USN	1	P	E							

PESIT Bangalore South Campus
Hosur road, 1km before Electronic City, Bengaluru -100
Department of MCA

INTERNAL ASSESSMENT TEST – II

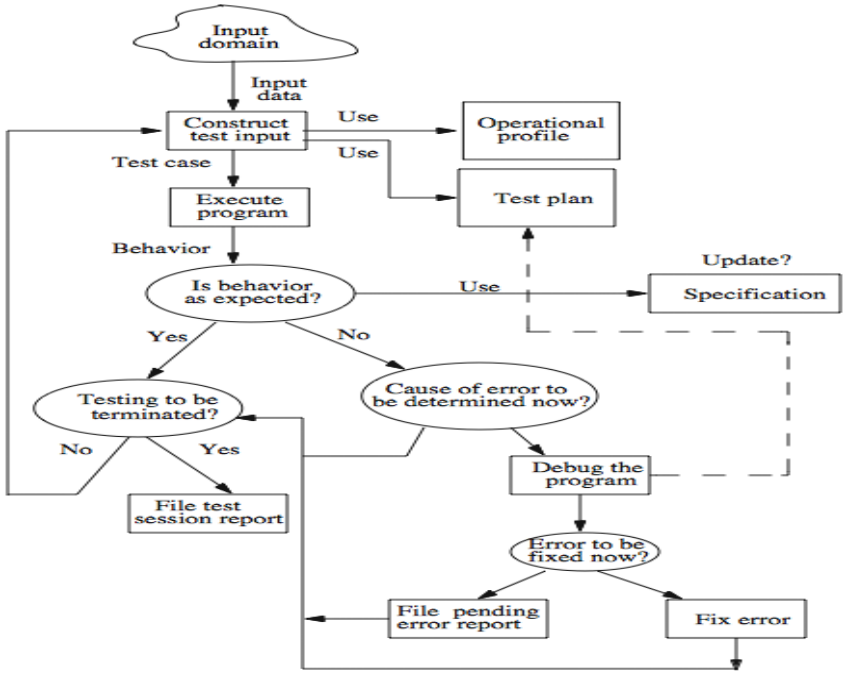
Date : 03/04/2018 Max Marks : 50
Subject & Code : Software Testing and practices 16MCA43 Sem& Section : 1
Name of faculty : Manjula.C.M.Prasad Time : 08.30-10.00

Note: Answer any Five full questions.

PART-A

1. a Write Testing and Debug cycle with a neat diagram 4

Testing is the process of determining if a program has any errors. When testing reveals an error, the process used to determine the cause of this error and to remove it, is known as debugging

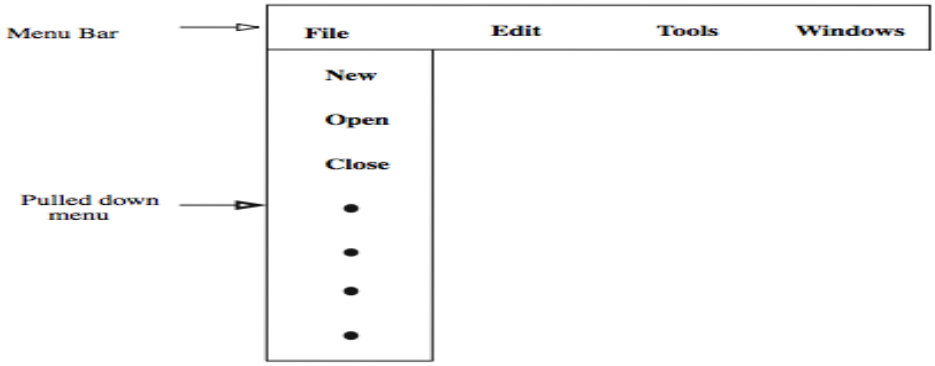


Testing is the process of determining if a program has any errors.
When testing reveals an error, the process used to determine the cause of this error and to remove it, is known as debugging.

