200+ Software Testing Interview Questions and Answers

1. What is a Bug?

When actual result deviates from the expected result while testing a software application or product then it results into a defect. Hence, any deviation from the specification mentioned in the product functional specification document is a defect. In different organizations it's called a Bug.

2. What is a Defect?

If software misses some feature or function from what is there in requirement it is called as defect.

3. What is CMM?

The Capability Maturity Model for Software (CMM or SW-CMM) is a model for judging the maturity of the software processes of an organization and for identifying the key practices that are required to increase the maturity of these processes.

4. What is Beta Testing?

Beta testing is testing of a release of a software product conducted by customers.

5. What is Black Box Testing?

Testing based on an analysis of the specification of a piece of software without reference to its internal workings. The goal is to test how well the component conforms to the published requirements for the component.

6. What is Bottom Up Testing?

An approach to integration testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

7. What is Boundary Testing?

Test which focus on the boundary or limit conditions of the software being tested. (Some of these tests are stress tests).

8. What is Boundary Value Analysis?

BVA is similar to Equivalence Partitioning but focuses on "corner cases" or values that are usually out of range as defined by the specification. This means that if a function expects all values in range of negative 100 to positive 1000, test inputs would include negative 101 and positive 1001.

9. What is Branch Testing?

Testing of all branches in the program source code is called Branch Testing.

10. What is Coding?

The generation of source code is called Coding.

11. What is Compatibility Testing?

Testing whether software is compatible with other elements of a system with which it should operate, e.g. browsers, Operating Systems, or hardware.

12. What is a Component?

A component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions.

13. What is Component Testing?

Testing of individual software components is called Component testing.

14. What is Acceptance Testing?

Testing conducted to enable a user/customer to determine whether to accept a software product.

Normally performed to validate the software meets a set of agreed acceptance criteria.

15. What is Accessibility Testing?

Verifying a product is accessible to the people having disabilities (deaf, blind, mentally disabled etc.).

16. What is Ad Hoc Testing?

A testing phase where the tester tries to 'break' the system by randomly trying the system's functionality is called Ad Hoc testing. This can include negative testing also.

17. What is Agile Testing?

Testing practice for projects using agile methodologies, treating development as the customer of testing and emphasizing a test-first design paradigm. See also Test Driven Development.

18. What is Application Binary Interface (ABI)?

A specification defining requirements for portability of applications in binary forms across different system platforms and environments is called Application Binary Interface (ABI).

19. What is Application Programming Interface (API)?

A formalized set of software calls and routines that can be referenced by an application program in order to access supporting system or network services is called Application Programming Interface (API).

20. What is Automated Software Quality (ASQ)?

The use of software tools, such as automated testing tools, to improve software quality is called Automated Software Quality (ASQ).

21. What is Automated Testing?

Testing employing software tools which execute tests without manual intervention is called Automated Testing. Can be applied in GUI, performance, API, etc. testing. The use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

22. What is Backus-Naur Form?

It is a meta-language used to formally describe the syntax of a language.

23. What is Basic Block?

A sequence of one or more consecutive, executable statements containing no branches is called Basic Block.

24. What is Basis Path Testing?

A white box test case design technique that uses the algorithmic flow of the program to design tests.

25. What is Basis Set?

The set of tests derived using basis path testing.

26. What is Baseline?

The point at which some deliverable produced during the software engineering process is put under formal change control.

27. What you will do during the first day of job?

Few things that you should be doing on the first day of your job are

1. Reach office at least 15 minutes early
2. Be calm and relaxed
3. Have a smile on your face
4. Don’t be shy
5. Don’t try too hard to impress your colleagues
6. If you're offered to go have lunch with your new boss and co-workers then you should go. It's important to show that you're ready to mingle with your new team.
7. Pay attention to how decisions are made.

28. What is Binary Portability Testing?

Testing an executable application for portability across system platforms and environments, usually for conformation to an ABI specification is called Binary Portability Testing.

29. What is Breadth Testing?

A test suite that exercises the full functionality of a product but does not test features in detail is called Breadth Testing.

30. What is CAST?

Computer Aided Software Testing refers to the computing-based processes, techniques and tools for testing software applications or programs.

31. What is Capture/Replay Tool?

A test tool that records test input as it is sent to the software under test. The input cases stored can then be used to reproduce the test at a later time. Most commonly applied to GUI test tools.

