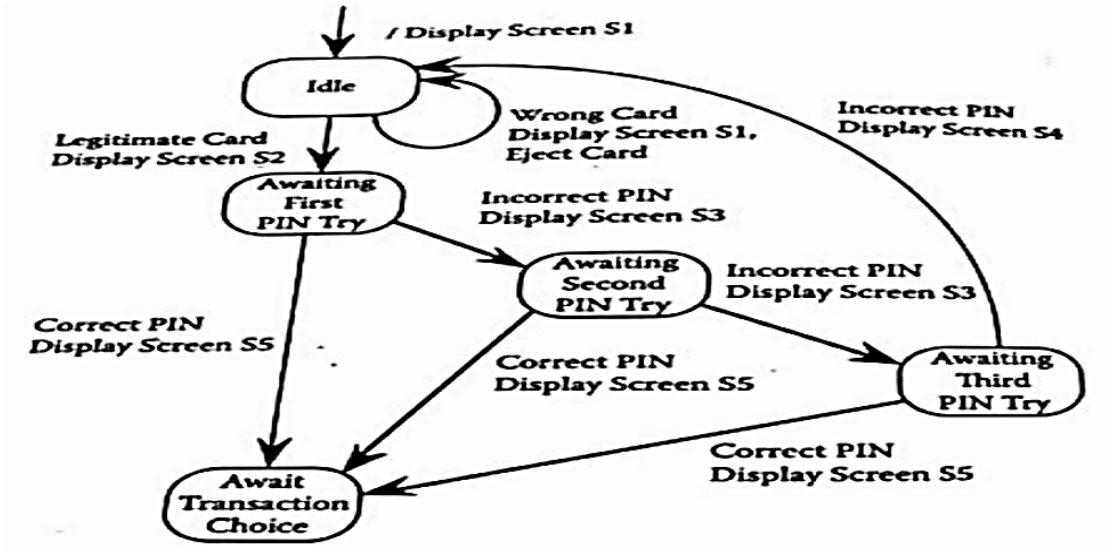
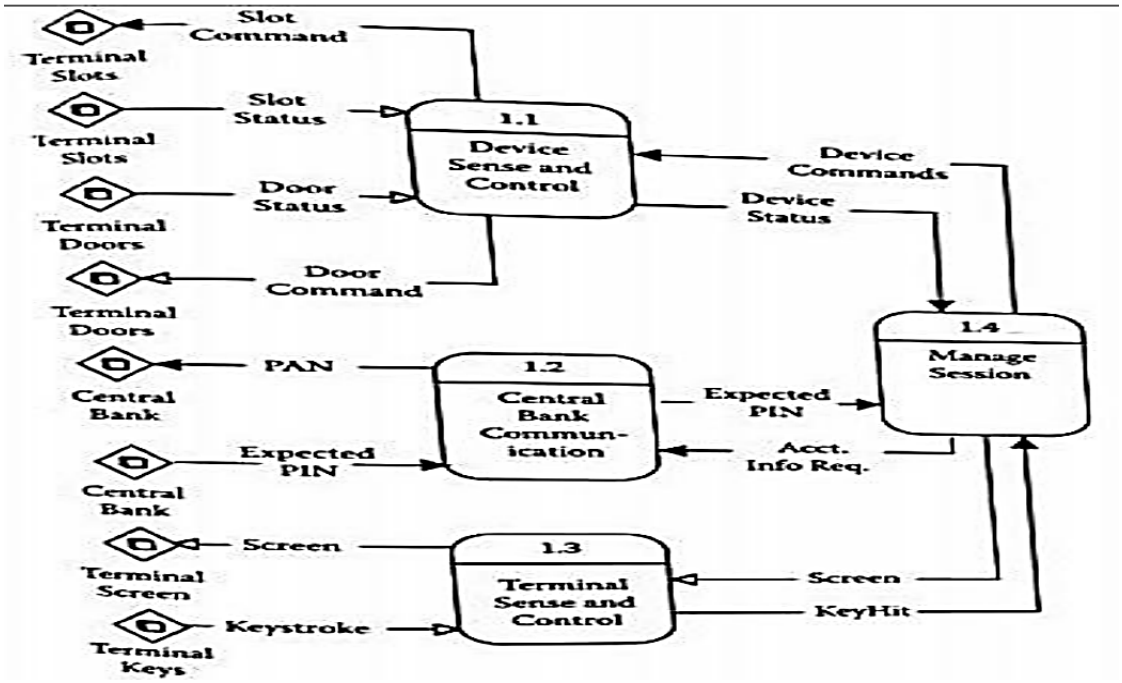
USN										