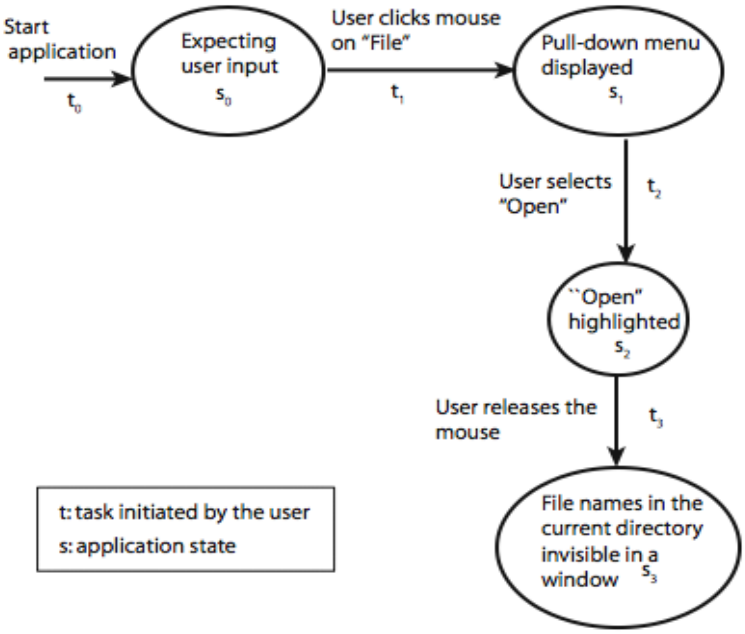
b Explain the program behavior by state diagram and assess the correctness of the program behavior. 4

Can be specified in several ways: plain natural language, a state diagram, formal mathematical specification, etc.
A state diagram specifies program states and how the program changes its state on an

input sequence. inputs.



2.



In the first step one observes the behavior.
In the second step one analyzes the observed behavior to check if it is correct or not.
Both these steps could be quite complex for large commercial programs.
The entity that performs the task of checking the correctness of the observed behavior is known as an oracle.

OR

What is test metrics? Discuss about the different types of metrics used in software testing and their relationships. 8

The term metric refers to a standard of measurement. Metrics can be computed at the organizational , process, project and product levels. Organizational level metrics allow senior management to monitor the overall strength of the organization and points to areas of weakness. These metrics help senior management in setting new goals and plan for resources needed to realize these goals.
Project metrics relate to a specific project. These are useful in the monitoring and control of a specific project. The ratio of actual-to-planned system test effort is one project metric.
Process metrics is used to assess the goodness of the process. When a test process consists of several phases. For example unit test, integration test and system test, one can measure how many defects were found in each phase.

Product metrics relate to a specific product such as a compiler for a programming language. There are cyclomatic complexity and halsted metrics.

PART-B

3. a Discuss about all the IEEE standard definition of the Testing terminologies. 4

- **Test case**: a set of inputs, execution conditions, and a pass/fail criterion.
- **Test case Specification**: a requirement to be satisfied by one or more test cases.
- **Test obligation:** a partial test case specification, requiring some property deemed important to thorough testing.
- **Test suite**: a set of test cases.
- **Test** or **test execution**: the activity of executing test cases and evaluating their results.

Adequacy criterion: a predicate that is true (satisfied) or false (not satisfied) of a ⟨program, test suite⟩ pair.

- A Test Case Includes input, the expected output, pass/fail criteria and the environment in which the test is being conducted.
- A Test Case specification is a requirement to be satisfied by one or more test cases.
- Specification-based testing uses the specification of the program as the point of reference for test input data selection and adequacy.

The distinction between a test case and test specification is similar to the distinction between program specification and program