32. What is Cause Effect Graph?

A graphical representation of inputs and the associated outputs effects which can be used to design test cases.

33. What is Code Complete?

Phase of development where functionality is implemented entirety; bug fixes are all that are left. Allfunctions found in the Functional Specifications have been implemented.

34. What is Code Coverage?
An analysis method that determines which parts of the software have been executed (covered) by the test case suite and which parts have not been executed and therefore may require additional attention.

35. What is Code Inspection?

A formal testing technique where the programmer reviews source code with a group who ask questions analyzing the program logic, analyzing the code with respect to a checklist of historically common programming errors, and analyzing its compliance with coding standards. Know more about the Inspection in software testing.

36. What is Code Walkthrough?

A formal testing technique where source code is traced by a group with a small set of test cases, while the state of program variables is manually monitored, to analyze the programmer's logic and assumptions. Know more about Walkthrough in software testing.

37. What is Concurrency Testing?

Multi-user testing geared towards determining the effects of accessing the same application code, module or database records. Identifies and measures the level of locking, deadlocking and use of single-threaded code and locking semaphores.

38. What is Conformance Testing?

Conformance testing, also known as compliance testing, is a methodology used in engineering to ensure that a product, process, computer program or system meets a defined set of standards.

39. What is Context Driven Testing?

The context-driven school of software testing is flavour of Agile Testing that advocates continuous and creative evaluation of testing opportunities in light of the potential information revealed and the value of that information to the organization right now.

40. What is Conversion Testing?

Testing of programs or procedures used to convert data from existing systems for use in replacement systems.

41. What is Cyclomatic Complexity?

A measure of the logical complexity of an algorithm, used in white-box testing is called Cyclomatic Complexity.

42. What is Data Dictionary?

A database that contains definitions of all data items defined during analysis.

43. What is Data Flow Diagram?

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects.

44. What is Data Driven Testing?

Testing in which the action of a test case is parameterized by externally defined data values, maintained as a file or spreadsheet. This is a common technique used in Automated Testing.

45. What is Debugging?

The process of finding and removing the causes of software failures is called Debugging.

46. What is Defect?

When actual result deviates from the expected result while testing a software application or product then it results into a defect.

47. What is Dependency Testing?

Dependency Testing, a testing technique in which an application's requirements are pre-examined for an existing software, initial states in order to test the proper functionality. The impacted areas of the application are also tested when testing the new features or existing features.

48. What is Depth Testing?

A test that exercises a feature of a product in full detail is called Depth testing.

49. What is Dynamic Testing?

Testing software through executing it is called Dynamic testing. Also know about Static Testing.

50. What is Emulator?

A device, computer program, or system that accepts the same inputs and produces the same outputs in a given system is called Emulator.

51. What is Endurance Testing?

Checks for memory leaks or other problems that may occur with prolonged execution.

52. What is End-to-End testing?

Testing a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate is called End-to-End testing.

53. What is Equivalence Class?

A portion of a component's input or output domains for which the component's behaviour is assumed to be the same from the component's specification.

54. What is Equivalence Partitioning?

A test case design technique for a component in which test cases are designed to execute representatives from equivalence classes.

55. What is Exhaustive Testing?

Testing which covers all combinations of input values and preconditions for an element of the software under test.

56. What is Functional Decomposition?

A technique used during planning, analysis and design; creates a functional hierarchy for the software.

57. What is Functional Specification?

A document that describes in detail the characteristics of the product with regard to its intended features is called Functional Specification.

58. What is Functional Testing?

Testing the features and operational behaviour of a product to ensure they correspond to its specifications. Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. or Black Box Testing.

59. What is Glass Box Testing?
White Box Testing is also known as Glass Box testing.

60. What is Gorilla Testing?

Gorilla Testing is a testing technique in which sometimes developers also join hands with testers to test a particular module thoroughly in all aspects.

61. What is Gray Box Testing?

A combination of Black Box and White Box testing methodologies is called Gray box testing. Testing a piece of software against its specification but using some knowledge of its internal workings.

62. What is High Order Tests?

Black-box tests conducted once the software has been integrated.

63. What is Independent Test Group (ITG)?

A group of people whose primary responsibility is software testing. Know more about Independent testing

64. What is Inspection?

A group review quality improvement process for written material. It consists of two aspects; product (document itself) improvement and process improvement (of both document production and inspection).

