CMR
INSTITUTE OF
TECHNOLOGY

Internal Assessment Test 3 – May. 2018

Scheme and Solution

Sub:	Software Testing							Code:	15IS63
Date:	22/ 05/2018	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	ISE

Note: Answer any five questions:

1. a) Explain decomposition based integration with example. (4 M)

Functional decomposition, is expressed either as a tree or in textual form. These help in the order in which modules are to be integrated. There are four choices: from the top of the tree downward (top down), from the bottom of the tree upward (bottom up), some combination of these (sandwich), or most graphically, none of these (the big bang).All of these integration orders presume that the units have been separately tested,

Thus the goal of decomposition based integration is to test the interfaces among separately tested units.

b) With an example explain the top-down integration and bottom-up integration. (6 M)

Top-down integration begins with the main program (the root of the tree). Any lower level unit that is called by the main program appears as a "stub", where stubs are pieces of throw-away code that emulate a called unit. If we performed top-down integration testing for the SATM system, the first step would be to develop stubs for all the units called by the main program:

Here are two examples of stubs.

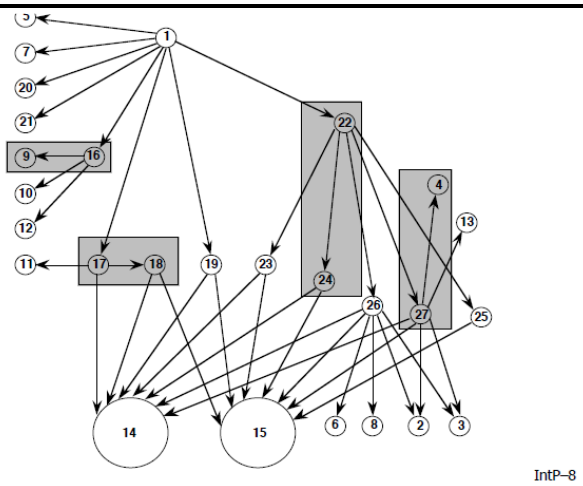
```
Procedure GetPINforPAN (PAN, ExpectedPIN)   STUB
IF PAN = '1123' THEN PIN := '8876';
IF PAN = '1234' THEN PIN := '8765';
IF PAN = '8746' THEN PIN := '1253';
End,

Procedure KeySensor (KeyHit) STUB
data: KeyStrokes STACK OF ' 8 ' , ' 8 ' , ' 7 ' , ' cancel '
KeyHit = POP (KeyStrokes)
End,
```

In the stub for GetPINforPAN, the tester replicates a table look-up with just a few values that will appear in test cases. In the stub for KeySensor, the tester must devise a sequence of port events that

Can occur once each time the KeySensor procedure is called. (Here, we provided the keystrokes to

	partially enter the PIN '8876', but the user hit the cancel button before the fourth digit.) In practice, the effort to develop stubs is usually quite significant. There is good reason to consider stub code as part of the software development, and maintain it under configuration management. Once all the stubs for SATM main have been provided, we test the main program as if it were a stand-alone unit. We could apply any of the appropriate functional and structural techniques, and look for faults. When we are convinced that the main program logic is correct, we gradually replace stubs with the actual code. Even this can be problematic. Would we replace all the stubs at once? If we did, we would have a "small bang" for units with a high outdegree. If we replace one stub at a time, we retest the main program once for each replaced stub. This means that, for the SATM main program example here, we would repeat its integration test eight times (once for each replaced stub, and once with all the stubs). ***Bottom-up Integration*** Bottom-up integration is a "mirror image" to the top-down order, with the difference that stubs are Replaced by driver modules that emulate units at the next level up in the tree. In bottom up integration we start with the leaves of the decomposition tree (units like ControlDoor and DispenseCash), and test them with specially coded drivers. There is probably less throw-away code in drivers than there is in stubs. Recall we had one stub for each child node in the decomposition tree. Most systems have a fairly high fan-out near at the leaves, so in the bottom-up integration order, we won't have as many drivers. This is partially offset by the fact that the driver modules will be more complicated.	
2.	Explain call graph based integration with the help of pair wise integration and neighborhood integration (10 M) One of the drawbacks of decomposition based integration is that the basis is the functional decomposition tree. If we use the call graph instead, we mitigate this deficiency; Call graph Is a directed, labeled graph , Vertices are methods , A directed edge joins calling vertex to the called vertex .Adjacency matrix is also used. **Pair Wise integration** The idea behind pair-wise integration is to eliminate the stub/driver development effort. Rather than Develop stubs and/or drivers, why not use the actual code? At first, this sounds like big bang integration, but we restrict a session to just a pair of units in the call graph. The end result is that we have one integration test session for each edge in the call graph (40 for the SATM call graph in Figure 4.2). This is not much of a reduction in sessions from either top-down or bottom-up (42 sessions), but it is a drastic reduction in stub/driver development.	