b Discuss about Test generation strategies with a neat diagram 4

Requirements, Models- FSM, state charts, petri charts, UML and algorithms

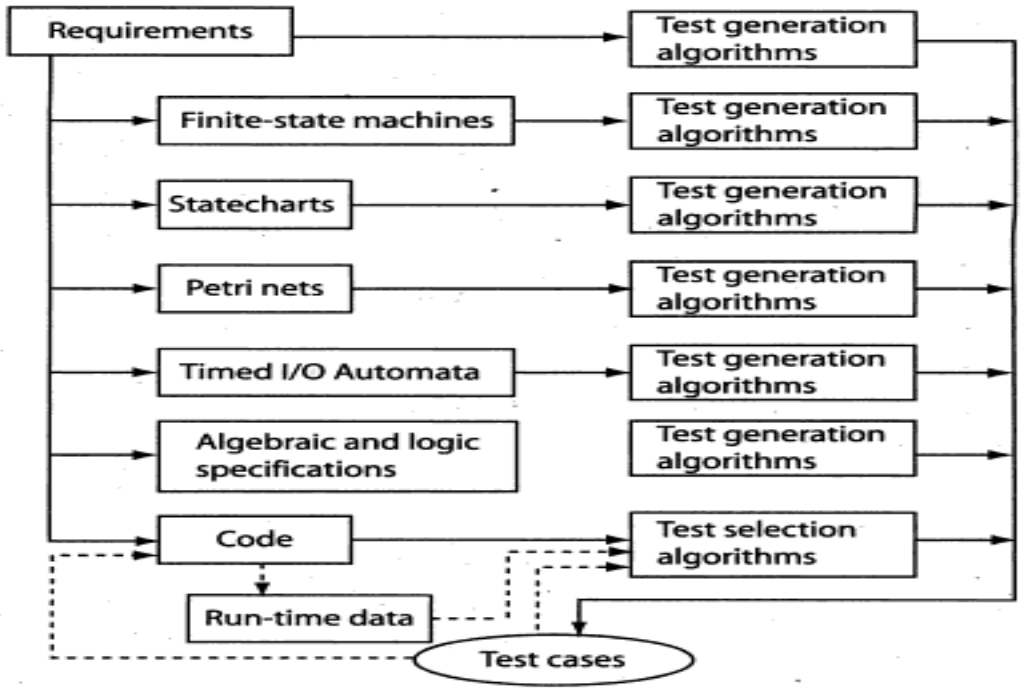


Fig. 1.12 Requirements, models, and test generation algorithms.

- **Model based-** FSMs, statecharts, petrinets and timed I/O automata are some of the well known and used formal notations for modelling various subset requirements.
- · Sequence & activity diagrams in UML also exist and are used as models of subsets of requirements.
- **Specification based-** based on requirements.
- **Code based-**There also exist techniques to generate tests directly from the code

i.e. code based test generation.

- Code based test generation techniques are also used during regression testing when there is often a need to reduce the size of the suite or prioritize tests, against which a regression test is to be performed

4.

OR

Discuss about all the basic principles of testing in detail with example. 8

Sensitivity
Redundancy
Restriction
Partition
Visibility
Feedback

PART –C

5. Explain Basic block, Control flow graph and DD-path with example. 8

```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATrinagle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is", a)
7  Output("Side B is", b)
8  Output("Side C is", c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12 EndIf
13 If IsATriangle
14    Then  If (a = b) AND (b = c)
15             Then Output ("Equilateral")
16             Else  If (a≠b) AND (a≠c) AND (b≠c)
17                      Then Output ("Scalene")
18                      Else Output ("Isosceles")
19                   EndIf
20             EndIf
21    Else  Output("Nota a Triangle")
22 EndIf
23 End triangle2
```

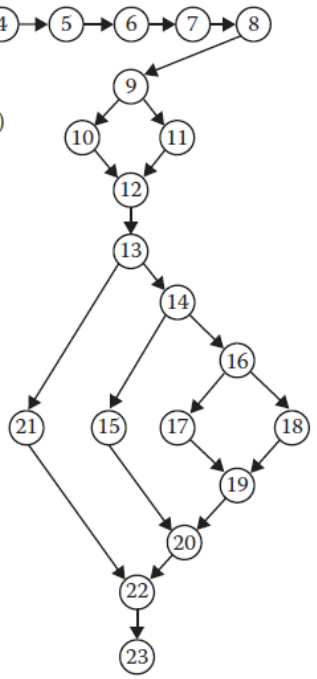
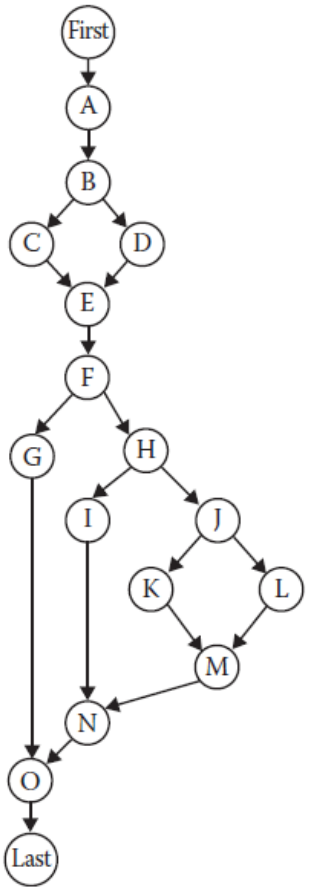


Figure 8.2 Nodes	DD-Path	Case of definition
4	First	1
5-8	A	5
9	B	3
10	C	4
11	D	4
12	E	3
13	F	3
14	H	3
15	I	4
16	J	3
17	K	4
18	L	4
19	M	3
20	N	3
21	G	4
22	O	3
23	Last	2



OR

6. Discuss about structural test coverage metrics with various metric-based testing with example. 8

Structural Coverage Metrics

Measurement of structural coverage of code is a means of assessing the thoroughness of testing. There are a number of metrics available for measuring structural coverage, with increasing support from software tools. Such metrics do not constitute testing techniques, but a measure of the effectiveness of testing techniques.