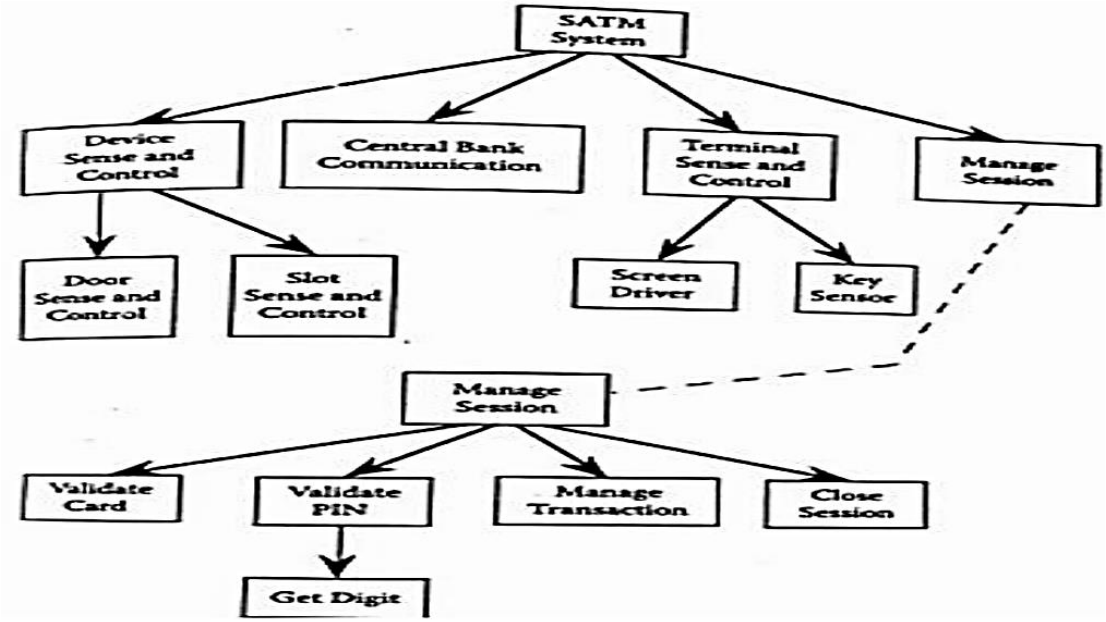
Internal Assessment Test 3 – JUL 2023
Scheme of Evaluation

Sub:	SOFTWARE TESTING					Sub Code:	18IS62	Branch:	ISE
Date:	04/07/2023	Duration:	90 min	Max Marks:	50	Sem/Sec:	VI / A, B & C		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1. (a)	Write Pin Entry finite state machine, Level-1 dataflow diagram, decomposition tree for SATM System. **Solution:** 2 marks for each diagram **Pin Entry finite state machine for SATM System: -**  **Level-1 dataflow diagram for SATM System: -**  	[6]	6	L1

Decomposition tree for SATM System: -



(b) Compare Unit, Integration and system testing. [4] 6 L1

Solution : Comparison between two tests carries 2 marks each.

Unit, Integration, and System Testing

	Unit Test	Integration Test	System Test
Test cases derived from	module specifications	architecture and design specifications	requirements specification
Visibility required	all the details of the code	some details of the code, mainly interfaces	no details of the code
Scaffolding required	Potentially complex, to simulate the activation environment (drivers), the modules called by the module under test (stubs) and test oracles	Depends on architecture and integration order. Modules and subsystems can be incrementally integrated to reduce need for drivers and stubs.	Mostly limited to test oracles, since the whole system should not require additional drivers or stubs to be executed. Sometimes includes a simulated execution environment (e.g., for embedded systems).
Focus on	behavior of individual modules	module integration and interaction	system functionality

2. Explain dependability properties in detail. [10] 4 L2

- Correctness:

✓ A program or system is correct if it is consistent with its specification. A specification divides all possible system behaviors into two classes, *successes (or correct executions) and failures. All of the possible* behaviors of a correct system are successes.

