4 UNIT

Software Quality Assurance and Testing

PART-1

Testing Objectives, Testing Principles, Test Plans, Test Cases, Types of Testing, Level of Testing, Testing Strategies.

CONCEPT OUTLINE : PART-1

- Software testing is the process of executing a program with the intention of finding errors in the code.
- Objectives of software testing are :
  a. Software quality improvement
  b. Verification and validation
  c. Software reliability estimation
- Some important principles of software testing are :
  a. All tests should be traceable to customer requirements.
  b. It is impossible to test everything.
  c. Use effective resources to test.
  d. Test should be planned long before testing begins.
- Test plan is a document describing the scope, approach, resources and schedule of intended testing activities.
- Levels of testing are :
  a. Unit testing
  b. Integration testing
  c. System testing
  d. Acceptance testing

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. **What is software testing ? What are the objectives of software testing ?**

Answer

1. Software testing is the process of executing a program with the intention of finding errors in the code.
2. It is a process of evolution of a system or its parts by manual or automatic means to verify that it is satisfying specified requirements or not.
3. Testing is the fundamental of software success.
4. Testing is not a distinct phase in system development life cycle but should be applicable throughout all phases *i.e.*, design development and maintenance phase.

Objectives of software testing : The software testing is usually performed for the following objectives :

1. **Software quality improvement :**
   a. Software quality means the conformance to the specified software design requirements.
   b. The minimum requirement of quality means performing as required under specified circumstances.
   c. Software testing is not only used to remove bugs, but also to find out design defect by the programmer.
2. **Verification and validation :**
   a. The second main objective of software testing is to verify the system and checking its validation.
   b. Verification means to test that we are building the product in right way *i.e.*, we are using the correct procedure for the development of software so that it can meet the user requirements.
   c. Whereas validation is the process which checks that whether we are building the right product or not.
3. **Software reliability estimation :**
   a. Software reliability has important relationship with many aspects of software development.
   b. Its objective is to discover the residual designing errors before delivery to the customer.
   c. The failure data during the testing process are taken down in order to estimate the software reliability.

Que 4.2. **State the principles of software testing.**

Answer

Principles of software testing :

1. All tests should be traceable to customer requirements.
2. Testing time and resources are limited *i.e.*, avoid redundant test.
3. It is impossible to test everything.
4. Use effective resources to test.
5. Test should be planned long before testing begins *i.e.*, after requirement phase.
6. Test for invalid and unexpected input conditions as well as valid conditions.
7. The probability of existence of more errors in a module or a group of modules is directly proportional to the number of errors already found.
8. Testing should begin "in the small" and progress towards testing "in the large".
9. For the most effective testing, testing must be conducted by an independent third party.

10. Testing should not be planned under the implicit assumption that no error will be found.

Que 4.3. Discuss the basic terms failure, test cases and test suite associated with testing.

Answer

Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.

1. **Test plan :** A test specification is called a test plan. A test plan is documented so that it can be used to verify and ensure that a product or system meets its design specification.
2. **Traceability matrix :** This is a table that correlates or design documents to test documents. This verifies that the test results are correct and is also used to change tests when the source documents are changed.
3. **Failure :** This is a manifestation of an error (or defect or bug). But, the mere presence of an error may not necessarily lead to a failure.
4. **Test case :** This is triplet [I, S, O], where I is the data input to the system, S is the state of the system at which the data is input, and O is the expected output of the system.
5. **Test data :** When multiple sets of values or data are used to test the same functionality of a particular feature in the test case, the test values and changeable environmental components are collected in separate files and stored as test data.
6. **Test scripts :** The test script is the combination of a test case, test procedure and test data.
7. **Test suite :** This is the set of all test cases with which a given software product is to be tested.

Que 4.4. What is the aim of software testing ?

Answer

1. The aim of the testing process is to identify all defects existing in a software product.
2. However, for most practical systems, even after satisfactorily carrying out the testing phase, it is not possible to guarantee that the software is error free.
3. This is because of the fact that the input data domain of most software products is very large.
4. It is not practical to test the software exhaustively with respect to each value that the input data may assume.

5. Even with this practical limitation of the testing process, the importance of testing should not be underestimated.
6. Testing provides a practical way of reducing defects in a system and increasing the user's confidence in a developed system.

Que 4.5. Define test plan and discuss various types of test plans.

Answer

Test plan is a document describing the scope, approach, resources, and schedule of intended testing activities. It identifies the test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. Several different general types of test plans are :

1. A mission plan tells "why". Usually, only one mission plan appears per organization or group. Mission plans describe the reason that the organization exists, are typically very short (5–10 pages), and are the least detailed of all plans.
2. A strategic plan tells "what" and "when". Again, only one strategic plan is usually used per organization, although some organizations develop a strategic plan for each category of project. Strategic plans are more detailed and longer than mission plans, sometimes 20-50 pages or more. They are seldom detailed enough to be directly useful to practicing testers or developers.
3. A tactical plan tells "how" and "who". Most organizations use one overall tactical plan per product. Tactical plans are the most detailed of all plans, and are usually living documents. For example, a tactical test plan would specify how each individual unit will be tested.

Que 4.6. What is the purpose of test plan and what are its contents ?

Answer

1. The main purpose of preparing a test plan is that everyone concerned with the project are in sync with regards to the scope, responsibilities, deadlines and deliverables for the project.
2. Purpose of preparing a test plan :
   a. A test plan is a useful way to think through the efforts needed to validate the acceptability of a software product.
   b. The completed document will help people outside the test group to understand the 'why' and 'how' of product validation.
   c. It should be thorough enough to be useful but not so thorough that no one outside the test group will read it.

Contents of a test plan are as follows :

1. **Purpose :** This section should contain the purpose of preparing the test plan.
2. **Scope :** This section should talk about the areas of the application which are to be tested by the QA team and specify those areas which are definitely out of scope (screens, database, mainframe processes etc).
3. **Test approach :** This would contain details on how the testing is to be performed and whether any specific strategy is to be followed (including configuration management).
4. **Entry criteria :** This section explains the various steps to be performed before the start of a test *i.e.*, pre-requisites.
5. **Resources :** This section should list out the people who would be involved in the project and their designation etc.
6. **Tasks/Responsibilities :** This section talks about the tasks to be performed and the responsibilities assigned to the various members in the project.
7. **Exit criteria :** Contains tasks like bringing down the system/server, restoring system to pre-test environment, database refresh etc.
8. **Schedules/Milestones :** This section deals with the final delivery date and the various milestone dates to be met in the course of the project.
9. **Hardware/Software requirements :** This section would contain the details of PC's/servers required to install the application or perform the testing.
10. **Risks and mitigation plans :** This section should list out all the possible risks that can arise during the testing and the mitigation plans that the QA team plans to implement the risk actually turns into a reality.
11. **Tools to be used :** This would list out the testing tools or utilities (if any) that are to be used in the project, example, WinRunner, Test Director, PCOM, WinSQL.
12. **Deliverables :** This section contains the various deliverables that are due to the client at various points of time *i.e.*, daily, weekly, start of the project, end of the project etc.
13. **References :**
    a. Procedures
    b. Templates (Client specific or otherwise)
    c. Standards/Guidelines, example, QView
    d. Project related documents (RSD, ADD, FSD etc).
14. **Annexure :** This could contain embedded documents or links to documents which have been/will be used in the course of testing, for example, templates used for reports, test cases etc.

15. **Sign-off :** This should contain the mutual agreement between the client and the QA team with both leads/managers signing off their agreement on the test plan.

Que 4.7. What should be the criteria for designing test cases ? Derive a set of test cases for the following : A sort routine which sorts arrays of integers.

Answer

1. A test case is a detailed procedure that fully tests a feature or an aspect of a feature whereas the test plan describes what is to be tested.
2. A test case describes how to perform a particular test.
3. We need to develop a test case for each test listed in the test plan or test specification.
4. Test cases should be written by someone who understands the function or technology being tested and should go through peer review.
5. Test cases include information such as the following :
   a. Purpose of the test.
   b. Special hardware requirements, such as modem.
   c. Special software requirements, such as a tool.
   d. Specific setup or configuration requirements.
   e. Description of how to perform the test.
   f. Expected results or success criteria for the test.

Test case for insertion sort :

```
Void sort(int array [ ], int len) {
int tempElem ;                  /* The element we are going to insert in
                                   the right place */
int tempElemIndex;              /*The index to the element we are about
                                   to move to its correct place */
int i ;                         /*A counter used to shift elements in the
                                   array to make room for tempElem */
if (NULL != array) { /* The array cannot be NULL */}
/* We are going to go through the entire array and insert each element
in its proper place in the already sorted portion of the array */
for (tempElemIndex = 0 ; tempElemIndex < len; tempElemIndex++)
{tempElem = array[tempElemIndex] ;
                i = tempElemIndex – 1 ;
                /* Go through the sorted portion of the array and if the
                element is larger than the element we are trying to insert,
                shift the larger element down one step.
                Repeat this until we find the right spot for the element we
                are inserting */
                while (i >= 0 && array[i] > tempElem) {
```

```
            array[i + 1] = array[i]
            i = i – 1 ;
            }
            /* Insert the element in its place in the stored portion of
            the array */
            array[i + 1] = tempElem ;
            }
    }
    return ;
    }
```

Que 4.8. **What do you mean by testing ? Describe type and levels of testing in brief. Also, discuss all testing strategies with suitable example and diagram.** **UPTU 2013-14, Marks 05**

OR

Discuss about types of testing and levels of software testing. **UPTU 2014-15, Marks 10**

Answer

Testing : Refer Q. 4.1, Page 112C, Unit-4.

Levels of testing : The relation of the faults introduced in different phases, and the different levels of testing is shown in Fig. 4.8.1.

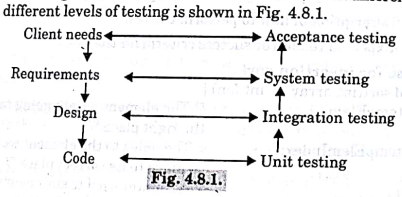


Fig. 4.8.1.

