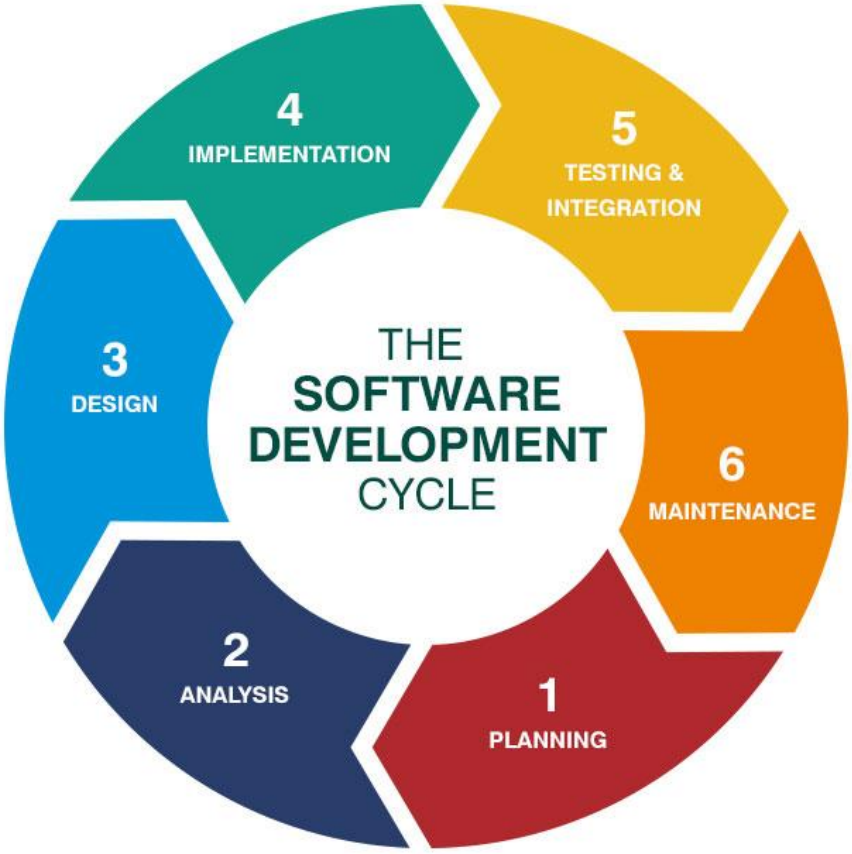
STQA CAT1 QB – Answers

1. a) What is SDLC? Describe in detail.

Ans 1. a)

SDLC stands for "software Development life cycle" it describes the sequence of phases pr steps to develop any software.
In simple words, we can say "entire lifetime of software from beginning to ending".



Stage 1: Planning and Requirement Analysis
Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements
Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture
SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product
In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product
This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance
Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

1. b) Difference between Verification and Validation.

Ans 1. b)

Verification	Validation
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does not include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

2. a) Explain in detail about the following terminologies in testing: Error, Fault, Failure.

Ans 2. a)

Error

The Problem in code leads to errors, which means that a mistake can occur due to the developer's coding error as the developer misunderstood the requirement or the requirement was not defined correctly.
The developers use the term error.

Fault

The fault may occur in software because it has not added the code for fault tolerance, making an application act up.

A fault may happen in a program because of the following reasons:

- **Lack of resources**
- **An invalid step**
- **Inappropriate data definition**

Failure

Many defects lead to the software's failure, which means that a loss specifies a fatal issue in software/ application or in its module, which makes the system unresponsive or broken.

In other words, we can say that if an end-user detects an issue in the product, then that particular issue is called a failure.

Possibilities are there one defect that might lead to one failure or several failures.

For example, in a bank application if the Amount Transfer module is not working for end-users when the end-user tries to transfer money, submit button is not working. Hence, this is a failure.