65. What is Integration Testing?

After integrating two different components together we do the integration testing. Integration testing is usually performed after unit and functional testing. This type of testing is especially relevant to client/server and distributed systems.
66. What is Installation Testing?

Installation testing is a kind of quality assurance work in the software industry that focuses on what customers will need to do to install and set up the new software successfully. The testing process may involve full, partial or upgrades install/uninstall processes.

67. What is Load Testing?

A load testing is a type of software testing which is conducted to understand the behaviour of the

application under a specific expected load.

Also see Performance Testing.
68. What is Localization Testing?

This term refers to making software specifically designed for a specific locality.

69. What is Loop Testing?

A white box testing technique that exercises program loops in order to validate them.

70. What is Metric?

Metric is a standard of measurement. Software metrics are the statistics describing the structure or content of a program. A metric should be a real objective measurement of something such as number of bugs per lines of code.

71. What is Monkey Testing?

Testing a system or an Application on the fly, i.e just few tests here and there to ensure the system or an application does not crash out.

72. What is Negative Testing?

Testing aimed at showing software does not work. This is also known as "test to fail". See also Positive Testing.

73. What is Path Testing?

Testing in which all paths are in the program source code are tested at least once.

74. What is Performance Testing?

Testing conducted to evaluate the compliance of a system or component with specified performance requirements. Often this is performed using an automated test tool to simulate large number of users.

75. What is Positive Testing?

Testing aimed at showing software works. This is also known as "test to pass".

76. What is Quality Assurance?

All those planned or systematic actions necessary to provide adequate confidence that a product or service is of the type and quality needed and expected by the customer.

77. What is Quality Audit?

A systematic and independent examination to determine whether quality activities and related results comply with planned arrangements and whether these arrangements are implemented effectively and are suitable to achieve objectives.

78. What is Quality Circle?

A group of individuals with related interests that meet at regular intervals to consider problems or other matters related to the quality of outputs of a process and to the correction of problems or to the improvement of quality.

79. What is Quality Control?

The operational techniques and the activities used to fulfill and verify requirements of quality.

80. What is Quality Management?

That aspect of the overall management function that determines and implements the quality policy.

81. What is Quality Policy?

In quality management system, a quality policy is a document jointly developed by management and quality experts to express the quality objectives of the organization, the acceptable level of quality and the duties of specific departments to ensure quality.

82. What is Quality System?

The organizational structure, responsibilities, procedures, processes, and resources for implementing quality management is called Quality System.

83. What is Race Condition?

A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.

84. What is Ramp Testing?

Continuously raising an input signal until the system breaks down.

85. What is Recovery Testing?

Recovery testing confirms that the program recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions.

86. What is Regression Testing?
Retesting a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made is called Regression testing.

87. What is Release Candidate?

A pre-release version, which contains the desired functionality of the final version, but which needs to be tested for bugs (which ideally should be removed before the final version is released).

88. What is Sanity Testing?

Brief test of major functional elements of a piece of software to determine if its basically operational.
See also Smoke Testing.

89. What is Scalability Testing?

Performance testing focused on ensuring the application under test gracefully handles increases in work load.

90. What is Security Testing?

Testing which confirms that the program can restrict access to authorized personnel and that the authorized personnel can access the functions available to their security level.

91. What is Smoke Testing?

In Software testing context, smoke testing refers to testing the basic functionality of the build.

92. What is Soak Testing?

Running a system at high load for a prolonged period of time is called Soak testing. For example; run several times more transactions in an entire day (or night) than would be expected in a busy day, to identify the performance problems that appear after a large number of transactions have been executed.

93. What is Software Requirements Specification?

A deliverable that describes all data, functional and behavioural requirements, all constraints, and all validation requirements for software/

94. What is Software Testing?

A set of activities conducted with the intent of finding errors in software.

95. What is Static Analysis?

Analysis of a program carried out without executing the program.

96. What is Static Analyzer?

A tool that carries out static analysis is called Static Analyzer.

97. What is Static Testing?

Analysis of a program carried out without executing the program.

98. What is Storage Testing?

Testing that verifies the program under test stores data files in the correct directories and that it reserves sufficient space to prevent unexpected termination resulting from lack of space. This is external storage as opposed to internal storage.

99. What is Stress Testing?

Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how. Often this is performance testing using a very high level of simulated load.