1. The basic levels are unit testing, integration testing, system testing and acceptance testing. These different levels of testing attempt to detect different types of faults.
2. The first level of testing is called unit testing. In this, different modules are tested against the specifications produced during design for the modules.

   This includes two types of testing :

   a. White box testing
   b. Black box testing
3. The next level of testing is often called integration testing. In this, many unit-tested modules are combined into subsystems, which are then tested. The goal here is to see if the modules can be integrated properly.

4. **Integration testing includes :**
   a. **Regression testing :** Change of behaviour due to modification or addition is called 'Regression'. It is used to bring changes from worst to least.
   b. **Incremental integration testing :** Checks out for bugs which encounter when a module has been integrated to the existing.
   c. **Smoke testing :** It is the battery of test which checks the basic functionality of program. If fails, then the program is not sent for further testing.
5. The next level of testing is system testing. Here, the entire software system is tested. The reference document for this process is the requirements document, and the goal is to see if the software meets its requirements.
6. **System testing includes :**
   a. **Recovery testing :** System is forced to fail and is checked out how well the system recovers the failure.
   b. **Security testing :** Checks the capability of system to defend itself from hostile attack on programs and data.
   c. **Performance testing :** Used to determine the processing speed.
   d. **Installation testing :** Installation and uninstallation is checked out in the target platform.
   e. **Load and stress testing :** The system is tested for maximum load and extreme stress points are figured out.
7. Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily.
8. **Acceptance testing includes :**
   a. **UAT (User acceptance testing) :** It ensures that the project satisfies the customer requirements.
   b. **Alpha testing :** It is the test done by the client at the developer's site.
   c. **Beta testing :** This is the test done by the end users at the client's site.
   d. **Long term testing :** Checks out for faults occurrence in a long term usage of the product.
   e. **Compatibility testing :** Determines how well the product is substantial to product transition.

Que 4.9. **What do you mean by unit/module testing ?**

Answer

1. Unit testing focuses verification effort on the smallest unit of software design, the software component or module.

2. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing.
3. The unit testing is white-box oriented, and the step can be conducted in parallel for multiple components.
4. In unit testing, individual components are tested to ensure that they are working properly in the same manner as required.
5. In this process, a module (software component) is taken and executed in isolation from the rest of software product. That's why, it is also called "software component testing".
6. It is the lowest level of testing of an application, under test software meets the requirements expected from it.
7. Unit testing is typically conducted by development team and programmer who coded the unit.
8. The main reasons for doing unit testing are :
   a. The size of a single module is small enough that can locate an error very easily.
   b. Due to small size, it can be tested in some demonstrably exhaustive fashion.
   c. Confusing interactions of multiple errors which occurs when many parts tested together are eliminated.

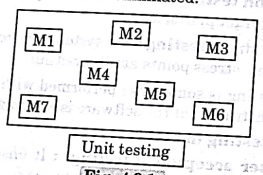


Fig. 4.9.1.

9. Unit testing becomes simple where modules are highly cohesive.
10. When module address less number of functions then testing is more easy and accurate as less number of test cases is required to test it.

Que 4.10. **What do you mean by software integration strategies ? Illustrate their importance.**

OR

Differentiate between top down, bottom up integration and big-bang strategies.

OR

Compare the relative merits and demerits of different integration testing strategies.

Answer

Integration testing :

1. Once individual program components have been tested, they must be integrated to create a partial or complete system.
2. Integration test should be developed during system specification, and this integration testing should begin as soon as usable versions of some of the system components are available.
3. The primary objective of integration testing is to check the modules interface *i.e.*, there are no errors in parameter passing, when one module invokes the functionality of another module.
4. After each integration step, the partially integrated system is tested.
5. This each step testing is done to avoid the errors causes in complex system during component connecting process.

The main integration techniques are as follows :

1. **Top-down integration testing :**
   a. In this approach, testing process starts with testing the top most module/component in the hierarchy and move downwards.
   b. This process continues until all components are integrated and then whole system has been completely tested.
   c. Top-down integrating testing approach requires the use of program stubs to simulate the effect of lower-level routines that are called by the routines under test.
   d. A pure top-down integration does not require any driver routines.

Advantages of top-down integration testing :

a. Design errors are detected as early as possible, saving development time and costs because corrections in the module design can be made before their implementation.
b. The characteristics of a software system are evident from the start, which enables a simple test of the development state and the acceptance by the user.

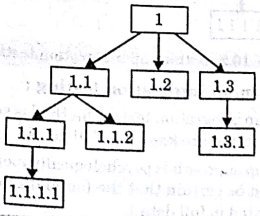


Fig. 1.10.1. Top-down integration.

c. The software system can be tested thoroughly from the start with test cases without providing (expensive) test environments.

Disadvantages of top-down integration testing :

a. Strict top-down testing proves extremely difficult because designing usable surrogate objects can prove very complicated, especially for complex operations.

b. Errors in lower hierarchy levels are hard to localize.

2. **Bottom-up integration testing :**

a. It is very popular approach of merging the components to test a larger system.

b. In this method, each subsystem (component) at the lower level of system hierarchy is tested individually first.

c. Then the next component who calls the previously tested ones to be tested.

d. This approach is followed repeatedly until all components are included in the testing.

e. The primary purpose to test each subsystem is to test interface among various models making up the subsystem.

f. In this, both control and data interface are tested.

g. In pure bottom-up testing, no stubs are required, and only test-drivers are required.

h. Bottom-up integration testing is mainly used when the performance and machine interface of the system is the main concern.

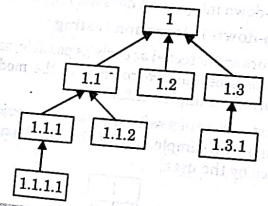


Fig. 1.10.2. Bottom-up integration testing.

Advantages of bottom-up integration testing :

a. The bottom-up integration testing method is solid and proven. The objects to be tested are known in full detail.

b. The bottom-up approach is psychologically more satisfying because the tester can be certain that the foundations for the test objects have been tested in full detail.

Disadvantages of bottom-up integration testing :

a. The characteristics of the finished product are only known after the completion of all implementation and testing, which means that design errors in the upper levels are detected very late.

b. Testing individual levels also inflicts high costs for providing a suitable test environment.

c. The complexity occurs when the system is made up of a large number of small subsystems that are at the same level.

3. **Big-bang testing :**

a. It is a non-incremental integration approach.

b. It is the simplest testing approach, where all modules making a complete system are integrated and tested as a whole and disorder usually gives unpredictable results.

c. The set of errors encountered here are very difficult to localize as it may potentially belong to any of the module being integrated.

d. And once these errors are corrected, new ones appear and the process continues in an endless loop.

e. Therefore, debugging errors reported during big-bang integration testing are very expensive to fix.

f. So, this technique can only be used for very small system because of difficulty in identifying the errors occurring in module interface for large system.

g. The main problem with big-bang integration is the difficulty of isolating the sources of error.

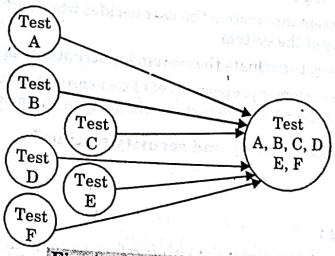


Fig. 4.10.3. Big-bang testing.

4. **Sandwich integration testing :**

a. The combination of top-down and bottom-up integration testing techniques is called sandwich integration.

b. This system is viewed as three layers just like a sandwich.

c. An upper layer of sandwich use top-down integration, the lower layer of sandwich integration use bottom-up integration, and the middle layer, that is, called target layer use the testing on the basis of system characteristics and structure of the component hierarchy.

d. It allows integration testing to begin early in the test process.

e. It contains advantage of top-down and bottom-up integration testing.

f. It is most commonly used integration testing approach.

Que 4.11. What is acceptance testing ? Why acceptance testing is needed ?

Answer

1. The purpose of acceptance testing is to confirm that the system meets its business requirements and to provide confidence that the system is working properly before it is "hand over" to the customer.
2. In this testing, customer is responsible for checking the performance of the system.
3. Acceptance testing is performed by nominated user representatives under guidance and supervision of testing team and developer.
4. The acceptance test is written, conducted and evaluated by customers, with assistance from the developers, only when the customer request an answer to a technical question.
5. During acceptance testing, it is very important that there should be an independent observer of testing process especially when the customers are not having too much IT (information technology) knowledge.
6. By doing acceptance testing the user decides whether to accept or reject the delivery of the system.
7. The main way to evaluate the system for acceptance is benchmark test.
8. In this, the customer prepares a set of test cases that represents typical condition under which the system will operate when actually installed.

Que 4.12. What is stress and security testing ?

Answer

Stress testing :

1. In software testing, stress testing refers to tests that determine the robustness of software by testing beyond the limits of normal operation.
2. Stress testing is particularly important for "mission critical" software, but is used for all types of software.
3. Stress tests commonly put a greater emphasis on robustness, availability, and error handling under a heavy load, than on what would be considered correct behaviour under normal circumstances.

4. Stress testing is the system testing of an integrated, black box application that attempts to cause failures involving how its performance varies under extreme but valid conditions (example, extreme utilization, insufficient memory, inadequate hardware, and dependency on over-utilized shared resources).
5. Stress testing typically involves the independent test team performing the following testing tasks using the following techniques :
   a. Test planning
   b. Test reuse
   c. Test design
      i. Use case based testing.
      ii. Workload analysis to determine the maximum production workloads.
   d. Test implementation
      i. Develop test scripts simulating extreme workloads.
   e. Test execution
      i. Regression testing
   f. Test reporting

Typical examples include stress testing as an application in :

1. Software only.
2. A system including software, hardware and data components.
3. Batch with no real time requirements.
4. Soft real time (*i.e.*, human reaction times).
5. Hard real time (for example, avionics, radar, automotive engine control).
6. Embedded within another system (for example, flight-control software, cruise-control software).