A coverage metric is expressed in terms of a ratio of the metric items executed or evaluated at least once to the total number of metric items. This is usually expressed as a percentage.

$$\text{Coverage} = \frac{\text{items executed at least once}}{\text{total number of items}}$$

Statement Coverage

Statement Coverage = s/S

where:

s = Number of statements executed at least once.

S = Total number of executable statements.

Statement coverage is the simplest structural coverage metric. From a measurement point of view one just keeps track of which statements are executed, then compares this to a list of all executable statements. Statement coverage is therefore suitable for **automation**.

Example

```
1. null;
2. DO_SOMETHING;
3. null;
4. ANOTHER_STATEMENT;
```

Decision coverage

Decision coverage = d/D

where:

d = Number of decision outcomes evaluated at least once.

D = Total number of decision outcomes.

To achieve 100% decision coverage, each condition controlling branching of the code has to evaluate to both true and false. In example 4a, decision coverage requires two test cases.

Example

```
1. if CONDITION then
2.     DO_SOMETHING;
3. else
4.     DO_SOMETHING_ELSE;
5. end if;
```

Test	CONDITION
1	True
2	False

LCSAJ Coverage

An LCSAJ is defined as an unbroken linear sequence of statements:

- which begins at either the start of the program or a point to which the control flow may jump,
- which ends at either the end of the program or a point from which the control flow may jump,
- and the point to which a jump is made following the sequence.

Hennell [3] gives a full explanation and some examples to help illustrate the definition of an LCSAJ.

LCSAJ coverage = l/L

where:

l = Number of LCSAJs exercised at least once.

L = Total number of LCSAJs.

Path Coverage

Path Coverage = p/P

where:

p = Number of paths executed at least once.

P = Total number of paths.

Path coverage looks at complete paths through a program. For example, if a module contains a loop, then there are separate paths through the module for one iteration of the loop, two iterations of the loop, through to n iterations of the loop.
The **thoroughness** of test data designed to achieve 100% path coverage is higher than that for decision coverage.

Example

```
1. while A loop
2.    A_STATEMENT;
3. end loop;
4. while B loop
5.    ANOTHER_STATEMENT;
6. end loop;
```

Test	A	B
1	False	False
2	(True, False)	False

3	(True, False)	(True, False)
4	(True, False)	(True True, False)

Condition Operand Coverage

Condition Operand Coverage = c/C

where:

c = Number of condition operand values evaluated at least once.

C = Total number of condition operand values.

Condition operand coverage gives a measure of coverage of the conditions which could cause a branch to be executed. Condition operands can be readily identified from both design and code, with condition operand coverage directly related to the operands. This facilitates **automation** and makes condition operand coverage both **comprehensible** and **maintainable**.

Example

```
1. if A and B then
2.     DO_SOMETHING;;
3. else
4.     DO_SOMETHING_ELSE;
5. end if;
```

Boolean Operand Effectiveness Coverage

Boolean Operand Effectiveness Coverage = b/B

where:

b = Number of Boolean operands shown to independently influence the outcome of Boolean expressions.

B = Total number of Boolean operands.

To achieve Boolean operand effectiveness coverage, each Boolean operand must be shown to be able to independently influence the outcome of the overall Boolean expression. The straight forward relationship between test data and the criteria of Boolean operand effectiveness coverage makes the metric **comprehensible** and

associated test data **maintainable**. This is illustrated by example 9a.