100. What is Structural Testing?

Testing based on an analysis of internal workings and structure of a piece of software. See also White Box Testing.

101. What is System Testing?

Testing that attempts to discover defects that are properties of the entire system rather than of its individual components is called System testing.

102. What is Testability?

The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

103. What is Testing?

The process of exercising software to verify that it satisfies specified requirements and to detect errors is called testing. The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs), and to evaluate the features of the software item (Ref. IEEE Std 829). The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.

104. What is the difference between quality assurance and testing?

Quality assurance involves the entire software development process and testing involves operation of a system or application to evaluate the results under certain conditions. QA is oriented to prevention and Testing is oriented to detection.

105. Why does software have bugs?

Software have bugs because of the following reasons: 1) Miscommunication 2) programming errors 3) time pressures. 4) Changing requirements 5) software complexity

106. How do you do usability testing, security testing, installation testing, ADHOC, safety and smoke testing?

Usability testing : testing the user friendliness , simplicity of software. Security testing is testing the access rights for example admin has all rights, user1 has only right to read and open not to write . Installation testing is testing the software while installing it in different environment. Safety and security goes hand in hand most of the time it used logins. Smoke testing is to test the software whether the basic functionality is working or not. If u wants to Test the basic functionalities then we go for ADHOC it happens if we don't have time to test procedure wise

107. What are memory leaks and buffer overflows?

Memory leaks means incomplete de-allocation - are bugs that happen very often. Buffer overflow means data sent as input to the server that overflows the boundaries of the input area, thus causing the server to misbehave.

108. What are the major differences between stress testing, load testing, Volume testing?

Stress testing means increasing the load and checking the performance at each level. Load testing means at a time giving more load by the expectation and checking the performance at that level. Volume testing means first we have to apply initial.

109. What is Exhaustive Testing?

Testing which covers all combinations of input values and preconditions for an element of the software under test.

110. What is Functional Decomposition?

A technique used during planning, analysis and design; creates a functional hierarchy for the software.

111. What is Functional Specification?

A document that describes in detail the characteristics of the product with regard to its intended features is called Functional Specification.

112. What is Test Bed?

An execution environment configured for testing. It may consist of specific hardware, OS, network topology, configuration of the product under test, other application or system software, etc. The Test Plan for a project should enumerate the test beds(s) to be used.

113. What is Test Case?

Test Case is a commonly used term for a specific test. This is usually the smallest unit of testing. A Test Case will consist of information such as requirements testing, test steps, verification steps, prerequisites, outputs, test environment, etc. A set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. Test Driven Development? Testing methodology associated with Agile Programming in which every chunk of code is covered by unit tests, which must all pass all the time, in an effort to eliminate unit-level and regression bugs during development. Practitioners of TDD write a lot of tests, i.e. an equal number of lines of test code to the size of the production code.

114. What is Test Driver?

A program or test tool used to execute a test. Also known as a Test Harness.

115. What is Test Environment?
The hardware and software environment in which tests will be run, and any other software with which the software under test interacts when under test including stubs and test drivers.

116. What is Test First Design?

Test-first design is one of the mandatory practices of Extreme Programming (XP).It requires that programmers do not write any production code until they have first written a unit test.

117. What is Test Harness?

A program or test tool used to execute tests. Also known as a Test Driver.

118. What is Test Plan?

A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

119. What is Test Procedure?

A document providing detailed instructions for the execution of one or more test cases is called test Procedure.

120. What is Test Script?

Test script is commonly used to refer to the instructions for a particular test that will be carried out by an automated test tool.

121. What is Test Specification?

Test specification is a document specifying the test approach for a software feature or combination or features and the inputs, predicted results and execution conditions for the associated tests.

122. What is Test Suite?

A collection of tests used to validate the behaviour of a product. The scope of a Test Suite varies from organization to organization. There may be several Test Suites for a particular product for example. In most cases however a Test Suite is a high level concept, grouping together hundreds or thousands of tests related by what they are intended to test.

123. What is Test Tool?

Computer programs used for the testing of a system, a component of the system, or its documentation is called a test tool.

124. What is Thread Testing?

A variation of top-down testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by successively lower levels.

125. What is Top Down Testing?

Top down testing is an approach to integration testing where the component at the top of the component hierarchy is tested first, with lower level components being simulated by stubs. Tested components are then used to test lower level components. The process is repeated until the lowest level components have been tested.