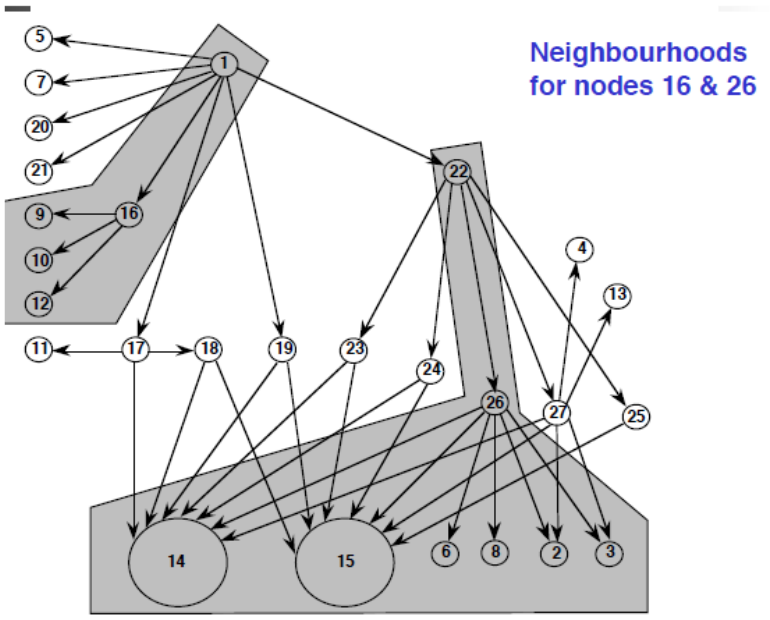
IntP–8

Neighborhood Integration

We define the neighborhood of a node in a graph to be the set of nodes that are one edge away from the given node. In a directed graph, this means all the immediate predecessor nodes and all the immediate successor nodes (notice that these correspond to the set of stubs and drivers of the node). The eleven neighborhoods for the SATM example (based on the call graph in Figure 4.2) are given in Table.

Table 3 SATM Neighborhoods **Node**

	Predecessors	**Successors**
16	1	9, 10, 12
17	1	11, 14, 18
18	17	14, 15
19	1	14, 15
23	22	14, 15
24	22	14, 15
26	22	14, 15, 6, 8, 2, 3
27	22	14, 15, 2, 3, 4, 13
25	22	15
22	1	23, 24, 26, 27, 25
1	n/a	5, 7, 2, 21, 16, 17, 19, 22



3. a) Briefly explain six basic principles of analysis and testing.(6 M)

- General engineering principles:
  - Partition: divide and conquer
  - Visibility: making information accessible
  - Feedback: tuning the development process
- Specific A&T principles:
  - Sensitivity: better to fail every time than sometimes
  - Redundancy: making intentions explicit
  - Restriction: making the problem easier

b) Write a note on software quality goals

- Process qualities (visibility)
- Product qualities
  - internal qualities (maintainability)
  - external qualities
    - usefulness qualities:
      - usability, performance, security, portability, interoperability
    - dependability
      - correctness, reliability, safety, robustness