Security testing : Any computer based system that manages sensitive information or causes actions that can harm individuals is a target for improper or illegal penetration.

1. Security testing attempts to verify that protection mechanisms built into a system will protect it from improper penetration.
2. During security testing, the tester plays the role of the individual who desires to penetrate the system.
3. The tester may attempt to acquire passwords through external means, may attack the system with custom software designed to breakdown any defenses that have been constructed, may overwhelm the system, thereby denying services to others.
4. Good security testing will ultimately penetrate a system. The role of the system designer is to make penetration cost more than the value of information that will be obtained.

Que 4.13. **What is regression testing ? When we need regression testing ?**

Answer

1. Testing is used to find out errors in the software due to which it is not working properly. When these errors are found out then different methods are used to correct these errors.
2. Regression testing is used to identify these new errors or faults.
3. Regression testing identifies new faults that may have been introduced as correct ones.
4. A regression test is applied on new version or on release of software or its components to verify that it is still performing the same function in same manner.
5. Regression testing can be applied in development as well as maintenance phase of software life cycle. In development phase, regression testing is done after correcting the errors found during the testing of software.
6. While in maintenance phase of software life cycle, adaptive, corrective and preventive maintenance is done due to which some modification is done in the software and these modification may cause some new errors.
7. To find out these errors, regression testing is used in maintenance phase of software.
8. Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.
9. Capture/playback tools enables software engineer to capture test cases and results for subsequent playback and comparison.
10. A regression test suite (the subset of test to be executed) contains three different classes of test cases :
    a. A representative sample of tests that will exercise all software functions.
    b. Additional tests that focus on software functions are likely to be affected by the change.
    c. Test that focuses on the software components that have been changed.

Que 4.14. **What do you understand by test driver and test stub ? Explain in detail.**

OR

What are drivers and stub modules in context of integration and unit testing of software product ? Why are stubs and drivers modules required ?

Answer

Test driver :

1. A test driver is a software module or application used to invoke a test and provide test data, control and monitor execution and reports test outcomes.
2. Test driver are used for testing of sub-module in the absence of main control module.
3. A component driver routine calls a particular component and passes test case to it.
4. The driver may be written for a unit (module) or for integration (for combining the module).
5. The test drivers are not difficult to code since it rarely requires complex processing.

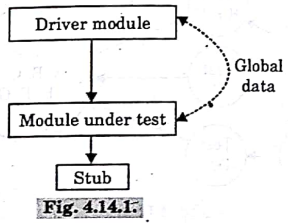


Fig. 4.14.1.

6. A driver module should contain the non-local data structures accessed by the module under test and should also have code to call the different functions of the module under test with appropriate parameter values for testing.
7. In case of bottom-up integration testing, the role of test driver can be easily understood by following example. Consider a component based hierarchy :

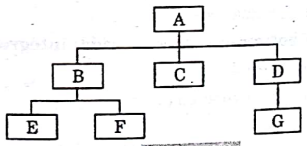


Fig. 4.14.2.

   a. To test the system bottom-up, we first test the lowest level *'E'*, *'F'*, *'G'* module as described above.
   b. Once these module is tested separately, we have to write a special code to aid integration *i.e.*, known as test driver.
   c. If each individual module *'E'*, *'F'*, and *'G'* are working correctly we move to next higher level.

d. Unlike the lowest level components, the next level components are not tested separately.

e. Instead they are combined with components they call (which have already tested).

f. In this case, we will test '*B*', '*E*' and '*F*' together.

g. If problem occurs we know that its cause is either in '*B*' or in the interface between '*B*' and '*E*' or '*B*' and '*F*'. Since '*E*' and '*F*' are functioning properly on their own.

h. If we try to test '*B*', '*E*' and '*F*' as a whole then we may not be easily able to isolate the problems caused.

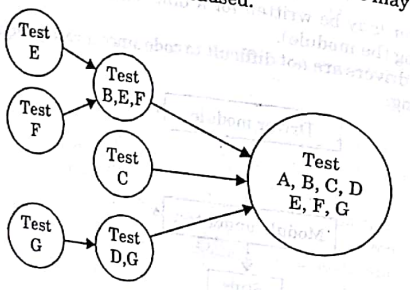


Fig. 4.14.3.

i. Similarly, we test '*D*' with '*G*'. Because '*C*' calls no other components we test it by itself.

j. Finally, we test all components together. Fig. 4.18.3 shows the sequence of tests and their dependencies.

Advantages of bottom-up (driver based) integration technique :

1. No test stub needed.
2. Errors in critical modules are found easily.

Disadvantages of bottom-up (driver based) integration technique :

1. More test driver needed.
2. Interface errors are discovered later.

Test stub :

1. Test stubs are specialized implementation of elements used for the testing purpose, which are dummy of a real component.
2. Test stubs are program or components that have deterministic behaviour and are used to interface with subsystem in order to take care of dependencies.
3. Basically, stubs are used in top-down approach.

4. In it, the main control module is tested in the absence of stub module.
5. For example, if we want to test a module '*A*' but it needs module '*B*', '*C*' and '*D*'. In this case, we create dummy modules for '*B*', '*C*' and '*D*'. These all are called stub and used to run '*A*'.

Advantages of top-down (stub based) integration technique :

1. No need of test driver.
2. Modular feature.

Disadvantage of top-down (stub based) integration technique :

1. It requires lots of stub writing.

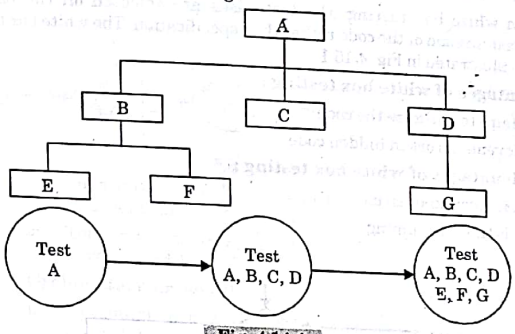


Fig. 4.14.4.

Que 4.15. What is white box testing ? List out advantages and disadvantages of white box testing.

Answer

1. White box testing is a software testing approach that examines the program structure and derives test data from the program logic.
2. In this approach, test group must have complete knowledge about the internal structure of the software.
3. Structural testing is usually applied to relatively small program units such as subroutine or operations associated with objects.
4. As the name implies, the tester can analyze the code and use knowledge about the structure of component to derive test data.
5. The white box testing allows to peep (preview) inside the box, it basically focuses on internal knowledge of the software to guide the selection of test data.
6. The white box testing is also known by name of glass box testing, structural testing, clear box testing, open-box testing, logic driven testing and path-oriented testing.

7. White box testing is a test case design method that uses the control structure of the procedural design methods to derive test cases.
8. Test cases can be derived that :
   a. Guarantee that all independent paths within a module have been exercise at least once.
   b. Exercise all logical decisions on their true and false sides.
   c. Execute all loops at their boundaries and within their operational bounds.
   d. Exercise internal data structures to assure their validity.
9. In white box testing, the test cases are selected on the basis of examination of the code rather than specification. The white box testing is illustrated in Fig. 4.15.1.

Advantages of white box testing :

1. Helps to optimize the code.
2. Reveals errors in hidden code.

Disadvantages of white box testing :

1. It is very expensive.
2. It is time consuming.

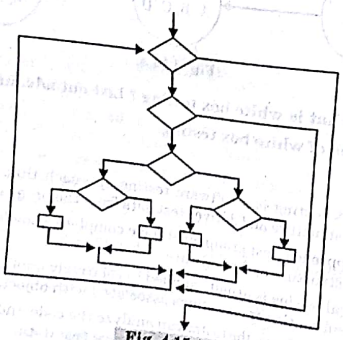


Fig. 4.15.1.

Que 4.16. **Explain white box testing techniques.**

OR

Explain basis path testing using flow graph analysis in detail.

Answer

The important white box techniques are as follows :

1. **Basis path testing :** This method enables the designer to derive a logical complexity measure of a procedural design and use it as a guide for defining a basis set of execution paths. Test cases that exercise the basis set are guaranteed to execute every statement in the program at least once during testing.
   a. **Flow graph notation :**
      i. A flow graph depicts logical flow control using the notation given in the Fig. 4.16.1.

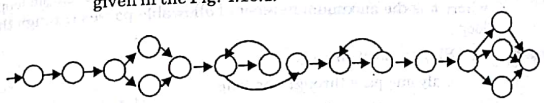


Fig. 4.16.1.

      ii. Each circle in a flow graph known as flow graph node represents one or more procedural statements.
      iii. The arrows on the flow graph, called edges or links, represent flow of control.
   b. **Cyclomatic complexity :**
      i. Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of a program.
      ii. It defines the number of independent paths in the basis set and thus provides an upper bound for the number of tests that must be performed.
      iii. $V(G) = 3$ (predicate nodes) $+ 1 = 4$.
2. **Condition testing :**
   i. Condition testing is a test case design method that is used for checking up logical conditions contained in a program module.
   ii. A simple condition is a boolean variable or a relational expression, which includes NOT (¬), EQUALTO (=), NOT EQUALTO (!=), LESS-THAN (<), LESS-THAN EQUALTO (<=), GREATER-THAN (>), GREATER-THAN EQUALTO (>=).
   iii. The compound statements are constructed with the help of OR (|), AND (&&) and NOT (>).
   iv. If a condition is incorrect, then at least one component of condition is incorrect.
   v. The condition testing method focuses on testing each condition in the program.

3. **Data flow testing :**
   i. Data flow testing focus on the points at which variables receives the values and the points at which these values are used.
   ii. This method selects test paths of a program according to the locations of definitions and uses of variables in the program.

4. **Loop testing :** Loops are the foundation stone for vast majority of all algorithms implemented in software. Hence, proper attention must be paid to loop testing while performing white box testing.