Example

```
1. if (A and B) or C then
2.     DO_SOMETHING;;
3. end if;
```

Test	A	B	C
1	true	true	false
2	false	true	false

(Tests 1 and 2 show independence of A)

3	true	true	false
4	true	false	false

(Tests 3 and 4 show independence of B)

5	false	false	true
6	false	false	false

(Tests 5 and 6 show independence of C)

PART-D

7. Considering commission problem write DD-path and Basis path with a neat diagram and write the test cases for the paths identified. 8

```
1   #include<stdio.h>
2   int main()
3   {
4           int locks, stocks, barrels, tlocks, tstocks, tbarrels;
5           float lprice,sprice,bprice,lsales,ssales,bsales,sales,comm;
6           lprice=45.0;
7           sprice=30.0;
8           bprice=25.0;
9           tlocks=0;
10          tstocks=0;
11          tbarrels=0;
12          printf("\nenter the number of locks and to exit the loop enter -1 for
            locks\n");
            scanf("%d", &locks);
14          while(locks!=-1)  {
```

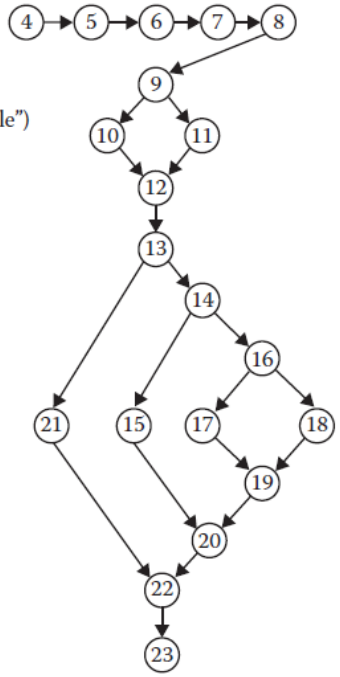
```
15                  printf("enter the number of stocks and barrels\n");
                    scanf("%d%d",&stocks,&barrels);
16                  tlocks=tlocks+locks;
17                  tstocks=tstocks+stocks;
18                  tbarrels=btarrels+barrels;
                    printf("\nenter the number of locks and to exit the loop enter -1
19                  for locks\n");
                    scanf("%d",&locks);
20          }
21          printf("\ntotal locks = %d\",tlocks);
22          printf("total stocks =%d\n",tstocks);
23          printf("total barrels =%d\n",tbarrels);

24          lsales = lprice*tlocks;
25          ssales=sprice*tstocks;
26          bsales=bprice*tbarrels;
27          sales=lsales+ssales+bsales;
28          printf("\nthe total sales=%f\n",sales);
29          if(sales > 1800.0)
30          {
31                  comm=0.10*1000.0;
32                  comm=comm+0.15*800;
33                  comm=comm+0.20*(sales-1800.0);
                }
34          else if(sales > 1000)
35          {
36                  comm =0.10*1000;
37                  comm=comm+0.15*(sales-1000);
              }
38          else
39                  comm=0.10*sales;
40          printf("the commission is=%f\n",comm);
41          return 0;
42  }
```

8.

OR

Considering the triangle problem write DD-path and Basis path with a neat diagram and write the test cases for the paths identified.	0 8	

```
1  Program triangle2
2  Dim a,b,c As Integer
3  Dim IsATrinagle As Boolean
4  Output("Enter 3 integers which are sides of a triangle")
5  Input(a,b,c)
6  Output("Side A is", a)
7  Output("Side B is", b)
8  Output("Side C is", c)
9  If (a < b + c) AND (b < a + c) AND (c < a + b)
10    Then IsATriangle = True
11    Else IsATriangle = False
12 EndIf
13 If IsATriangle
14    Then  If (a = b) AND (b = c)
15             Then Output ("Equilateral")
16             Else  If (a≠b) AND (a≠c) AND (b≠c)
17                      Then Output ("Scalene")
18                      Else Output ("Isosceles")
19                   EndIf
20             EndIf
21    Else  Output("Nota a Triangle")
22 EndIf
23 End triangle2
```



V(G) = 23 - 20 + 2(1) = 5

Basis Path Set B1

p1: 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 18, 19, 20, 22, 23 (mainline)

p2: 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 16, 18, 19, 20, 22, 23 (flipped at 9)

p3: 4, 5, 6, 7, 8, 9, 11, 12, 13, 21, 22, 23 (flipped at 13)

p4: 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 20, 22, 23 (flipped at 14)

p5: 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 19, 20, 22, 23 (flipped at 16)

PART-E

9. Discuss about Alternative life cycle models. 8

There are three derivatives of the waterfall model, each involves a series of increments or builds:

– Incremental development

– Evolutionary development

– Spiral Model

In Incremental Development:

• The whole scope of the software is known. Break it into manageable increments. New functionality is constantly added onto the existing code base. Each new build is created by developing new code, unit testing the new code, testing the integration of the new code with the existing code, and testing the entire new system with both regression and progression testing.