4. Give the standard structure of analysis and test plan. (10 M)
A test and analysis plan may not address all aspects of software quality and testing activities. It should

indicate the features to be verified and those that are excluded from consideration (usually because responsibility for them is placed elsewhere). For example, if the item to be verified includes a graphical user interface, the test and analysis plan might state that it deals only with functional properties and not with usability, which is to be verified separately by a usability and human interface design
Team Explicit indication of features *not* to be tested, as well as those included in an analysis and test plan, is important for assessing completeness of the overall set of analysis and test activities. Assumption that a feature not considered in the current plan is covered at another point is a major cause of missing verification in large projects. The quality plan must clearly indicate criteria for deciding the success or failure of each planned activity, as well as the conditions for suspending and resuming analysis and test. Plans define items and documents that must be produced during verification. Test deliverables are particularly important for regression testing, certification, and process improvement
The core of an analysis and test plan is a detailed schedule of tasks. The schedule is usually illustrated with GANTT and PERT diagrams showing the relation among tasks as well as their relation to other project milestones.[1] The schedule includes the allocation of limited resources (particularly staff) and indicates responsibility for resources and responsibilities A quality plan document should also include an explicit risk plan with contingencies. As far as possible, contingencies should include unambiguous triggers (e.g., a date on which a contingency is activated if a particular task has not be completed) as well as recovery procedures. Finally, the test and analysis plan should indicate scaffolding, oracles, and any other software or hardware support required for test and analysis activities.

Analysis and test items:
The items to be tested or analyzed. The description of each item indicates version and installation procedures that may be required.
Features to be tested:
The features considered in the plan.
Features not to be tested:
Features not considered in the current plan.
Approach:
The overall analysis and test approach, sufficiently detailed to permit identification of the major test and analysis tasks and estimation of time and resources.
Pass/Fail criteria:
Rules that determine the status of an artifact subjected to analysis and test.
Suspension and resumption criteria:
Conditions to trigger suspension of test and analysis activities (e.g., an excessive failure rate) and conditions for restarting or resuming an activity.
Risks and contingencies:
Risks foreseen when designing the plan and a contingency plan for each of the identified risks.
Deliverables:
A list all A&T artifacts and documents that must be produced.
Task and schedule:
A complete description of analysis and test tasks, relations among them, and relations between A&T and development tasks, with resource allocation and constraints. A task schedule usually includes GANTT and PERT diagrams.
Staff and responsibilities:
Staff required for performing analysis and test activities, the required skills, and the allocation of responsibilities among groups and individuals. Allocation of resources to tasks is described in the schedule.
Environmental needs:
Hardware and software required to perform analysis or testing activities.

5. List out the integration faults and explain (6 M)

Integration fault	Example
Inconsistent interpretation of parameters or values Each module's interpretation may be reasonable, but they are incompatible.	Unit mismatch: A mix of metric and British measures (meters and yards) is believed to have led to loss of the Mars Climate Orbiter in September 1999.
Violations of value domains or of capacity or size limits Implicit assumptions on ranges of values or sizes.	Buffer overflow, in which an implicit (unchecked) capacity bound imposed by one module is violated by another, has become notorious as a security vulnerability. For example, some versions of the Apache 2 Web server between 2.0.35 and 2.0.50 could overflow a buffer while expanding environment variables during configuration file parsing.
Side-effects on parameters or resources	A module often uses resources that are not explicitly mentioned in its interface. Integration problems arise when these implicit effects of one module interfere with those of another. For example, using a temporary file "tmp" may be invisible until integration with another module that also attempts to use a temporary file "tmp" in the same directory of scratch files.
Missing or misunderstood functionality Underspecification of functionality may lead to incorrect assumptions about expected results.	Counting hits on Web sites may be done in many different ways: per unique IP address, per hit, including or excluding spiders, and so on. Problems arise if the interpretation assumed in the counting module differs from that of its clients.
Nonfunctional problems	Nonfunctional properties like performance are typically specified explicitly only when they are expected to be an issue. Even when performance is not explicitly specified, we expect that software provides results in a reasonable time. Interference between modules may reduce performance below an acceptable threshold.
Dynamic mismatches Many languages and frameworks allow for dynamic binding. Problems may be caused by failures in matchings when modules are integrated.	Polymorphic calls may be dynamically bound to incompatible methods, as discussed in Chapter 15.

b) What is a test and analysis report (2 M)

Reports of test and analysis results serve both developers and test designers. They identify open faults for developers and aid in scheduling fixes and revisions. They help test designers assess and refine their approach, for example, noting when some class of faults is escaping early test and analysis and showing up only in subsystem and system testing

6. Explain in detail dependable properties

Each quality explanation in detail (2 X 4= 8M)

Example : 2 M

Dependability Qualities:

- Correctness:
  - A program is correct if it is consistent with its specification