   Loops are classified as :

   a. **Simple loops :** The following tests can be applied to simple loops where n is the maximum number of allowable passes through the loop :
      i. Skip the loop entirely.
      ii. Only one pass through the loop.
      iii. m passes through the loop where $m < n$.
      iv. $n-1, n, n+1$ passes through the loop.

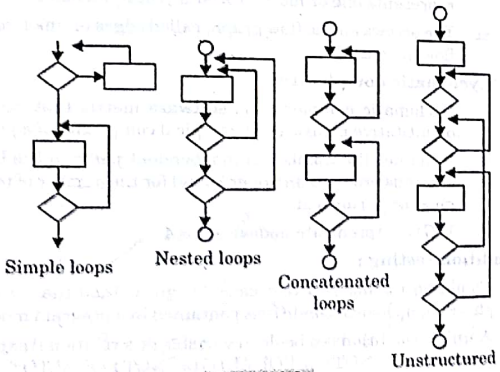


Fig. 4.16.2.

   b. **Nested loops :** The testing of nested loops cannot simply extend the technique of simple loops since this would results in a geometrically increasing number of test cases. The approach for nested loops that will help to reduce the number of test :
      i. Start at the innermost loop. Set all other loops to minimum values.
      ii. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter values. Add other tests for out-of-range or excluded values.

      iii. Work outward, conducting tests for the next loop while keeping all other outer loops at minimums and other nested loops to typical values.
      iv. Continue until all loops have been tested.

   c. **Concatenated loops :** Concatenated loops can be tested as simple loops, if each loop is independent of the other. If they are not independent (for example, the loop counter for one is the loop counter for the other), then the nested approach can be used.

   d. **Unstructured loops :** Whenever possible, this type of loops should be redesigned to reflect the use of the structured programming constructs.

Que 4.17. What is black box testing ? List out advantages and disadvantages of black box testing.

Answer

1. In black box testing, the tester don't know the internal structure of module, he tests only for input/output behaviour.
2. Black box testing focuses on the functional requirements of the software.
3. It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
4. Black box tests serve to detect deviations of a test object from its specification.
5. The selection of test cases is based on the specification of the test object without knowledge of its inner structure.
6. It is also important to design test cases that demonstrate the behaviour of the test object on erroneous input data.
7. The black box testing is also known by the name of functional testing, exterior testing, specification testing, data-driven testing and input/output driven testing.

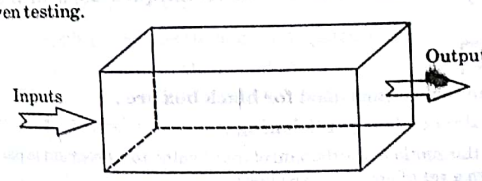


Fig. 4.17.1. Black box testing.

8. Test case should be derived which :
   a. Reduce the number of additional test cases that must be designed to achieve reasonable testing.
   b. Tell us something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

Tests are designed to answer the following questions :

1. How is the function's validity tested ?
2. What classes of input will make good test cases ?
3. Is the system particularly sensitive to certain input values ?
4. How are the boundaries of a data class isolated ?
5. What data rates and data volume can the system tolerate ?
6. What effect will specific combination of data have on system operation?

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

a. Incorrect functions
b. Missing functions
c. Interface errors
d. Data structure errors
e. External database access errors
f. Performance errors
g. Initialization and termination errors

Advantages of black box testing :

1. The test is unbiased because the designer and tester are independent of each other.
2. Test is done from the point of view of user not the designer.
3. Test cases can be designed as soon as specification is completed.
4. For tester, there is no need to know any programming language.

Disadvantages of black box testing :

1. The test can be redundant if software designer has already run a test case.
2. The test cases are difficult to design.

Que 4.18. Explain the different techniques used in black box testing.

Answer

Some of the techniques used for black box are :

1. Equivalence class partitioning :

a. In this method, the domain of input value to a program is partitioned into a set of equivalence classes.
b. Equivalence classes are usually formed for input domains and are classified into a valid equivalence class and one or more invalid equivalence classes.
c. The idea is to partition the input domain of the system into a finite number of equivalence classes such that each member of the class would behaves in a similar fashion, *i.e.*, if a test case in one class results some errors then the other member of class would also results same error.
d. The techniques increase the efficiency of software testing as the numbers of input states are drastically reduced.
e. This techniques involves two steps :
   i. Identification of equivalence classes.
   ii. Generating the test cases.

2. Boundary value analysis :

a. The boundary value analysis is refinement of equivalence class approach.
b. In boundary value analysis, we choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence classes.
c. Boundary values for each equivalence class, including the equivalence classes of the output, should be covered.
d. Boundary value test cases are also called "extreme cases".
e. Hence, we can say that a boundary value test case is a set of input data that lies on the edge of boundary of a class of input data or that generates output that lies at the boundary of a class of output data.
f. Guidelines for boundary value analysis are :
   i. If an input condition specifies a range bounded by values a and b, test cases should be designed with the values a and b and just above and just below a and b.
   ii. If an input condition specifies a number of input value, test cases should be developed that exercise the minimum and maximum number. Values just above and just below minimum and maximum are also tested.
   iii. Apply guidelines (ii) and (iii) for each output condition.

By applying above guidelines, the boundary testing will be more complete thereby having a higher likelihood for error detection.

3. Cause-effect graphing :

a. One weakness with the equivalence class partitioning and boundary value methods is that they consider each input separately.
b. That is, both concentrate on the conditions and classes of one input.
c. They do not consider combinations of input circumstances that may form interesting situations that should be tested.
d. One way to exercise combinations of different input conditions is to consider all valid combinations of the equivalence classes of input conditions.

e. This simple approach will result in an unusually large number of test cases, many of which will not be useful for revealing any new errors.

f. Cause-effect graphing is a technique that aids in selecting combinations of input conditions in a systematic way, such that the number of test cases does not become unmanageably large.

g. This technique starts with identifying causes and effects of the system under testing.

h. A cause is a distinct input condition, and an effect is a distinct output condition.

i. Each condition forms a node in the cause-effect graph.

j. The condition should be states such that they can be set to either true or false.

In steps, cause effect testing are as follows :

1. For a model identify input conditions (causes) and actions (effect).
2. Develop a cause-effect graph.
3. Transform cause-effect graph into a decision table.
4. Convert decision table rules to test cases. Each column of decision table represents a test case.

Que 4.19. Discuss the differences between black box and structural testing and suggest how they can be used together in the defect testing process ?

Answer

Refer Q. 4.15, Page 129C and Q. 4.17, Page 133C; Unit-4.

Que 4.20. Differentiate between alpha and beta testing.

Answer

1. When software is developed for a particular customer a series of acceptance test can be done to help the customer to validate his/her requirements but when the software is developed for multiple customers (example utility software) this series of testing is not feasible.
2. For this, two main types of testing are connected with acceptance testing *i.e.*, Alpha and Beta testing. These are referred as internal and external testing.

a. **Alpha testing :**

1. A lot of testing of software product is done to ensure about the system performance before delivering the software product to the customer but it is not possible to "predict" how the customer will really use program/module/system.

2. While operating manuals and user manual are explained and given to user, even then, there are possibility of misinterpretation of instruction, complex data combination that is not understandable to user, the output produced by system in user environment is not upto the mark though tester found it ok when he/she tested the system.
3. To solve this problem customer is called at developer site to test the system. This type of testing is called alpha testing.
4. In alpha testing, the software is used in its future environment on which it has to work on user site.
5. In alpha testing, the developer play a role of "invisible observer" he do not test anything, he/she only records the usage problems/errors that occurs while end user interact with system/software. It is done at developer's site so it is described as internal testing process.

b. **Beta testing :**

1. Beta testing of software is done at user site in "real-world environment".
2. This type of testing is done when developer is quite confident on the performance of their system and thinks it is ready for final delivery.
3. Its purpose is to validate that the software is ready to release to "real customers".
4. This testing is done out of developing environment of software so it is described as external testing.
5. Following points should be kept in mind while doing beta testing :
   i. "Know" the beta tester : It is very important to have knowledge about beta customer as whether they are experienced or novice (beginner) tester. As experienced user/ tester will not take interest in lower detail while it will be important for novice user.
   ii. Make sure that beta tester are testing the given system and providing appropriate report according to it.
   iii. It is very good way for checking the capability and configuration of the bugs especially for web based applications.
   iv. In beta testing only the superficial problems may get reported. A lot of efforts required to motivate the user to spend an adequate time on trying out the system.

Que 4.21. What is the difference between testing and debugging ?

Answer

The difference between testing and debugging are as follows :

S. No.	Testing	Debugging
1.	Starts with known conditions, user predefined procedures, predictable outcomes.	Unknown initial condition, end cannot be predicted.
2.	Should be planned, designed, and scheduled.	Cannot be constrained.
3.	Is a demonstration of errors/ apparent correctness.	Is a deductive process.
4.	Proves a programmer's "failure"	Vindicates a programmer.
5.	Should strive to be predictable, dull constrained, rigid, inhuman.	Demands intuitive leaps, conjectures, experimentations, freedom.
6.	Much can be done without design knowledge.	Impossible without design knowledge.
7.	Can be done by outsider.	Must be done by insider.
8.	Theory of testing is available.	Only recently have theorists started.
9.	Much of the test design and execution can be automated.	Automation is still a dream.

Que 4.22. Write short note on walkthrough.

Answer

Refer Q. 3.16, Page 105C, Unit-3.

Que 4.23. Write a short note on test strategies of software testing.

UPTU 2012-13, Marks 05

Answer

The purpose of test strategy is to clarify the major tasks and challenges of the test project. A good test strategy must be specific, practical and justified. The test strategy aims are as follows :

1. "Testing-in-the-small" and move toward "testing-in-the-large".
2. State testing objectives explicitly.
3. Understand the users of the software and develop a profile for each user category.