– In Evolutionary Development:

• The whole scope of the software is not known. We allow for changes in the requirements and specifications as the software is developed. Thus, user feedback can be incorporated by directing future builds. The system evolves to meet the changing needs of the user

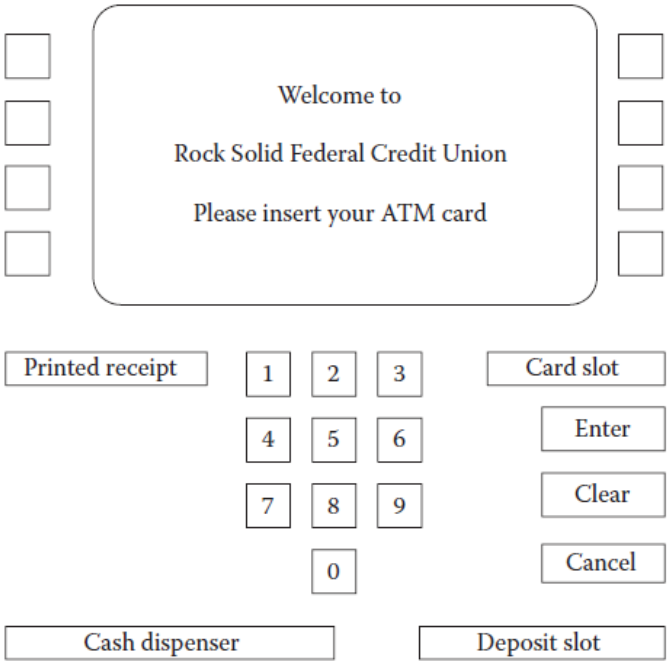
– In the Spiral Model

• A build is defined first in terms of rapid prototyping and then is subjected to a go/no-go decision based on technologyrelated

risk factors
– In all the three spin-off models
• a build is a set of deliverable end-user functionality, thus all three yield earlier synthesis
• Also earlier customer feedback

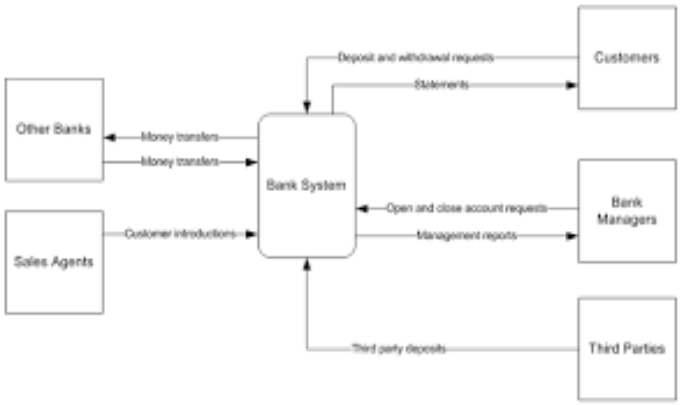
OR

10. Discuss about SATM system with finite state machine and context diagram. 8



Screen 1 Welcome please insert your ATM card	Screen 2 Please enter your PIN ____	Screen 3 Your PIN is incorrect. Please try again.
Screen 4 Invalid ATM card. It will be retained.	Screen 5 Select transaction: balance > deposit > withdrawal >	Screen 6 Balance is $dddd.dd
Screen 7 Enter amount. Withdrawals must be multiples of $10	Screen 8 Insufficient Funds! Please enter a new amount	Screen 9 Machine can only dispense $10 notes
Screen 10 Temporarily unable to process withdrawals. Another transaction?	Screen 11 Your balance is being updated. Please take cash from dispenser.	Screen 12 Temporarily unable to process deposits. Another transaction?
Screen 13 Please insert deposit into deposit slot.	Screen 14 Your new balance is being printed. Another transaction?	Screen 15 Please take your receipt and ATM card. Thank you.

Context diagram



Finite state machine diagram

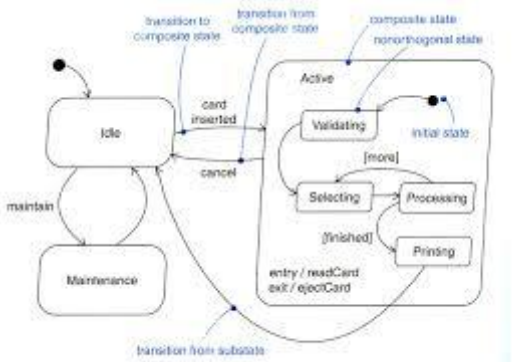


Figure 22-5: Sequential Substates