✓ A program cannot be mostly correct or somewhat . It is absolutely correct on all possible behaviors, or else it is not correct.

- Reliability:

✓ It is a statistical approximation to correctness which means 100% reliable = correctness.

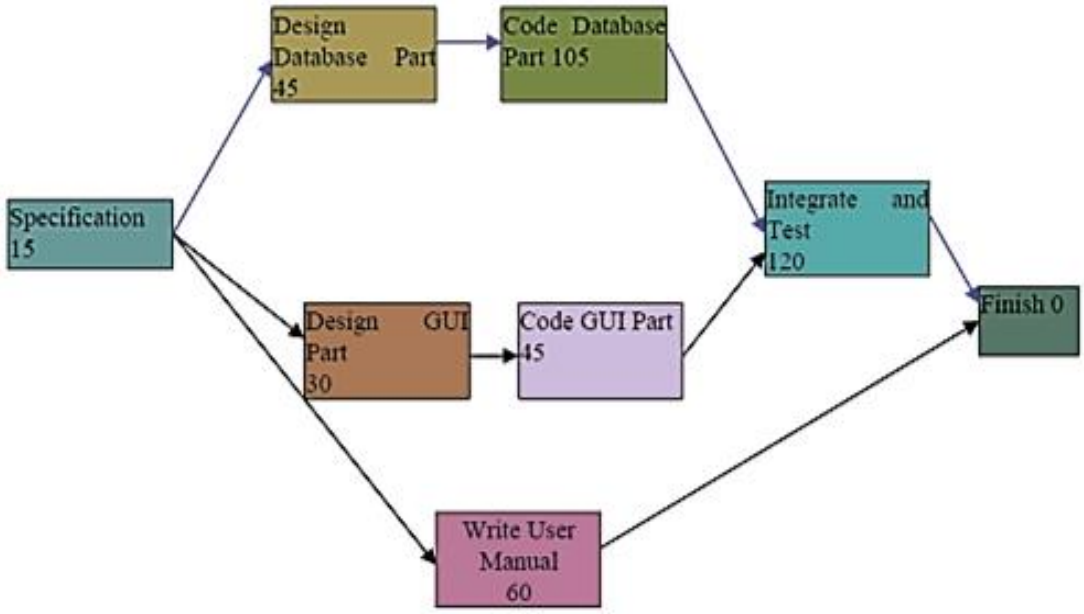
✓ It is the likelihood of correct function for some ``unit'' of behavior.

✓ It is relative to a specification and usage profile. The same program can be more or less reliable depending on how it is used.