4. Develop a testing plan that emphasizes "rapid cycle testing".
5. Build "robust" software, that is, designed to test itself.
6. Use effective formal technical reviews as a filter prior to testing.
7. Conduct formal technical reviews to assess the test strategy and test cases themselves.
8. Develop a continuous improvement approach for the testing process.

PART-2

Program Correctness, Program Verification and Validation, Testing Automation and Testing Tools, Concept of Software Quality, Software Quality Attributes, Software Quality Metrics and Indicators.

CONCEPT OUTLINE : PART-2

- Software validation refers to a set of activities that ensure that the software, that is, going to be built is traceable to customer requirements.
- Software verification refers to a set of activities that ensures that software correctly implements a specific function.
- Software test automation refers to the activities and efforts that intend to automate engineering tasks, and operations in a software test process.
- Software quality refers to achieving high levels of user satisfaction, portability, maintainability, robustness and fitness for use.
- Software metrics are all about measurements which in turn, guide us how to make things better.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.24. What is software validation and verification ? Explain its objectives.

OR

Define verification and validation.

Answer

1. Requirements validation is concerned with showing that the requirements actually define the system that the customer wants.

2. It refers to a set of activities that ensure that the software, that is, going to be build is traceable to customer requirements, *i.e.*, "Are we building the right product".

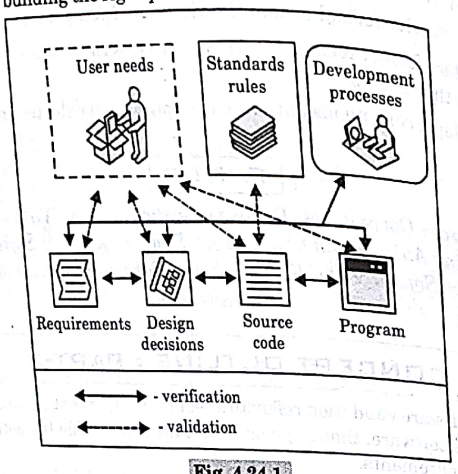


Fig. 4.24.1.

3. Whereas software verification refers to a set of activities that ensure that software correctly implements a specific function *i.e.*, "Are we building the product right".
4. Traditionally, software verification and validation is defined as system engineering methodology to ensure that quality is built into software during development.
5. The analysis and test activities are performed by V & V, evaluate and assess the software products and the development processes during each software life cycle phase in parallel with, not after the completion of, the development effort.
6. This provides early identification of errors, improving software performance, identification of program risks. V & V attacks on two of major contributor of software failure :
   a. incorrect or missing requirements, and
   b. poor organization in software architecture and failure to plan effectively.
7. It also acts as powerful risk management tool. In many cases, V & V is governed by standards establishing software development project management, and the software quality assurance requirements.

Objectives of verification and validation : Software V & V comprehensively analyzes and tests software during all stages of its development and maintenance in parallel with development process to :

1. Determine that it performs its intended functions correctly.
2. Ensure that it performs no unplanned functions.
3. Measure and assess the quality and reliability of software.

4. Hardware including all hardware directly or indirectly influenced by the software.
5. External software linked to the system.
6. Find errors in the software process and product as early as possible.
7. Assist in determining good requirements and design.
8. Ensure that quality is built into the software and that the software will satisfies the software requirements.
9. Satisfy the user that system is being developed according to standards and specifications.
10. Predict how well the interim products will result in a final product that satisfies the software requirement.

Que 4.25. Differentiate between validation and verification.

UPTU 2004-05, Marks 05

Answer

S. No.	Validation	Verification
1.	Am I building the right product ?	Am I building the product right ?
2.	Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs. It is traditional and is performed at the end of the project.	The review of interim work steps and interim deliverables during a project to ensure they are acceptable. To determine if the system is consistent, adheres to standards, uses reliable techniques and prudent practices, and performs the selected functions in the correct manner.
3.	Am I accessing the right data (in terms of the data required to satisfy the requirements).	Am I accessing the data right (in the right place; in the right way).
4.	High level activity.	Low level activity.
5.	Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment.	Performed during development on key artifacts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards.

6.	Determination of correctness of the final software product by a development project with respect to the user needs and require-ments.	Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

Que 4.26. **Define the term "software quality". Why we need to manage the quality ?**

OR

What is software quality ?

Answer

a. Now a days, quality is critical for survival and success. The quality of any product describes as "achieving high levels of user satisfaction, portability, maintainability, robustness and fitness for use".

b. Maintaining the quality of a product is not an easy task.

c. Software is now a global business and organizations will not succeed in the global market unless they produce quality products and services.

d. There are several reasons why business should be concerned with quality :

1. Quality is competitive issue now.
2. Quality is must for survival.
3. Quality gives you global reach.
4. Quality is cost effective.
5. Quality helps in retaining customers and increasing the profits.
6. Quality is a hallmark of world class business.

e. In software development, quality of designs encompasses requirements, specifications and the design of system.

f. Quality of conformance is an issue focused primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.

g. The quality plays important role for user satisfaction. It can be observed as,

User satisfaction = Complaint product + Good quality + Delivery within budget and schedule

h. The quality control or quality management involves a series of inspections, reviews and test used throughout the software process to ensure that each work product meets the requirements placed upon it.

i. Quality control includes a feedback loop to the process that created the work product.

j. The combination of measurement and feedback allows us to tune the process when the work product created fails to meet their specifications.

k. This approach views quality control as part of the manufacturing process.

l. A key concept of quality control is that all work product have defined, measurable specifications to which we may compare the output of each process.

m. Thus, at last we can describe software quality as "Conformance to explicitly stated functional and performance requirements, explicitly documented development standard and implicit characteristics that are expected from all professionally developed software".

Que 4.27. **What are software quality assurance attributes ?**

Answer

The software quality assurance attributes are as follows :

1. **Runtime system qualities :** It can be measured as the system executes :
   a. **Functionality :** The ability of the system to do the work for which it was intended.
   b. **Performance :** The response time, utilization, and throughput behaviour of the system. Not to be confused with human performance or system delivery time.
   c. **Security :** A measure of system's ability to resist unauthorized attempts at usage or behaviour modification, while still providing service to legitimate users.
   d. **Availability (reliability quality attributes falls under this category) :** The measure of time that the system is up and running correctly; the length of time between failures and the length of time needed to resume operation after a failure.
   e. **Usability :** The ability of two or more systems to cooperate at runtime.
2. **Non-runtime system qualities :** It cannot be measured as the system executes :
   a. **Modifiability :** The ease with which a software system can accommodate changes to its software.
   b. **Portability :** The ability of a system to run under different computing environments. The environment types can be either hardware or software, but is usually a combination of the two.
   c. **Integrability :** The ability to make the separately developed components of the system to work together in a correct way.
   d. **Testability :** The ease with which software can be made to demonstrate its faults.

3. **Business qualities :** Non-software system qualities that influence other quality attributes :
   a. **Cost and schedule :** The cost of the system with respect to time to market, expected project lifetime, and utilization of legacy.
   b. **Marketability :** The use of the system with respect to market competition.
   c. **Appropriateness for organization :** Availability of the human input, allocation of expertise, and alignment of team and software structure.
4. **Architecture qualities :** Quality attributes specific to the architecture itself.
   a. **Conceptual integrity :** The integrity of the overall structure that is composed from a number of small architectural structures.
   b. **Correctness :** Accountability for satisfying all requirements of the system.

Que 4.28. **What do you mean by software metrics ? Also, explain its application area.**

OR

Describe the characteristics of good software metric. Also, discuss various software metrics for project monitoring.

UPTU 2013-14, Marks 10

Answer

Software metrics :

1. Software metrics can be defined as "It is a continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products".
2. Software metrics are all about measurements which, in turn, guides us how to make things better, to improve the process of developing software and to improve all aspects of the management of that process.
3. Software metrics are applicable to the whole development life cycle from initiation, when costs must be estimated, to monitoring the reliability of the end product in the field, and the way that product changes over time with enhancement.
4. It covers the techniques for monitoring and controlling the process of the software development, such that the fact that it is going to be six months late, is recognized as early as possible rather than the day before delivery is due.

5. Some metrics belong to multiple categories like quality metric. It may belong to all three categories. It focuses on the quality aspects of the product process, and the project.
6. The prediction of quality levels for software, often in terms of reliability, is another area where software metrics have an important role to play.
7. The use of software metrics to provide quantitative checks on software design is also a well established area.
8. Much research has been carried out, and some organizations have used such techniques to very good effect.
9. This area of software metrics is also being used to control software products which are in place and are subject to enhancement.
10. Software metrics are also used to provide management information. This includes information about productivity, quality and process effectiveness.
11. It is important to realize that this should be seen as an ongoing activity.
12. Snapshots of the current situation have their place, but the most valuable information comes when we can see trends in data. Is productivity or quality getting better or worse overtime ? Why is this happening ? What can management do to improve things ?
13. Statistical analysis is a part of it, but the information must be presented in a way that managers find it useful, at the right time and for the right reasons.
14. All this shows that software metrics is a vast field and have wide variety of applications throughout the software life cycle.

Categories of metrics : There are three categories of software metrics which are given below :

1. **Product metrics :** It describes the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability etc.
2. **Process metrics :** It describes the effectiveness and quality of the processes that produce the software product. Examples are :
   a. Effort required in the process.
   b. Time to produce the product.
   c. Effectiveness of defect removal during development.
   d. Number of defects found during testing.
   e. Maturity of the process.
3. **Project metrics :** It describes the project characteristics and execution. Examples are :
   a. Number of software developers.
   b. Staffing pattern over the life cycle of the software.
   c. Cost and schedule.
   d. Productivity.

Areas of applications : The most established area of software metrics is cost and size estimation techniques. These are :

1. There are many proprietary packages in the market that provide estimates of software system size, cost to develop a system, and the duration of the development or enhancement of the project.
2. These packages are based on estimation models, like COCOMO-I, COCOMO-II, developed by Berry Boehm. Various techniques that do not require the use of readymade tools are also available.
3. There has been a great deal of research carried out in this area, and this research continues in all important software industries and other organizations.
4. One thing that does come across strongly from the results of this research work is that organizations cannot rely, solely, on the use of proprietary packages.