2. b) What is test case? Design the test case to for the scenario: "Check Login Functionality"

Ans 2. b)

A Test Case is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement.

The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

Test scenarios are rather vague and cover a wide range of possibilities. Testing is all about being very specific.

For a Test Scenario: Check Login Functionality there many possible test cases are:

Test Case 1: Check results on entering valid User Id & Password
Test Case 2: Check results on entering Invalid User ID & Password
Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

The format of Standard Test Cases
Below is a format of a standard login Test cases example.

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results
TU01	Check Customer Login with valid Data	1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit	Userid = guru99 Password = pass99	User should Login into an application
TU02	Check Customer Login with invalid Data	1. Go to site http://demo.guru99.com 2. Enter UserId 3. Enter Password 4. Click Submit	Userid = guru99 Password = glass99	User should not Login into an application

3. a) What are the different practices for designing good test case? Explain in detail.

Ans 3. a)

Best practices to write a test case :

Test cases which are easy to execute are considered as good test cases. They make the testing process more efficient by saving time and effort.

We can achieve this by following these 10 best practices :

1) Keeping it Simple and Easy to Understand

A good test case is the one which is easy to understand and execute for the testers. To be a good test case, it should be simple and organized category-wise. Different grouping techniques could be splitting the test cases based on user story or modules like browser specific behaviours etc. This makes it easy to review and maintain. Information given in the test cases should be clear to testers, developers and other stakeholders involved in the project.

2) Including End User Perspective

The end user perspective is a key element when it comes to maintaining the quality of software. Therefore, before drafting a test case, it's important to think like an end user. Understanding the requirement and the functionality aspects covered by the end user's perspective will help in identifying test scenarios that arise in real life business conditions.

3) Using Correct Naming Conventions

Naming the test cases in a way that makes it easy for stakeholders to identify as it will help create good and readable test cases. These also help in traceability as per requirements.

4) Providing Test Case Description

A proper test case description will allow users to understand what is being tested and how. Relevant details like the test environment, test data and tools to be used should also be provided.

5) Including Assumptions

All the assumptions and preconditions that are applicable to the test case should be specified clearly in the test case document.

6) Providing Steps Involved

Steps involved to test a test case should be clearly specified so that if any other person performs the test, they would be clear about what all steps to follow.

7) Giving Details of Test Data

The details of the test data for execution of the test case especially in cases where the same data can be reused helps in saving time for the creation of the test data for each

cycle to be run. Aim for maximum coverage by choosing a few select values from each equivalence class.

8) Making it Reusable
Ensuring that there is no dependency or conflict among test cases helps in making it modular. In case there are test cases that are inter-dependent, it should be clearly mentioned in the test document.

9) Assign Testing Priority
Priority of different features in an application is different and therefore assigning a testing priority ensures that during execution, high priority test cases are executed first.

10) Provide The Expected Result and Post Conditions

Expected Results and post conditions help in deciding whether a test case is passed or failed as per the user's acceptance so these should be clearly mentioned in test cases.

The entire testing process becomes more effective when proper time and efforts are invested in creating the test cases thus ensuring the success of the testing plan for the software project!

3. b) What do you mean by Test Case Management? Enlist any two Test Case Management Tools.

Ans 3. b)
Test case management is the process of managing testing activities to ensure high-quality and end-to-end testing of software applications.

To deliver a high-quality software application, the method entails organizing, controlling, and ensuring traceability and visibility of the testing process. In addition, it ensures that the software testing process is continuous as per your plan.
Here are several test managements tools such as TestRail, PractiTest, QTest, Zephyr, etc. and are good for writing test cases.

4. a) What are Test Oracles? Explain in detail.

Ans 4. a)

Test Oracle is a mechanism, different from the program itself, that can be used to test the accuracy of a program's output for test cases. Conceptually, we can consider testing a process in which test cases are given for testing and the program under test. The output of the two then compares to determine whether the program behaves correctly for test cases. This is shown in figure.

