8 ES = EF of predecessor + 1 = 9 + 1 = 10

{Reason as specified for A-7}

EF = Act. Duration + ES − 1 = 12 + 10 − 1 = 21

A-9 ES = EF of predecessor + 1 = 9 + 1 = 10

{Reason as specified for A-7}

EF = Act. Duration + ES − 1 = 18 + 10 − 1 = 27

A-10 ES = EF of predecessor + 1 = 9 + 1 = 10

EF = Act. Duration + ES − 1 = 10 + 10 − 1 = 19

A-11. ES = EF of predecessor + 1 = 27 + 1 = 28

{Reason as specified for A-7}

EF = Act. Duration + ES − 1 = 8 + 28 − 1 = 35

A-12 ES = EF of predecessor + 1 = 35 + 1 = 36

EF = Act. Duration + ES − 1 = 2 + 36 − 1 = 37

(b) Late Start (LS) and Late Finish (LF)

For critical path activities, LS and LF are same as ES and EF.

So,

A:2 LS = 1, LF = 4

A:5 LS = 5, LF = 7

A:6 LS = 8, LF = 9

A:9 LS = 10, LF = 27

A:11 LS = 28, LS = 35

A:12 LS = 36, LF = 37

For remaining activities,

A:10

LF = LS of Successor − 1 = 36 − 1 = 35

LS = LF − act. Duration + 1 = 35 − 10 + 1 = 26

A:7

LF = LS of Successor − 1 = 28 − 1 = 27

LS = LF − act. duration + 1 = 27 − 8 + 1 = 20

A : 8.

LF = LS of Successor − 1 = 28 − 1 = 27

LS = LF − act. duration + 1 = 27 − 12 + 1

= 16

A − 4

LF = LS of Successor − 1 = 20 − 1 = 19

(The Successor with maximum LS time is Selected).

LS = LF − act. duration + 1 = 19 − 4 + 1

= 16

A − 3

LF = LS of Successor − 1 = 20 − 1 = 19

(Reason as specified for A − 4)

LS = LF − act. duration + 1 = 19 − 2 + 1 = 18

A − 1

LF = LS of Successor − 1 = 18 − 1 = 17

(Reason as specified in A − 4)

LS = LF − act. duration + 1 = 17 − 4 + 1 = 14

ES, EF, LS, and LF

Activity	ES	EF	LS	LF
A-1	1	4	14	17
✓ A-2	1	4	1	4
A-3	5	6	18	19
A-4	5	8	16	19
✓ A-5	5	7	5	7
✓ A-6	8	9	8	9
A-7	10	17	20	27
A-8	10	21	16	27
✓ A-9	10	27	10	27
A-10	10	19	26	35
✓ A-11	28	35	28	35
✓ A-12	36	37	36	37

✓ :- critical Activities.