Que 4.29. **Describe lines of code (LOC), the size estimation method.**

Answer

1. Lines of code (LOC) was the first measurement attempted. It has the advantage of being easily recognizable, seen and therefore counted.
2. Although this may seem to be a simple metric that can be counted algorithmically, there is no general agreement about what constitutes a line of code.
3. Early users of lines of code did not include data declarations, comments, or any other lines that did not result in object-code.
4. Later users decided to include declarations and other unexecutable statements but still excluded comments and blank lines.
5. The reason for this shift is the recognition that modern code can have 50 % or more data statements and that bugs occur as often in such statements as in real code.
6. For example, in the function, if LOC is simply a count of the number of lines then it contains 18 LOC.

```
1.  int sort (int x[ ], int n)
2.  {
3.  int i, j, save, im ;
4.  /* This function sorts array x in ascending order*/
5.  if (n < 2) return 1 ;
6.  for (i = 2; i < = n; i++)
7.  {
8.  im = i - 1 ;
9.  for (i = 1; i < = im; i++)
```



```
10. if (x[i] < x[j])
11. {
12. save = x[i] ;
13. x[i] = x[j] ;
14. x[j] = save ;
15. }
16. }
17. return 0 ;
18. }
```

7. But most researchers agree that the LOC metric should not include comments or blank lines. Since these are really internal documentation and their presence or absence does not affect the functions of the program.
8. Moreover, comments and blank lines are not as difficult to construct as program lines. The inclusion of comments and blank lines in the count may encourage developers to introduce artificially many such lines in project development in order to create the illusion of high productivity, which is normally measured in LOC/PM (lines of code/person-month).
9. When comments and blank lines are ignored, the above program contains 17 LOC.
10. Line of code is defined as "A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line".
11. This specially includes all lines containing program header, declarations, and executable and non-executable statements.
12. LOC is language dependent. A line of assembler is not the same as a line of COBOL. They also reflect what the system is rather than what it does.

Advantage of LOC : Simple to measure.

Drawbacks of LOC :

1. It is defined on code. For example, it cannot measure the size of specification.
2. It characterize only one specific view of size, namely length, it takes no account of functionality or complexity.
3. Bad software design may cause excessive line of code.
4. It is language dependent.
5. Users cannot easily understand it.

Que 4.30. **Give Halstead's software science measures for :**

1. **Program length**

2. Program volume
3. Program level
4. Efforts
5. Language level

OR

Describe two metrics that have been used to measure programmer productivity. Comment briefly on the advantages and disadvantages of each of these metrics.

Answer

Halstead's software science measure :

Halstead's software science is an analytical technique to measure size, development effort and development cost of software products. Halstead uses a few primitive programs parameters to develop the expressive for the overall program length, potential minimum volume, actual volume, language level, effort and development time.

1. **Program length :** The length of the program is defined by Halstead's quantifies the total usage of all operators and operands in the program so length,

$$N = N_1 + N_2$$

where

N_1 = total usage of all of the operators appearing in that implementation

N_2 = total usage of all of the operands appearing in that implementation.

2. **Program volume :** This is for the size of any implementation of any algorithm. The number of bits required to encode the program measured is known as volume and given by,

$$V = N \log_2 n$$

where n represents different identifiers uniquely

and $n = n_1 + n_2$

then $V = N \log_2 (n_1 + n_2)$

3. **Program level :** It measures the level of abstraction provided by the programming language.

The program level is given by,

$$L = \frac{V^*}{L}$$

Here, V^* = Potential minimum volume

4. **Efforts or programming effort :** The effort required to implement the program and can be obtained as,

$$\text{Efforts } (E) = \frac{V}{L} = \frac{\text{Program volume}}{\text{Program level}}$$

But $$L = \frac{V^*}{V}$$

Then $$E = \frac{V.V}{V^*} = \frac{V^2}{V^*}$$

5. **Language level :** Language level implies that as higher the level of a language, the less effort it takes to develop a program using that language, *i.e.*, language level is inversely proportional to the effort to develop a program.

$$L \alpha \frac{1}{E}$$

6. **Potential volume :** It is a metric for denoting the corresponding parameters in an algorithm's shortest possible form. Neither operators nor operands can require repetition.

$$V^* = (n_1^* + n_2^*) \log_2 (n_1^* + n_2^*)$$

Advantages of Halstead metrics :

1. Do not require in-depth analysis of programming structure.
2. Predicts rate of error.
3. Predicts maintenance effort.
4. Useful in scheduling and reporting projects.
5. Measure overall quality of programs.
6. Simple to calculate.
7. Can be used for any programming language.
8. Numerous industry studies support the use of Halstead in predicting programming effort and mean number of programming bugs.

Disadvantages of Halstead metrics :

1. It depends on the completed code.
2. It has little or no use of predictive estimating model. But McCabe's model is more signed to application at the design level.

Que 4.31. Calculate Halstead's measure on factorial code given below :

```
int fact (int n) {
if (n == 0)
{
(return 1 ;)
}
else
{return n * fact (n – 1) ; } }
```

Answer

The table factorial of operand and operators is as follows :

1. Program length $N = N_1 + N_2 = 18 + 8 = 26$
2. Vocabulary of program $n = n_1 + n_2 = 10 + 3 = 13$

Operator	Occurrence	Operand	Occurrence
int	2	n	4
( )	4	fact	2
if	1	1	2
;	2		
==	1		
–	1		
return	2		
else	1		
*	1		
{ }	3		
$n_1 = 10$	$N_1 = 18$	$n_2 = 3$	$N_2 = 8$

3. Volume $V = N \log_2 n = 26 \log_2 13 = 96.211$
4. Estimated length $N' = n_1 \log_2 n_1 + n_2 \log_2 n_2 = 10 \log_2 10 + 3 \log_2 3$
$= 33.219 + 4.754 = 37.973$
5. Effort $E = \frac{n_1 N_2}{2n_2}(N \log_2 n) = \frac{10 \times 8}{2 \times 3}(26 \log_2 12)$
$= \frac{10 \times 8 \times 93.209}{6}$
$= 1242.787$

Que 4.32. Describe the function point based measures and metrics.

Answer

1. The collection of function point data has two primary motivations. One is the desire by managers to monitor levels of productivity, for example, number of function points achieved per working hour expended.
2. Another use of function points is in the estimation of software development cost which is the most important potential use of function point data.
3. Function point may be used to compute the following important metrics:

Productivity = FP / persons-months
Quality = Defects / FP
Cost = Rupees / FP
Documentation = Pages of documentation per FP

Thus, metrics are controversial and are not universally acceptable.

4. The five functional units are ranked according to their complexity, *i.e.*, Low, Average, or High, using a set of prescriptive standards.

5. Organizations that use FP methods develop criteria for determining whether a particular entry is Low, Average or High.
6. Nonetheless, the determination of complexity is somewhat subjective.
7. After classifying each of the five function types, the unadjusted function points (UFP) are calculated using predefined weights for each function type as given in Table 3.32.1.

Table 3.32.1.

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Outputs (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal Logical Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

Table 3.32.2.

Functional Units	Count	Complexity	Complexity Total	Functional Unit Totals
External Inputs (EIs)		Low × 3 = Average × 4 = High × 6 =		
External Outputs (EOs)		Low × 4 = Average × 5 = High × 7 =		
External Inquiries (EQs)		Low × 3 = Average × 4 = High × 6 =		
Internal Logical Files (ILFs)		Low × 7 = Average × 10 = High × 15 =		
External Interface Files (EIFs)		Low × 5 = Average × 7 = High × 10 =		
Total unadjusted function point count =				

8. The weighting factors are identified as per Table 3.32.1 for all functional units and multiplied with the functional units accordingly.

9. The procedure for the calculation of UFP in mathematical form is given below :

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} * W_{ij}$$

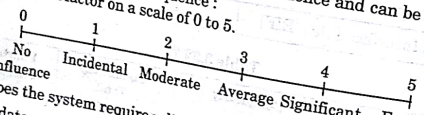
where i indicates the row and j indicates the column of Table 3.32.1.
W_{ij} : It is the entry of the i^{th} row and j^{th} column of the Table 3.32.1.
Z_{ij} : It is the count of the number of functional units of type i that have been classified as having the complexity corresponding to column j.

10. The final number of function points is arrived at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity.
11. In this, adjustment factor allows the UFP count to be modified by at most 35%. The final adjusted FP count is obtained by using the following relationship,

$$FP = UFP * CAF$$

where CAF is complexity adjustment factor and is equal to $[0.65 + 0.01 \times \Sigma F_i)$.

12. The F_i (i = 1 to 14) are the degrees of influence and can be calculated according to following sequence :
Rate each factor on a scale of 0 to 5.



1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing heavily utilized operational environment ?
6. Does the system require online data entry ?
7. Does the online data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated online ?
9. Is the inputs, outputs, files or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by user ?