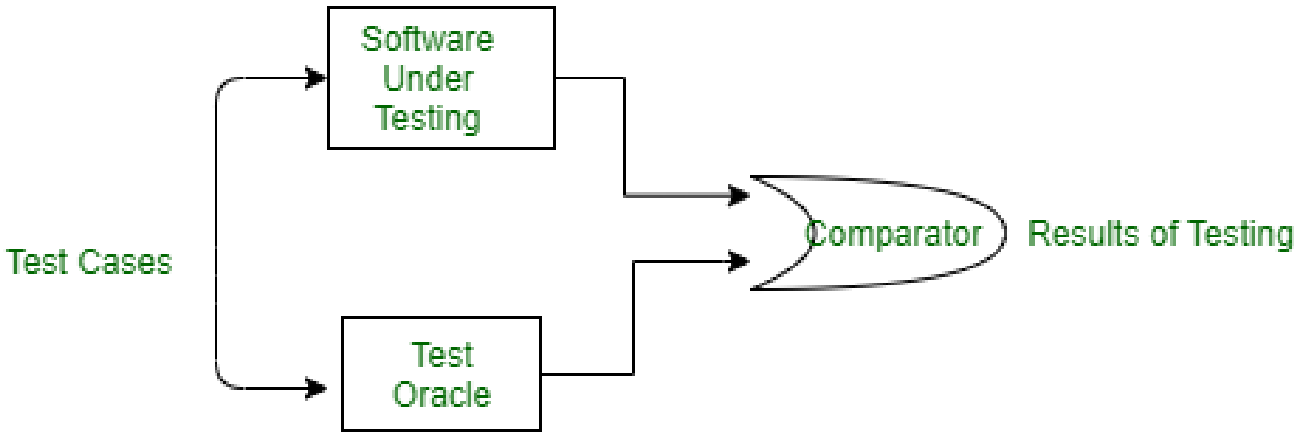


Figure - Testing and Test Oracles

Testing oracles are required for testing. Ideally, we want an automated oracle, which always gives the correct answer. However, often oracles are human beings, who mostly calculate by hand what the output of the program should be. As it is often very difficult to determine whether the behaviour corresponds to the expected behaviour, our "human deities" may make mistakes. Consequently, when there is a discrepancy, between the program and the result, we must verify the result produced by the oracle before declaring that there is a defect in the result.

The human oracles typically use the program's specifications to decide what the correct behaviour of the program should be. To help oracle determine the correct behaviour, it is important that the behaviour of the system or component is explicitly specified and the specification itself be error-free. In other words, actually specify the true and correct behaviour.

There are some systems where oracles are automatically generated from the specifications of programs or modules. With such oracles, we are assured that the output of the oracle conforms to the specifications. However, even this approach does not solve all our problems, as there is a possibility of errors in specifications. As a result, a divine generated from the specifications will correct the result if the specifications are correct, and this specification will not be reliable in case of errors. In addition, systems that generate oracles from specifications require formal specifications, which are often not generated during design.

4. b) What is Test Case Scenario? Explain in detail with suitable example.

Ans 4. b)

A Test Scenario is defined as any functionality that can be tested. It is also called Test Condition or Test Possibility. As a tester, you should put yourself in the end user's shoes and figure out the real-world scenarios and use cases of the Application Under Test.

Scenario Testing

Scenario Testing in software testing is a method in which actual scenarios are used for testing the software application instead of test cases. The purpose of scenario testing is to test end to end scenarios for a specific complex problem of the software. Scenarios help in an easier way to test and evaluate end to end complicated problems.

Why create Test Scenarios?