	• Availability: ✓ Particular measures of reliability can be used for different units of execution and different ways of counting success and failure. ✓ *Availability is an appropriate measure when a failure has some* duration in time. • *Mean time between failures (MTBF) :* ✓ *Mean time between failures (MTBF) is yet another measure of reliability, also using time as the unit* of execution. ✓ The hypothetical network switch that typically fails once in a 24-hour period and takes about an hour to recover has a mean time between failures of 23 hours. ✓ Note that availability does not distinguish between two failures of 30 minutes each and one failure lasting an hour, while MTBF does. • Safety and hazards: ✓ Software safety is an extension of the well-established field of system safety into software. ✓ Safety is concerned with preventing certain undesirable behaviors, called *hazards.* ✓ Software safety is typically a concern in "critical" systems such as avionics and medical systems, but the basic principles apply to any system in which undesirable behaviors can be distinguished from failure. • Robustness: ✓ Software that fails under some conditions, which violate the premises of its design, may still be "correct" in the strict sense, yet the manner in which the software fails is important. ✓ It is acceptable that the word processor fails to write the new file that does not fit on disk, but unacceptable to also corrupt the previous version of the file in the attempt. ✓ It is acceptable for the database system to cease to function when the power is cut, but unacceptable for it to leave the database in a corrupt state.			
3.	Explain basic principles of testing process framework. a. Sensitivity b. Restriction c. Redundancy d. Visibility e. Feedback f. partition. • Human developers make errors, producing faults in software. Faults may lead to failures, but faulty software may not fail on every execution. • The sensitivity principle states that it is better to fail every time than sometimes. • If a fault is detected in unit testing, the cost of repairing is relatively small. • If a fault survives at the unit level, but triggers a failure detected in integration testing, the cost of correction is much greater. • If the first failure is detected in system or acceptance testing, the cost is very high indeed, and the most costly faults are those detected by customers in the field. • Redundancy is the opposite of independence. In software test and analysis, we wish to detect faults that could lead to differences between intended behavior and actual behavior, so the redundancy is in the form of making intentions explicit. • Redundancy can be introduced to declare intent and automatically check for consistency. • Static type checking is a classic application of this principle: The type declaration is a statement of intent that is at least partly redundant with the use of a variable in the source code. • The type declaration constrains other parts of the code, so a consistency check can be applied. • When there are no acceptably cheap and effective ways to check a property, checking can be done on more restrictive property or limit the check to a smaller, more restrictive class of programs. • Consider the problem of ensuring that each variable is initialized before it is used, on every execution. It is not possible for a compiler or analysis tool to precisely determine whether it holds. • Partition, often also known as "divide and conquer," is a general engineering principle. Dividing a complex problem into sub problems to be attacked and solved independently is probably the most common human problem-solving strategy. • In Analysis and testing the partition principle is widely used and exploited. • Partitioning can be applied both at process and technique levels. • At the process level, we divide complex activities into sets of simple activities that can be attacked independently. For example, testing is usually divided into unit, integration, subsystem, and system testing. In this way, we can focus on different sources of faults at different steps, and at each step, we can take advantage of the results of the former steps.	[10]	4	L2

- Visibility means the ability to measure progress or status against goals.
- In software engineering, the visibility principle is in the form of process visibility, and project schedule visibility.
- Quality process visibility also applies to measuring achieved (or predicted) quality against quality goals.
- Visibility is closely related to observability, the ability to extract useful information from a software artifact.
- Feedback is another classic engineering principle that applies to analysis and testing.
- Feedback applies both to the process itself (process improvement) and to individual techniques.
- Systematic inspection derive its success from feedback.
- Participants in inspection are guided by checklists, and checklists are revised and refined based on experience.

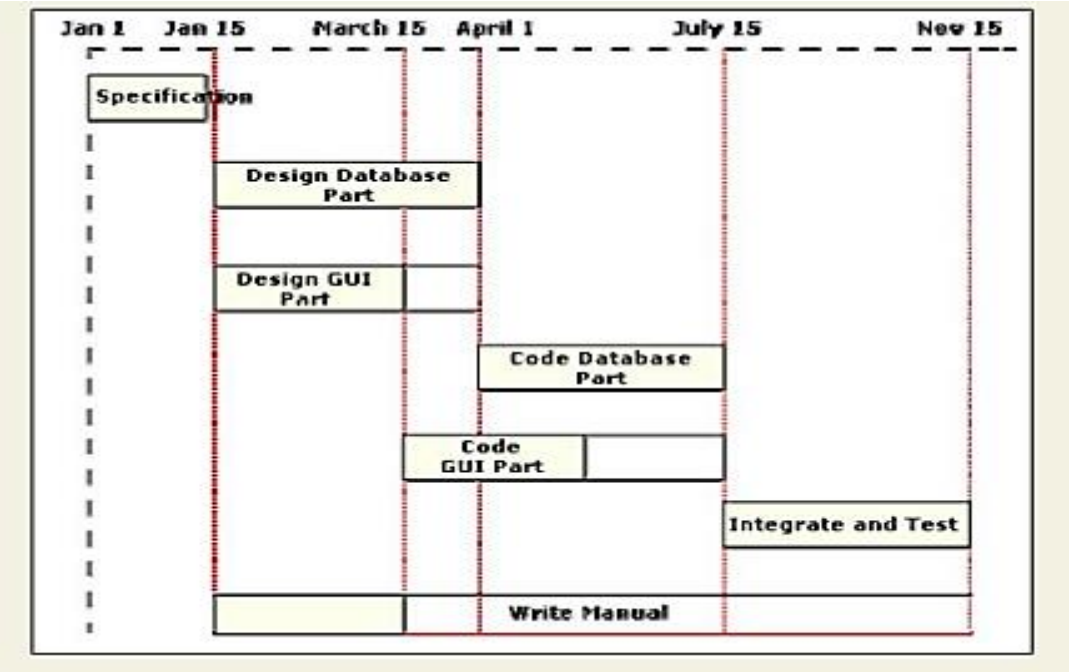
4. Apply Risk Planning and Critical path analysis for the given activity diagram in software process:



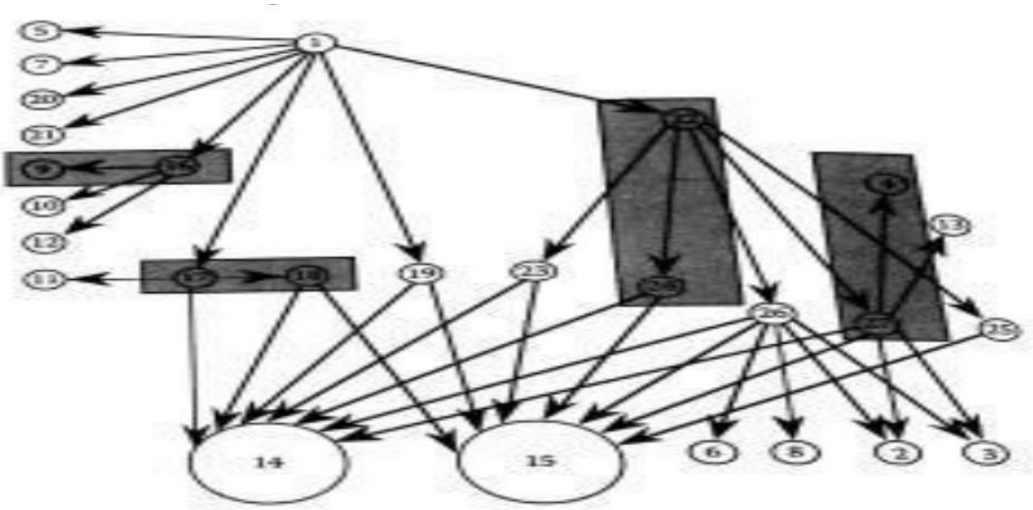
[10] 4 L3

Solution:-

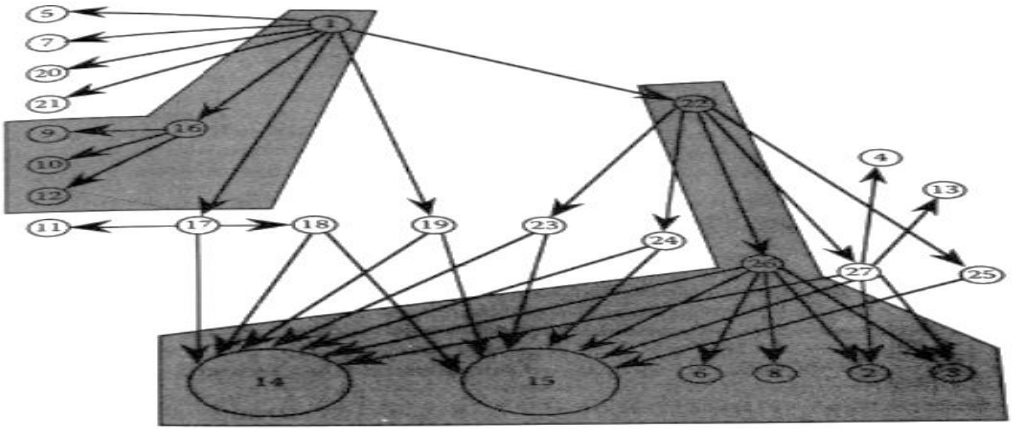
Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code database	60	165	60	165	0
Code GUI part	45	90	120	165	75
Integrate and test	165	285	165	285	0
Write user manual	15	75	225	285	210