- seldom practical for non-trivial systems

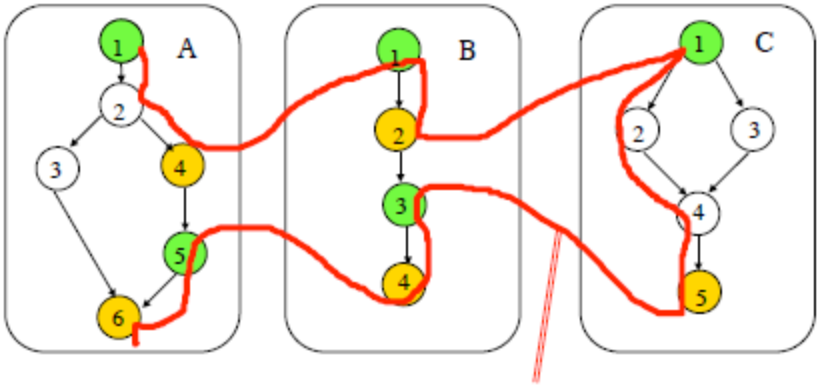
- Reliability:
  - likelihood of correct function for some ``unit" of behavior
    - relative to a specification and usage profile
    - statistical approximation to correctness (100% reliable = correct)
- Safety:
  - preventing hazards
- Robustness

  -acceptable (degraded) behavior under extreme conditions

7. a) Explain path based integration with example.

Path-Based Integration

- Focus on interactions among system units
- Rather than merely to test interfaces among separately developed and tested units



Source nodes

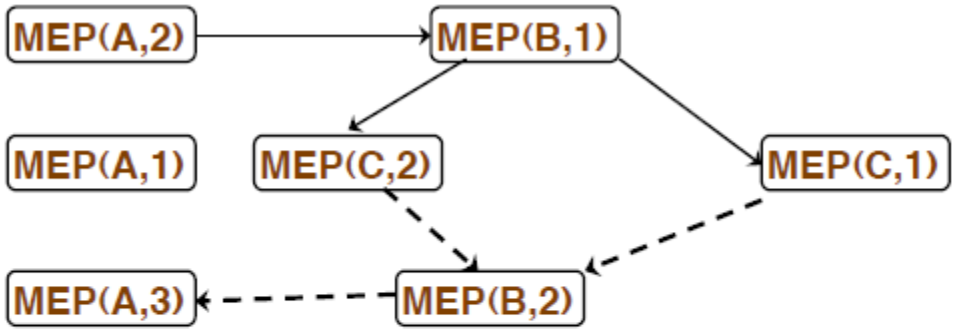
Sink nodes

Module Execution Paths

MEP(A,1) = <1, 2, 3, 6>
MEP(A,2) = <1, 2, 4>
MEP(A,3) = <5, 6>

MEP(B,1) = <1, 2>
MEP(B,2) = <3, 4>

MEP(C,1) = <1, 2, 4, 5>
MEP(C,2) = <1, 3, 4, 5>



Solid lines indicate messages (calls)
Dashed lines indicate returns from calls

b) Differentiate between manual inspection and automated analysis.

Manual inspection

- can be applied to essentially any document
  - requirements statements
  - architectural and detailed design documents
  - test plans and test cases
  - program source code
- may also have secondary benefits
  - spreading good practices
  - instilling shared standards of quality.
- takes a considerable amount of time
- re-inspecting a changed component can be expensive
- used primarily
  - where other techniques are inapplicable
  - where other techniques do not provide sufficient coverage

Automatic Static Analysis:

- More limited in applicability
  - can be applied to some formal representations of requirements models
  - not to natural language documents
- are selected when available
  - substituting machine cycles for human effort makes them particularly cost-effective.

8.	Define i) MM Path ii) MEP iii) Data quiescence iv) Message quiescence v) Port and give example for each **MM Path:** A module to module path! **An interleaved sequence of module execution paths and messages** Used to describes sequences of module execution paths that include transfers of control among separate units MM-paths always represent feasible execution paths, and these paths cross unit boundaries **MEP**: A sequence of statements within a module that! □ Begins with a source node □ Ends with a sink node □ With no intervening sink nodes **Message quiescence** Occurs when a unit that sends no messages is reached! **Data quiescence** Occurs when a sequence of processing ends in the creation of stored data that is not immediately used! The causal path Data A has no quiescence" The non-causal path D1 and D2 is quiescent at the node P-1"