Test Scenarios are created for the following reasons,

- **Creating Test Scenarios ensures complete Test Coverage**
- **Test Scenarios can be approved by various stakeholders like Business Analyst, Developers, Customers to ensure the Application Under Test is thoroughly tested. It ensures that the software is working for the most common use cases.**
- **They serve as a quick tool to determine the testing work effort and accordingly create a proposal for the client or organize the workforce.**
- **They help determine the most important end-to-end transactions or the real use of the software applications.**
- **For studying the end-to-end functioning of the program, Test Scenario is critical.**

When not create Test Scenario?

- **Test Scenarios may not be created when**
- **The Application Under Test is complicated, unstable and there is a time crunch in the project.**
- **Projects that follow Agile Methodology like Scrum, Kanban may not create Test Scenarios.**
- **Test Scenario may not be created for a new bug fix or Regression Testing. In such cases, Test Scenarios must be already heavily documented in the previous test cycles. This is especially true for Maintenance projects.**

5. a) What is boundary value analysis? Explain in detail with suitable example.

Ans 5. a)

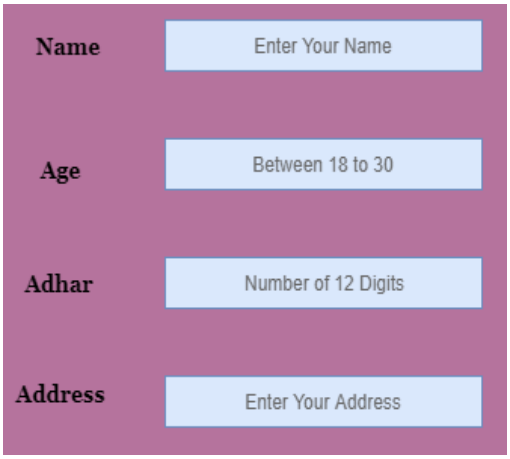
Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable.

Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.



There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

Example:

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

5. b) What is Equivalence class partitioning? Explain in detail with suitable example.

Ans 5. b)

Equivalence class partitioning is a black-box testing technique or specification-based testing technique in which we group the input data into logical partitions called equivalence classes.

All the data items lying in an equivalence class are assumed to be processed in the same way by the software application to be tested when passed as input.

So, instead of testing all the combinations of input test data, we can pick and pass any of the test data from a particular equivalence class to the application and assume that the application will behave in the same way for the other test data of that class. Let's understand this with the help of an example.

Example:

Consider an example of an application that accepts a numeric number as input with a value between 10 to 100 and finds its square. Now, using equivalence class testing, we can create the following equivalence classes-

Equivalence Class	Explanation
Numbers 10 to 100	This class will include test data for a positive scenario.
Numbers 0 to 9	This class will include test data that is restricted by the application. Since it is designed to work with numbers 10 to 100 only.
Greater than 100	This class will again include test data that is restricted by the application but this time to test the upper limit.
Negative numbers	Since negative numbers can be treated in a different way so, we will create a different class for negative numbers in order to check the robustness of the application.
Alphabets	This class will be used to test the robustness of the application with non-numeric characters.
Special characters	Just like the equivalence class for alphabets, we can have a separate equivalence class for special characters.

6. a) Differentiate between black box testing and white box testing.

Ans 6. a)

S. No.	Black Box Testing	White Box Testing
1.	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behaviour testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.
13.	Example: Search something on Google by using keywords	Example: By input to check and verify loops

6. b) What is state transition testing? Explain in detail with suitable example.

Ans 6. b)

