Sample Question Paper

Software Testing (ETIT 414)

Q 1

i) What is functional testing?

This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

ii) Differentiate between alpha and beta testing.

Alpha testing - In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

Beta testing – Testing typically done by end-users or others. Final testing before releasing application for commercial purpose.

iii) What is the MAIN benefit of designing tests early in the life cycle?

It helps prevent defects from being introduced into the code.

iv) Given the following specification, which of the following values for age are in the SAME equivalence partition?

If you are less than 18, you are too young to be insured.
Between 18 and 30 inclusive, you will receive a 20% discount.
Anyone over 30 is not eligible for a discount.
a) 17, 18, 19
b) 29, 30, 31
c) 18, 29, 30
d) 17, 29, 31

Solution
The classes will be as follows:
Class I: age < 18 => not insured
Class II: age 18 to 30 => 20 % discount
Class III: age > 30 => no discount

Here we cannot determine if the above classes are valid or invalid, as nothing is mentioned in the question. (But according to our guess we can say I and II are valid and III is invalid. But this is not required here.) We have to select values which are in SAME equivalence partition. Values from option 'c' fall in same partition. **So answer is 'C'.**

v) What is the KEY difference between preventative and reactive approaches to testing?

Preventative tests are designed early; reactive tests are designed after the software has been produced.

vi) 13. Given the following fragment of code, how many tests are required for 100% decision coverage?

```
if width > length
then biggest_dimension = width
if height > width
    then biggest_dimension = height
end_if
else biggest_dimension = length
    if height > length
    then biggest_dimension = height
end_if
end_if
```

4

vii) Given the following code, which statement is true about the minimum number of test cases required for full statement and branch coverage?

```
Read p
Read q
IF p+q> 100
THEN Print "Large"
ENDIF
IF p > 50
THEN Print "p Large"
```

ENDIF

1 test for statement coverage, 2 for branch coverage

viii) **Which is the MOST important advantage of independence in testing?**

An independent tester may be more effective at finding defects missed by the person who wrote the software.

ix) **Consider the following techniques. Which are static and which are dynamic techniques?**

i. Equivalence Partitioning.

ii. Use Case Testing.

iii.Data Flow Analysis.

iv.Exploratory Testing.

v. Decision Testing.

vi. Inspections.

Data Flow Analysis and Inspections are static, Equivalence Partitioning, Use Case Testing, Exploratory Testing and Decision Testing are dynamic.

x) **Why are static testing and dynamic testing described as complementary?**

Because they share the aim of identifying defects but differ in the types of defect they find.

Q 2

a) What is component testing ?

Component testing, also known as unit, module and program testing, searches for defects in, and verifies the functioning of software (e.g. modules, programs, objects, classes, etc.) that are separately testable. Component testing may be done in isolation from the rest of the system depend-ing on the context of the development life cycle and the system. Most often stubs and drivers are used to replace the missing software and simulate the interface between the software components in a simple manner. A stub is called from the software component to be tested; a driver calls a component to be tested.

a) Given the following code, which statement is true about the minimum number of test cases required for full statement and branch coverage?

```
Read p
Read q
IF p+q> 100
THEN Print "Large"
ENDIF
IF p > 50
THEN Print "p Large"
ENDIF
```

1 test for statement coverage, 2 for branch coverage

b) **What is exploratory testing?**

Exploratory testing is a hands-on approach in which testers are involved in minimum planning and maximum test execution. The planning involves the cre-ation of a test charter, a short declaration of the scope of a short (1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used. The test design and test execution activities are performed in parallel typi-cally without formally documenting the test conditions, test cases or test scripts. This does not mean that other, more formal testing techniques will not be used. For example, the tester may decide to use boundary value analysis but will think through and test the most important boundary values without necessarily writing them down. Some notes will be written during the exploratory-testing session, so that a report can be produced afterwards.

Q 3

a) When do we prepare RTM (Requirement traceability matrix), is it before test case designing or after test case designing?

The would be before. Requirements should already be traceable from Review activities since you should have traceability in the Test Plan already. This question also would depend on the organisation. If the organisation do test after development started then requirements must be already traceable to their source. To make life simpler use a tool to manage requirements.

b) Why we use decision tables?

The techniques of equivalence partitioning and boundary value analysis are often applied to specific situations or inputs. However, if different combinations of inputs result in different actions being taken, this can be more difficult to show using equivalence partitioning and boundary value analysis, which tend to be more focused on the user interface. The other two specification-based tech-niques, decision tables and state transition testing are more focused on business logic or business rules. A decision table is a good way to deal with combinations of things (e.g. inputs). This technique is sometimes also referred to as a 'cause-effect' table. The reason for this is that there is an associated logic diagramming technique called 'cause-effect graphing' which was sometimes used to help derive the decision table.