Que 4.33. **Consider a project with the following parameters :**

1. **External inputs :**
   a. **10 with low complexity**
   b. **15 with average complexity**
   c. **17 with high complexity**
2. **External outputs :**
   a. **6 with low complexity**
   b. **13 with high complexity**
3. **External inquiries :**
   a. **3 with low complexity**
   b. **4 with average complexity**
   c. **2 with high complexity**
4. **Internal logical files :**
   a. **2 with average complexity**
   b. **1 with high complexity**
5. **External interface files :**
   a. **9 with low complexity**

In addition to above, system requires

1. **Significant data, communication**
2. **Performance is very critical**
3. **Designed code may be moderately reusable**
4. **System is not designed for multiple installations in different organizations.**

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Answer

To calculate the function point firstly, we have to calculate the unadjusted functional point counts as follows :

Functional Units	Count	Complexity	Complexity Total	Functional Unit totals
External inputs (ELs)	10	Low × 3 =	30	
	15	Average × 4 =	60	
	17	High × 6 =	102	192
External outputs (EOs)	6	Low × 4 =	24	
	0	Average × 5 =	0	
	13	High × 7 =	91	115

External inquiries (EQs)	3	Low × 3	=	9	37
	4	Average × 4	=	16	
	2	High × 6	=	12	
Internal logical files (ILFs)	0	Low × 7	=	0	35
	2	Average × 10	=	20	
	1	High × 15	=	15	
External interface files (EIFs)	9	Low × 5	=	45	45
	0	Average × 7	=	0	
	0	High × 10	=	0	
		Total unadjusted function point count =			424

The factors given previously can be calculated as :

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3 = 41$$

$$CAF = (0.65 + 0.01 * \Sigma F_i)$$
$$= (0.65 + 0.01 * 41)$$
$$= 1.06$$
$$FP = UFP * CAF$$
$$= 424 * 1.06$$
$$= 449.44$$

Que 4.34. What do you mean by cyclomatic complexity ? Explain.

OR

Describe the cyclomatic complexity measure and metrics.

Answer

1. The cyclomatic complexity is also known as structural complexity because it gives internal view of the code.
2. This approach is used to find the number of independent paths through a program. This provides us the upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once and every condition has been executed on its true and false side.
3. If a program has backward branch then it may have infinite number of paths. Although it is possible to define a set of algebraic expressions that gives the total number of possible paths through a program, however, using total number of paths has been found to be impractical.
4. Because of this, the complexity measure is defined in terms of independent paths that when taken in combination will generate every possible path.

5. An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
6. McCabe's cyclomatic metric, $V(G)$ of a graph G with n vertices, e edges, and P connected components is $V(G) = e - n + 2P$.
7. Given a program, we will associate with it a directed graph that has unique entry and exit nodes. Each node in the graph corresponds to a block of code in the program where the flow is sequential and the arcs correspond to branches taken in the program.
8. This graph is classically known as flow graph and it is assumed that each node can be reached by the entry node and each node can reach the exit node.
9. For example, a flow graph shown in Fig. 4.34.1 represents entry node '*a*' and exit node '*f*'.

The value of cyclomatic complexity can be calculated as :

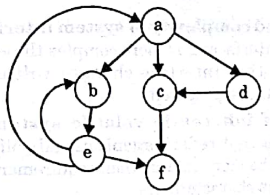


Fig. 4.34.1.

$$V(G) = 9 - 6 + 2 = 5$$

Here, $e = 9$, $n = 6$ and $P = 1$

There will be five independent paths for the flow graph illustrated in Fig. 4.34.1.

path 1 : a c f

path 2 : a b e f

path 3 : a d c f

path 4 : a b e a c f or a b e a b e f

path 5 : a b e b e f

Properties of cyclomatic complexity :

1. $V(G) >= 1$.
2. $V(G)$ is the maximum number of independent paths in graph G.
3. Inserting and deleting functional statements to G does not affect $V(G)$.
4. G has only one path if and only if $V(G) = 1$.
5. Inserting a new row in G increases $V(G)$ by unity.
6. $V(G)$ depends only on the decision structure of G.

Que 4.35. **Why it is difficult to validate the relationships between internal product attributes such as cyclomatic complexity and external attributes such as maintainability ? Discuss.**

UPTU 2012-13, Marks 10

Answer

1. Predicting the number of change requests for a system requires an understanding of the relationship between the system and its external environment.
2. Some systems have a very complex relationship with their external environment and changes to that environment inevitably result in changes to system.
3. To evaluate the relationships between system and its environment, we assess :
   a. **Number and complexity of system interfaces :** The larger the number of interfaces and more complex these interfaces, the more likely it is that interface changes will be required as new requirements are proposed.
   b. **Number of inherently volatile system requirements :** Requirement that reflect organizational policies and procedures are likely to be more volatile than requirements that are based on stable domain characteristics.
4. The relationship between program complexity, as measured by metrics such as cyclomatic complexity and maintainability is that the more complex a system or component, the more expensive it is to maintain.
5. Complexity measurements are particularly useful in identifying program components that are likely to be expensive to maintain.
6. To reduce maintenance costs, therefore, we should try to replace complex system components with simpler alternatives.
7. After a system has been put into service, we are able to use process data to predict maintainability.

PART-3

The SEI Capability Maturity Model (CMM), SQA Activities, Formal SQA Approaches : Proof of Correctness, Statistical Quality Assurance, Cleanroom Process.

CONCEPT OUTLINE : PART-3

- The Capability Maturity Model (CMM) is a strategy for improving the software process, irrespective of actual life cycle model used.

- Software quality assurance is defined as a planned and systematic approach to the evaluation of quality.
- SQA plan is a standard activity which ensures the quality goals of the organization.
- The cleanroom strategy is a software development process intended to produce software with a certifiable level of reliability.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.36. **What do you understand by ISO 9000 certificate ?**

Answer

ISO 9000 certification :

1. ISO 9000 certification serves as reference for contact between independent parties.
2. The ISO 9000 standard specifies the guidelines for maintaining a quality system.
3. The ISO standard mainly addresses operational aspects and organizational aspects such as responsibility, reporting etc.
4. In a nutshell, ISO 9000 specifies a set of guidelines for repeatable and high quality development.
5. It is important to realize that ISO 9000 standard is a set of guidelines for the production process and is not directly concerned with product itself.
6. **ISO 9000 is a series of three standards :** ISO 9001, ISO 9002 and ISO 9003. The different types of software industries to which the different ISO standard apply are as follows :
   a. **ISO 9001 :** This standard applies to the organizations engaged in design, development, production and servicing of goods. This is the standard that is applicable to most software development organizations.
   b. **ISO 9002 :** This standard applies to those organizations which do not design products but are only involved in production. Like category of industries include steel and car manufacturing industries who buy the products and plant design from external sources and only involved in manufacturing those products. Therefore, ISO 9002 is not applicable to software development organizations.
   c. **ISO 9003 :** This standard applies to organizations involved only in installation and testing of the product.

Benefits of ISO 9000 certification: Some basic benefits of ISO 9000 certification are as follows :

1. Continuous improvement : Each procedure and work instruction must be documented to become spring board for continuous improvement.
2. Eliminate verification.
3. Higher real and perceived quality.
4. Boost employee morale.
5. Improved customer satisfaction.
6. Increased employee participation.
7. Better product and services.
8. Greater quality assurance.
9. Improved profit levels.
10. Improved communication.
11. Reduced cost.
12. Competitive edge.

Limitations of ISO 9000 certification : Though ISO 9000 has proved to be very effective for software development organizations, however, it also suffers from several limitations.

1. ISO 9000 does not automatically lead to total quality management (TQM) *i.e.*, continuous improvement.
2. ISO 9000 does not provide any guideline for defining an appropriate process.
3. ISO 9000 certification process is not full proof and thus variation in the certification norms may exist.

Que 4.37. **Explain SEI capability maturity model.**

UPTU 2012-13, Marks 05

OR

Describe the term : SEI-CMM.

UPTU 2013-14, Marks 05

Answer

1. The capability maturity model (CMM) is not a software life cycle model. Instead, it is a strategy for improving the software process, irrespective of actual life cycle model used.
2. The CMM has been developed by Software Engineering Institute (SEI) at Carnegie Mellon University around 1987 and is still undergoing the process of refinement.
3. CMM is used to judge the maturity of the software processes of an organization and to identify the key practices that are required to increase the maturity of these processes.

4. The model is based on best practices followed in the organizations world wide. The five CMM level of maturity are :

i. **Level 1: Initial :**

a. At the level 1, the processes followed by the organizations are not well defined.

b. They are immature. As a result, a stable environment is not available for software development.

c. Further, success and failure of any project depends on the team member's competence.

d. There is no basis for predicting product quality, time for completion etc. No KPA's are defined at level 1.

ii. **Level 2 : Repeatable :**

a. The level 2 focuses on establishing basic project management policies. In this, experience with earlier projects is used for managing new projects of similar nature.

b. The managers are able to predict the cost and schedule of the project based on earlier experience. At this level, organization can be called a disciplined and organized organization.

c. The key process areas to achieve this level of maturity are :

1. **Requirement management :** Establishes common understanding of the project established between the developer and customer by documenting the requirements.
2. **Project planning :** Establishes plans for performing the software engineering and for managing the software project, by defining resources, goals, constraints etc.

Level 5.	3. Process change management 2. Technology and optimizing management 1. Defect prevention	**Optimizing**
Level 4.	2. Software quality management 1. Quantitative management	**Managed**
Level 3.	7. Peer reviews 6. Inter group coordination 5. Software product engineering 4. Integrated software management 3. Training program 2. Organization process definitions 1. Organization process focus	**Defined**
Leve 2.	6. Configuration management 5. Software quality assurance 4. Subcontract management 3. Software project tracking and oversight 2. Project planning 1. Requirements management	**Repeatable**
Level 1.	No KPA's	**Initial**

3. **Software project tracking and oversight :** Establishes people's visibility into actual progress of software projects.
4. **Subcontract management :** Focuses on selection of qualified software contractors and their effective management.
5. **Software quality assurance :** Provides adequate visibility into software project process, and the product being built by the management by following suitable quality standards.
6. **Configuration management :** Focuses on maintaining the integrity of all the products of the software project throughout the life cycle of the project.

iii. **Level 3 : defined :** At this level standard process to be used across the organization are defined and documented. Standardization of organization wide processes helps the manager and other team members to perform efficiently. Compared to level 2 defects are down 20 %, project cycle wise is down 10 %, the main features at level 3 are :

1. **Organization process focus :** Focuses on improvement of organization's overall software process capabilities.
2. **Training program :** Focuses on training for improving knowledge, skills and efficiency of individuals.
3. **Organization process definition :** Focuses on development and maintenance of organization standard process.
4. **Integrated software management :** Focuses on integration of software engineering and management activities into well defined coherent software process.
5. **Software product engineering :** Focuses on well defined engineering process and integrating all activities.
6. **Inter group coordination :** Focuses on planning interactions and technical interfaces between different groups.
7. **Peer reviews :** Focuses on removing the defects efficiently from the different software work products early in the life cycle.

iv. **Level 4 : Managed :**

a. At this level of CMM, quantitative quality goals are set for the organization for the software product as well as software processes.

b. Here quality and productivity of the process are measured and maintained by organization, wide software process database and feedback is given to management.

c. The key processes associated at management level are :

1. **Quantitative process management :** Focuses on controlling the software project process performance quantitatively.