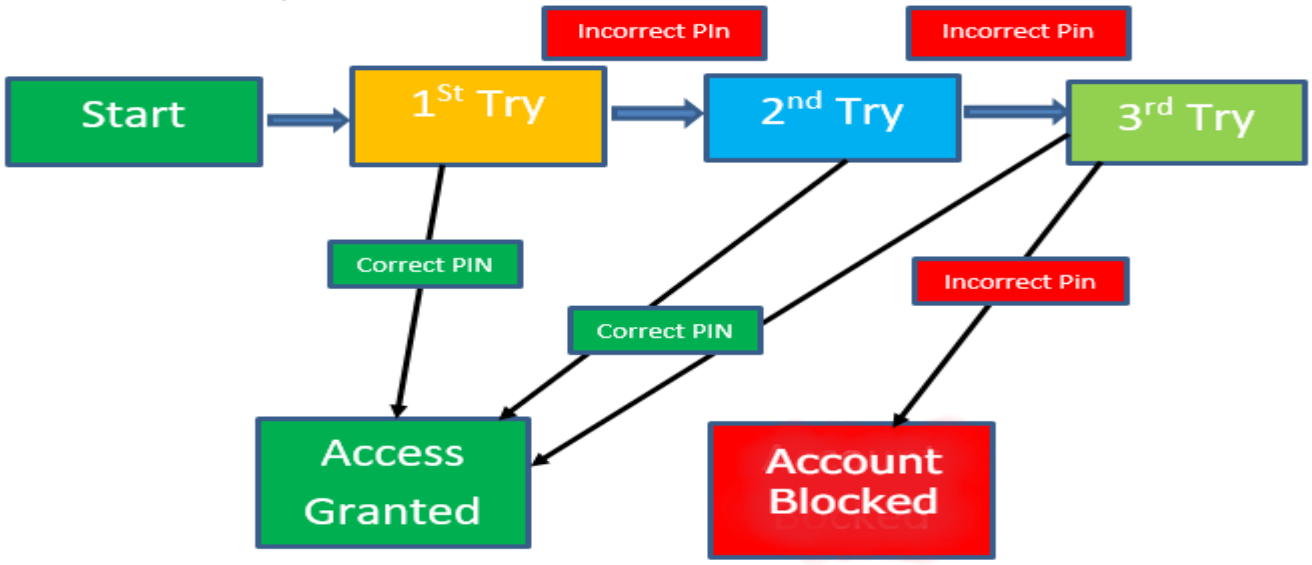
State Transition Testing is a black box testing technique in which changes made in input conditions cause state changes or output changes in the Application under Test(AUT). State transition testing helps to analyze behavior of an application for different input conditions. Testers can provide positive and negative input test values and record the system behavior.

Example :

Let's consider an ATM system function where if the user enters the invalid password three times the account will be locked.

In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.

State transition diagram



In the diagram whenever the user enters the correct PIN, he is moved to Access granted state, and if he enters the wrong password, he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.

State Transition Table

	Correct PIN	Incorrect PI
S1) Start	S5	S2
S2) 1st attempt	S5	S3
S3) 2nd attempt	S5	S4
S4) 3rd attempt	S5	S6
S5) Access Granted	–	–
S6) Account blocked	–	–

In the table when the user enters the correct PIN, state is transitioned to S5 which is Access granted. And if the user enters a wrong password he is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

7. a) Enlist any four tools to perform White box testing? Also explain advantages and disadvantages of white box testing.

Ans 7. a)

White Box Testing Tools

#1) Veracode

Veracode's white box testing tools will help you in identifying and resolving the software flaws quickly and easily at a reduced cost. It supports several application languages like .NET, C++, JAVA etc. and also enables you to test the security of desktop, web as well as mobile applications. Still, there are several other benefits of Veracode tool.

#2) EclEmma

EclEmma was initially designed for test runs and analysis within the Eclipse workbench. It is considered to be a free Java code coverage tool and has several features as well.

#3)RCUNIT

A framework which is used for testing C programs is known as RCUNIT. RCUNIT can be used accordingly based on the terms of the MIT License. It is free to use and install.

#4) cfix

cfix is one of the unit testing frameworks for C/C++ which solely aims at making test suites development as simple and easy as possible. Meanwhile, cfix is typically specialized for NT Kernel mode and Win32.

#5) Google test

Google test is Google's C++ test framework. Test Discovery, Death tests, Value-parameterized tests, fatal & non-fatal failures, XML test report generation etc. are few features of Google Test but there are several other features too. Linux, Windows, Symbian, Mac OS X are few platforms where Google Test has been used.