Q 4

Write short notes on any two of the following:

a) Black Box Testing

The purpose of black-box testing is to devise a set of data inputs that fully exercise all functional requirements for a program. It is important to know that in black-box testing the test designer has no knowledge of algorithm implementation. The test cases are designed from the requirement statements directly, supplemented by the test designer's knowledge of defects that are likely to be present in modules of the type being tested.

It is also called *behavioral testing*, which focuses on the functional requirements of the software.

Black-box testing attempts to find errors in the following categories:

1. incorrect or missing functions
2. interface errors
3. errors in data structures or external database access
4. behavior or performance errors
5. initialization and termination errors

Black-box testing tends to be applied during the later stage of testing. Tests are designed to answer the following questions:

- How is functional validity tested?
- How is system behavior and performance tested?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?
- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation?

OOT—Test Case Design

Berard [BER93] proposes the following approach:

1. Each test case should be uniquely identified and should be explicitly associated with the class to be tested,
2. The purpose of the test should be stated,
3. A list of testing steps should be developed for each test and should contain [BER94]:
    a. a list of specified states for the object that is to be tested
    b. a list of messages and operations that will be exercised as a consequence of the test
    c. a list of exceptions that may occur as the object is tested
    d. a list of external conditions (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)
    e. supplementary information that will aid in understanding or implementing the test.

Testing Methods

Fault-based testing

- The tester looks for plausible faults (i.e., aspects of the implementation of the system that may result in defects). To determine whether these faults exist, test cases are designed to exercise the design or code.

Class Testing and the Class Hierarchy

- Inheritance does not obviate the need for thorough testing of all derived classes. In fact, it can actually complicate the testing process.

Scenario-Based Test Design

- Scenario-based testing concentrates on what the user does, not what the product does. This means capturing the tasks (via use-cases) that the user has to perform, then applying them and their variants as tests.

OOT Methods: Random Testing at the Class Level

Random testing

- identify operations applicable to a class
- define constraints on their use
- identify a minimum test sequence
    - an operation sequence that defines the minimum life history of the class (object)
- generate a variety of random (but valid) test sequences
    - exercise other (more complex) class instance life histories

OOT Methods: Partition Testing

Partition Testing

- reduces the number of test cases required to test a class in much the same way as equivalence partitioning for conventional software
- state-based partitioning
    - categorize and test operations based on their ability to change the state of a class
- attribute-based partitioning
    - categorize and test operations based on the attributes that they use
- category-based partitioning
    - categorize and test operations based on the generic function each performs

b) **Test Coverage**

Software test cases help make software testing efficient. These test cases consist of a series of steps and their expected results .

Test coverage measures in some specific way the amount of testing performed by a set of tests (derived in some other way, e.g. using specification-based techniques). Wherever we can count things and can tell whether or not each of those things has been tested by some test, then we can measure coverage.

c) **Regression Testing**

Regression testing is a style of testing that focuses on retesting after changes are made. In traditional regression testing, we reuse the same tests (the regression tests). In risk-oriented regression testing, we test the same areas as before, but we use different (increasingly complex) tests. Traditional regression tests are often partially automated. These note focus on traditional regression.

Regression testing attempts to mitigate two risks:

- A change that was intended to fix a bug failed.
- Some change had a side effect, unfixing an old bug or introducing a new bug.

In addition, proponents of traditional regression testing argue that retesting is a measurement or control process, a means of assuring that the program is as stable as it was previously.

Regression testing approaches differ in their focus. Common examples include:

- *Bug regression*: We retest a specific bug that has been allegedly fixed.
- *Old fix regression testing*: We retest several old bugs that were fixed, to see if they are back. (*This is the classical notion of* regression*: the program has* regressed *to a bad state.)*
- *General functional regression*: We retest the product broadly, including areas that worked before, to see whether more recent changes have destabilized working code. *(This is the typical scope of automated regression testing.*)
- *Conversion or port testing*: The program is ported to a new platform and a subset of the regression test suite is run to determine whether the port was successful. (Here, the main changes of interest might be in the new platform, rather than the modified old code.)

- *Configuration testing:* The program is run with a new device or on a new version of the operating system or in conjunction with a new application. This is like port testing except that the underlying code hasn't been changed--only the external components that the software under test must interact with.
- *Localization testing:* The program is modified to present its user interface in a different language and/or following a different set of cultural rules. Localization testing may involve several old tests (some of which have been modified to take into account the new language) along with several new (non-regression) tests.
- *Smoke testing* also known as *build verification testing*:A relatively small suite of tests is used to qualify a new build. Normally, the tester is asking whether any components are so obviously or badly broken that the build is not worth testing or some components are broken in obvious ways that suggest a corrupt build or some critical fixes that are the primary intent of the new build didn't work. The typical result of a failed smoke test is rejection of the build (testing of the build stops) not just a new set of bug reports.