5.	Explain the call graph based integration with the help of : a. Pair-wise Integration b. Neighborhood Integration.	[10]	5	L2



Pairwise Integration



Neighbourhoods Integration

6.	Apply Slice based Testing for Commission Problem.	[10]	3	L3

```
1.  Program Commission (INPUT,OUTPUT)
2.      Dim locks, stocks, barrels As Integer
3.      Dim lockPrice, stockPrice, barrelPrice As Real
4.      Dim totalLocks, totalStocks, totalBarrels As Integer
5.      Dim lockSales, stockSales, barrelSales As Real
6.      Dim sales, commission As Real
7.      lockPrice = 45.0
8.      stockPrice = 30.0
9.      barrelPrice = 25.0
10.     totalLocks = 0
11.     totalStocks = 0
12.     totalBarrels = 0
13.     Input(locks)
14.     While NOT(locks = -1)   'loop condition uses -1 to
        indicate end of data
15.         Input(stocks, barrels)
16.         totalLocks = totalLocks + locks
17.         totalStocks = totalStocks + stocks
18.         totalBarrels = totalBarrels + barrels
19.         Input(locks)
20.     EndWhile
21.     Output("Locks sold: ", totalLocks)
22.     Output("Stocks sold: ", totalStocks)
23.     Output("Barrels sold: ", totalBarrels)
24.     lockSales = lockPrice * totalLocks
25.     stockSales = stockPrice * totalStocks
26.     barrelSales = barrelPrice * totalBarrels
27.     sales = lockSales + stockSales + barrelSales
28.     Output("Total sales: ", sales)
29.     If (sales > 1800.0)
30.         Then
31.             commission = 0.10 * 1000.0
32.             commission = commission + 0.15 * 800.0
33.             commission = commission + 0.20 *
                (sales-1800.0)
34.     Else If (sales > 1000.0)
35.         Then
36.             commission = 0.10 * 1000.0
37.             commission = commission + 0.15 *
                (sales-1000.0)
38.         Else
39.             commission = 0.10 * sales
40.         EndIf
41.     EndIf
42.     Output("Commission is $", commission)
43.     End Commission
```

- **DEFINITION**
  - Given a program P and a set V of variables in P, a slice on the variable set V at statement n, written S(V, n) is the set of all statements in **P prior to node n that contribute to the values of variables in V at node n**

Two forms of ***definition nodes***

- I-def (defined by input, e.g. scanf())
- A-def (defined by assignment)

SLICE ON LOCK VARIABLE

S1: S(locks, 13) = {13} DEFINING NODE I-DEF
S2: S(locks, 14) = {13, 14, 19, 20}
S3: S(locks, 16) = {13, 14, 19, 20}
S4: S(locks, 19) = {19} DEFINING NODE I-DEF

SLICE ON STOCKS AND BARRELS

S_5: S(stocks,15) = {13,14,15,19,20}

S_6: S(stocks,17) = {13,14,15, 19,20}

S_7: S(barrels,15) = {13,14,15,19,20}

S_8: S(barrels,18) = {13,14,15, 19,20}

SLICE ON TOTAL LOCKS

S_9: S(totallocks,10) = {10} (A-def)

S_{10}: S(totallocks,16) = {10,13,14,16,19,20} (A-def & C-use)

S(totallocks,21)= {10,13,14,16,19,20} 21 is an O-Use of totallocks, excluded

S_{11}: S(totallocks,24) = {10,13,14,16,19,20} (24 is a C-use of total locks)

SLICE ON TOTAL STOCKS AND TOTAL BARRELS

S_{12}: S(totalstocks,11) = {11} (A-def)

S_{13}: S(totalstocks,17) = {11,13,14,15,17,19,20} (A-def & C-use)

S_{14}: S(totalstocks,22) = {11,13,14,15,17,19,20}(22 is an O-Use of totalstocks)

S_{15}: S(totalbarrels,12) = {12}

S_{16}: S(totalbarrels,18) = {12,13,14,15,18,19,20} (A-def & C-use)

S_{17}: S(totalbarrels,23) = {12,13,14,15,18,19,20} (23 is an O-Use of totalbarrels)

Faculty Signature CCI Signature HOD Signature