7. b) What is static testing? Explain in detail with suitable example.

Ans 7. b)

Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application. Whereas in Dynamic Testing checks, the code is executed to detect the defects. Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that cannot be found using Dynamic Testing, can be easily found by Static Testing.

8. a) What is McCabe's Cyclomatic Complexity?
Compute Cyclomatic Complexity of following program code.

```
i = 0;
n=4; //N-Number of nodes present in the graph
while (i<n-1) do
j = i + 1;
while (j<n) do
if A[i]<A[j] then
swap(A[i], A[j]);
end do;
j=j+1;
end do;
```

Ans 8. a)

Cyclomatic Complexity in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs or with respect to functions, modules, methods or classes within a software program.

8. b) Enlist any two tools for computation of Cyclomatic Complexity? How does Cyclomatic complexity is useful in software testing?

Ans 8. b)

Tools for Cyclomatic Complexity calculation:

Many tools are available for determining the complexity of the application. Some complexity calculation tools are used for specific technologies. Complexity can be found by the number of decision points in a program. The decision points are if, for, for-each, while, do, catch, case statements in a source code.

Examples of tools are

- **OCLint – Static code analyzer for C and Related Languages**
- **Reflector Add In – Code metrics for .NET assemblies**
- **GMetrics – Find metrics in Java related applications**

Uses of Cyclomatic Complexity:

Cyclomatic Complexity can prove to be very helpful in

- **Helps developers and testers to determine independent path executions**
- **Developers can assure that all the paths have been tested at least once**
- **Helps us to focus more on the uncovered paths**
- **Improve code coverage in Software Engineering**
- **Evaluate the risk associated with the application or program**
- **Using these metrics early in the cycle reduces more risk of the program**

9. a) What is unit testing? Explain in detail with suitable example.

Ans 9. a)

- **Unit testing is testing the smallest testable unit of an application.**
- **It is done during the coding phase by the developers.**
- **To perform unit testing, a developer writes a piece of code (unit tests) to verify the code to be tested (unit) is correct.**

A Real-world Example

You have written a function to add two numbers:

```
int Add(int a, int b) { return a+b; }
```

The above function takes two numbers as input and returns their sum.

A unit test code would look something like this:

```
void TestAdd1() { Assert.IsEqual(Add(5, 10), 15) }
```

The above unit test "asserts" that 5 + 10 is equal to 15. If the Add function returns anything else Assert.IsEqual result in error and the test case will fail.

You will probably add a few more unit test cases like these:

```
void TestAdd2() { Assert.IsEqual(Add(500, 1000), 1500) }
void TestAdd3() { Assert.IsEqual(Add(0, 1000), 1000) }
void TestAdd4() { Assert.IsEqual(Add(-100, 100), 0) }
void TestAdd5() { Assert.IsEqual(Add(-100, -1100), -1200) }
```

After you write your test cases, you will run them to verify that everything is working correctly.

Later another developer, in addition to adding some of his code, accidentally modifies the Add function as:

```
int Add(int a, int b) { return a*b; }
```

As you can see, instead of adding the two numbers, the code now multiplies them.

When the developer runs the unit tests that you have designed for this function, they will fail. The new developer can trace the failed test cases back to the function and fix the code.

9. b) Explain in detail about advantages and disadvantages of unit testing.