2. **Software quality management :** Establishing strategies and plans for quantitative understanding of software project.

v. **Level 5 : Optimizing :** It focuses on continuous process improvement in organization using quantitative feedback. The main features at level 5 are :

1. **Defect prevention :** Focuses on identification of causes of defects and to prevent them from recurring in future projects by improving project defined process.
2. **Technology change management :** Focuses on identification of new technology and transferring them into an organization in systematic way to improve quality of product.
3. **Process change management :** Focuses on continuous improvement of software processes to improve productivity, quality and cycle time.

Que 4.38. Write a short note on SEI Capability Maturity Model (CMM). How does it differ from ISO 9000 ? UPTU 2014-15, Marks 10

Answer

SEI-CMM : Refer Q. 4.37, Page 158C, Unit-4.

ISO9000 v/s CMM :

S.No.	ISO	CMM
1.	ISO talks about minimum requirement for acceptable quality system.	CMM focuses on continuous software process improvement.
2.	ISO is a way to communicate the process.	CMM is a way to communicate capabilities.
3.	ISO9000 procedures describe a (possibly) definite development process but gives no indication of the likely quality of the designs or whether multiple software efforts are likely to produce software of similar quality.	The CMM is a very specific way of classifying an organizations software development methods. In a certain way, it tells how the quality of its software design is likely to be repeated.
4.	ISO basically used in manufacturing and production.	SEI CMM was developed specifically for software industry.

Que 4.39. **How would you define 'software quality' ? What do you mean by software quality assurance ? Also, discuss the various software quality activities.** **UPTU 2013-14, Marks 10**

Answer

Software quality : Refer Q. 4.26, Page 142C, Unit-4.

Software quality assurance :

1. To ensure that final product produced of high quality, some quality control activities must be performed throughout the development process.
2. If it is not done, correcting errors in the final stage can be very expensive, especially if they are originated in early phases.
3. The purpose of software quality assurance plan (SQAP) is to specify all the work products that are needed to be produced during the project, activities that need to be performed for checking the quality of each of work products and the tools and methods that may be used for the SQA activities.
4. Note that SQAP takes a broad view of quality. It is interested in the quality of not only final product but also the intermediate products ; even though in a project we are ultimately interested in the quality of the delivered product.
5. To ensure that the delivered software is of good quality, it is essential to make sure that the requirements and design are also of good quality.
6. For this reason, an SQAP will contain QA activities carried out throughout the project.
7. The SQAP specifies the tasks that need to be undertaken at different times in the life cycle to improve software quality and how they are to be managed.
8. These tasks generally include reviews and audits.
9. Each task should be defined with an entry and exit criterion, that is, the criterion that should be satisfied to initiate the task and the criterion that should be satisfied to terminate the task.
10. Both criteria should be stated so that they can be evaluated objectively. The responsibilities for different tasks should be identified.
11. The software quality assurance plan is developed during project planning and is reviewed by all interested parties.
12. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan.
13. The plan identifies :
    a. Evaluation to be performed.

    b. Audits and reviews to be performed.
    c. Standards those are applicable to the project.
    d. Procedures for error reporting and tracking.
    e. Documents to be produced by the SQA group.
    f. Amount of feedback provided to the software project team.

Major activities of SQA are :

1. **Application of technical methods :** Software quality is designed into the software but not added afterwards. It must have set of technical methods and tools to help analyst to generate high quality specifications and designs.
2. **Formal technical reviews :** A formal technical review is a meeting conducted by technical staff to uncover quality problems.
3. **Software testing :** Appropriate strategies for software testing should be applied.
4. **Enforcement of standards :** There are several quality standards such as ISO 9000, SEI CMMs, etc. meant for ensuring software quality. If these standards are used, they can be applied by developers as part of a formal review.
5. **Control of change :** Changes can introduce errors. Change control includes :
    a. Formalised requests for change.
    b. Evaluates the nature of the change.
    c. Controls the impact of the change.
6. **Measurement :** Software metrics are needed to track quality and assess impact of methodological procedural changes.
7. **Record keeping and reporting :** Collection and dissemination of software quality assurance information is required. Results of audits, reviews, change control, testing and other SQA activities are part of the historical record of the project.

Que 4.40. **Develop your own metrics for correctness, maintainability, integrity and usability of software. What is Statistical Quality Assurance (SQA) ?** **UPTU 2014-15, Marks 10**

Answer

Correctness : The extent to which a program satisfies its specification and fulfills the customer are the main objectives of correctness.

Maintainability : The effort required to learn, operate, prepare input and interpret output of a program.

Integrity : The extent to which access to software or data by unauthorized persons can be controlled.

Usability : The extent to which a program can be used in other applications also.

Statistical quality assurance :

Statistical quality assurance reflects a growing trend throughout industry to become more quantitative about quality. For software, statistical quality assurance implies the following steps :

1. Information about software errors and defects is collected and categorized.
2. An attempt is made to trace each error and defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, poor communication with the customer).
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the vital few).
4. Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects.

For example :

1. To illustrate the use of statistical methods for software engineering work, assume that a software engineering organization collects information on errors and defects for a period of one year.
2. Some of the errors are uncovered as software is being developed. Others (defects) are encountered after the software has been released to its end users.
3. Although hundreds of different problems are uncovered, all can be tracked to one (or more) of the following causes :
   a. Incomplete or erroneous specifications (IES)
   b. Misinterpretation of customer communication (MCC)
   c. Intentional deviation from specifications (IDS)
   d. Violation of programming standards (VPS)
   e. Error in data representation (EDR)
   f. Inconsistent component interface (ICI)
   g. Error in design logic (EDL)
   h. Incomplete or erroneous testing (IET)
   i. Inaccurate or incomplete documentation (IID)
   j. Error in programming language translation of design (PLT)
   k. Ambiguous or inconsistent human/computer interface (HCI)
   l. Miscellaneous (MIS)

Que 4.41. **What do you understand by the term SQA plans ?**

Answer

SQA plan is a standard activity which ensures the quality goals of the organization. It gives details and templates on all activities, which have become part of the standard implementation.

The SQA plan include the following :

1. Project planning
2. Models of data, classes and objects, processes, design architecture
3. SRS
4. Test plans for testing SRS
5. User manuals online help etc.
6. Audit and review

In SQA plan, mainly three kinds of activities are performed :

1. Organization specific
2. Software specific
3. Customer specific

Que 4.42. **Explain the following formal SQA approaches :**

1. **Proof of correctness**
2. **Statistical quality assurance**
3. **Cleanroom process**

OR

Describe the term cleanroom process. **UPTU 2013-14, Marks 05**

Answer

1. Proof of correctness :

A proof of correctness is a mathematical proof that a computer program or a part thereof will, when executed, yield correct results *i.e.*, results fulfilling specific requirements. Before proving a program correct, the theorem to be proved must, of course, be formulated.

1. Treat a program as a mathematical object.
2. Developed with a language with a rigorous syntax.
3. Are attempts at developing a rigorous approach to specification of software requirements.
4. With both can attempt to develop a mathematical proof that a program conforms exactly to its specification.
5. In the code, can at selected statements formulate assertions on the set of correct values for program variables.
6. Can then show that the statements between these assertions do the correct transformation of the values in the assertions.

2. **Statistical quality assurance :** Refer Q. 4.40, Page 163C, Unit-4.

3. **Cleanroom process :**

a. The cleanroom software engineering process is a software development process intended to produce software with a certifiable level of reliability.

b. The cleanroom process was originally developed by Harlan Mills and several of his colleagues including Alan Henver at IBM.

c. The focus of the cleanroom process is on defect prevention, rather than defect removal.

d. The basic principles of the cleanroom process are :

1. **Software development based on formal methods :** Cleanroom development makes use of the box structure method to specify and design a software product. Verification that the design correctly implements the specification is performed through team review.

2. **Incremental implementation under statistical quality control :**

a. Cleanroom development uses an iterative approach, in which the product is developed in increments that gradually increase the implemented functionality.

b. The quality of each increment is measured against pre-established standards to verify that the development process is proceeding acceptably.

c. A failure to meet quality standards results in the cessation of testing for the current increment, and a return to the design phase.

3. **Statistically sound testing :**

i. Software testing in the cleanroom process is carried out as a statistical experiment.

ii. Based on the formal specification, a representative subset of software input/output trajectories is selected and tested.

iii. This sample is then statistically analyzed to produce an estimate of the reliability of the software, and a level of confidence in that estimate.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What is software testing ? Explain testing objectives and principle.

Ans: Refer Q. 4.1 and Q. 4.2.

Q. 2. Discuss test plans and test cases.

Ans: Refer Q.4.5 and 4.7.

Q. 3. What are various levels of testing and types of testing ?

Ans: Refer Q.4.8.

Q. 4. Write a short note on test strategies.

Ans: Refer Q. 4.23.

Q. 5. Explain the terms "software verification" and "software validation".

Ans: Refer Q. 4.24.

Q. 6. Define the term :

a. Software quality
b. Software quality assurance
c. Software quality metrics

Ans:

a. Refer Q. 4.26.
b. Refer Q. 4.40.
c. Refer Q. 4.28.

Q. 7. Write a note on CMM model.

Ans: Refer Q. 4.37.

Q. 8. Explain cleanroom process.

Ans: Refer Q. 4.42

☺☺☺