Ans 9. a)

Advantages of Unit Testing:

1. **Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.**
2. **Unit testing allows the programmer to refine code and make sure the module works properly.**
3. **Unit testing enables testing parts of the project without waiting for others to be completed.**
4. **Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.**
5. **Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.**
6. **Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.**
7. **Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.**
8. **Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.**
9. **Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.**
10. **Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.**

Disadvantages of Unit Testing:

1. **The process is time-consuming for writing the unit test cases.**
2. **Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.**
3. **Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.**
4. **It requires more time for maintenance when the source code is changed frequently.**
5. **It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.**
6. **Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.**
7. **Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.**
8. **Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.**
9. **Difficulty in Testing Interactions: Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.**
10. **Difficulty in Testing User Interfaces: Unit testing may not be suitable for testing user interfaces, as it typically focuses on the functionality of individual units.**
11. **Over-reliance on Automation: Over-reliance on automated unit tests can lead to a false sense of security, as automated tests may not uncover all possible issues or bugs.**
12. **Maintenance Overhead: Unit testing requires ongoing maintenance and updates, as the code and test cases must be kept up-to-date with changes to the software.**

10. a) What is integration testing? Explain in detail with suitable example.

Ans 10. a) Integration testing -- also known as integration and testing (I&T) -- is a type of software testing in which the different units, modules or components of a software application are tested as a combined entity. However, these modules may be coded by different programmers.
The aim of integration testing is to test the interfaces between the modules and expose any defects that may arise when these components are integrated and need to interact with each other.

Integration testing example
Consider a video-streaming mobile application. Its core features include the following:

- **Sign up/log in.**
- **View different monthly/yearly subscription plans.**
- **Choose personalized plans.**
- **Watch streaming video.**

Once users download the applications, they see a sign-up form where they can enter their account information. After successful authorization, they are redirected to a page listing different subscription plans. They can choose their own plan and then complete the payment.

Any errors in this logical flow could cause problems for the user and lead to losses for the app company. Integration testing can help find and fix such errors.
So, after each module is ready, testers conduct unit testing. And once all modules are available, testers test them together to check their interfaces and data flows. If no errors are detected, the end user should be able to successfully complete their transaction.

10. b) Explain in detail about advantages and disadvantages of integration testing.

Integration testing:

The objective of system integration is to build a "working" version of the system by
(i)putting the modules together in an incremental manner and
(ii) ensuring that the additional modules work as expected without disturbing the functionalities of the modules put together.

Advantages of integration testing:

- **Integration testing provides a systematic technique for assembling a software system while conducting tests to uncover errors associated with interfacing.**
- **The application is tested in order to verify that it meets the standards set by the client as well as reassuring the development team that assumptions which were made during unit testing are correct.**
- **Integration testing need not wait until all the modules of a system are coded and unit tested. Instead, it can begin as soon as the relevant modules are available.**
- **Integration testing or incremental testing is necessary to verify whether the software modules work in unity.**
- **System Integration testing includes a number of techniques like Incremental, Top-down, Bottom –Up, Sandwich and Big Bang Integration techniques.**

Disadvantages of Integration testing :

The Software Industry uses variety of strategies to execute Integration testing , that are :

- **Big Bang Approach :**
- **Incremental Approach: which is further divided into following**
  - **Top Down Approach**
  - **Bottom Up Approach**
  - **Sandwich Approach - Combination of Top Down and Bottom Up**

Big Bang Approach :

- **Here all component are integrated together at once, and then tested.**
  Disadvantages:
  - **Fault Localization is difficult.**
  - **Given the sheer number of interfaces that need to be tested in this approach, some interfaces links to be tested could be missed easily.**
  - **Since the integration testing can commence only after "all" the modules are designed, testing team will have less time for execution in the testing phase.**
  - **Since all modules are tested at once, high risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.**

Incremental Approach:

In this approach, testing is done by joining two or more modules that are logically related. Then the other related modules are added and tested for the proper functioning. Process continues until all of the modules are joined and tested successfully.

Incremental Approach in turn is carried out by two different Methods:

- **Bottom Up**

-Top Down
Bottom up Integration
In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Disadvantages:

- **Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.**
- **Early prototype is not possible**

Top down Integration:
In Top to down approach, testing takes place from top to down following the control flow of the software system.

Disadvantages:

- **Needs many Stubs.**

-Modules at lower level are tested inadequately.