126. What is Total Quality Management?

Total Quality Management is a company commitment to develop a process that achieves high quality product and customer satisfaction.

127. What is Traceability Matrix?

Traceability Matrix is a document showing the relationship between Test Requirements and Test Cases.

128. What is Usability Testing?

Testing the ease with which users can learn and use a product.

129. What is Use Case?

Use case is the specification of tests that are conducted from the end-user perspective. Use cases tend to focus on operating software as an end-user would conduct their day-to-day activities.

130. What is Unit Testing?

Testing of individual software components is called Unit testing.

131. What is Validation?

The process of evaluating software at the end of the software development process to ensure compliance with software requirements is called Validation.

132. What is Verification?

The process of determining whether or not the products of a given phase of the software development cycle meet the implementation steps and can be traced to the incoming objectives established during the previous phase.

133. What is White Box Testing?

Testing based on an analysis of internal workings and structure of a piece of software. This includes

techniques such as Branch Testing and Path Testing. This also known as Structural Testing and

Glass Box Testing. Contrast with Black Box Testing. White box testing is used to test the internal logic

of the code for ex checking whether the path has been executed once, checking whether the

branches has been executed at least once .This is used to check the structure of the code.

134. What is Workflow Testing?

Workflow processes technique in software testing by routing a record through each possible path. These tests are performed to ensure that each workflow process accurately reflects the business process. This kind of testing holds good for workflow-based applications.

135. What's the difference between load and stress testing ?

One of the most common, but unfortunate misuse of terminology is treating “load testing” and “stress testing” as synonymous. The consequence of this ignorant semantic abuse is usually that the system is neither properly “load tested” nor subjected to a meaningful stress test. Stress testing is subjecting a system to an unreasonable load while denying it the resources (e.g., RAM, disc, mips, interrupts, etc.) needed to process that load. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in a decent manner (e.g., not corrupting or losing data). Bugs and failure modes discovered under stress testing may or may not be repaired depending on the application, the failure mode, consequences, etc. The load (incoming transaction stream) in stress testing is often deliberately distorted so as to force the system into resource depletion. Load testing is subjecting a system to a statistically representative (usually) load. The two main reasons for using such loads is in support of software reliability testing and in performance testing. The term 'load testing' by itself is too vague and imprecise to warrant use. For example, do you mean representative load,' 'overload,' 'high load,' etc. In performance testing, load is varied from a minimum (zero) to the maximum level the system can sustain without running out of resources or having, transactions >suffer (application-specific) excessive delay. A third use of the term is as a test whose objective is to determine the maximum sustainable load the system can handle. In this usage, 'load testing' is merely testing at the highest transaction arrival rate in performance testing.

136. What's the difference between QA and testing?

QA is more a preventive thing, ensuring quality in the company and therefore the product rather than just testing the product for software bugs? TESTING means 'quality control' QUALITY CONTROL measures the quality of a product QUALITY ASSURANCE measures the quality of processes used to create a quality product.

137. What is the best tester to developer ratio?

Tester : developer ratios range from 10:1 to 1:10. There's no simple answer. It depends on so many things, Amount of reused code, number and type of interfaces, platform, quality goals, etc. It also can depend on the development model. The more specs, the less testers. The roles can play a big part also. Does QA own beta? Do you include process auditors or planning activities? These figures can all vary very widely depending on how you define 'tester' and 'developer'. In some organizations, a 'tester' is anyone who happens to be testing software at the time -- such as their own. In other organizations, a 'tester' is only a member of an independent test group. It is better to ask about the test labour content than it is to ask about the tester/developer ratio. The test labour content, across most applications is generally accepted as 50%, when people do honest accounting. For life-critical software, this can go up to 80%.

138. How can new Software QA processes be introduced in an existing organization?

- A lot depends on the size of the organization and the risks involved. For large organizations with high-risk (in terms of lives or property) projects, serious management buy-in is required and a formalized QA process is necessary. - Where the risk is lower, management and organizational buy-in and QA implementation may be a slower, step-at-a-time process. QA processes should be balanced with productivity so as to keep bureaucracy from getting out of hand. - For small groups or projects, a more ad-hoc process may be appropriate, depending on the type of customers and projects. A lot will depend on team leads or managers, feedback to developers, and ensuring adequate communications among customers, managers, developers, and testers. - In all cases the most value for effort will be in requirements management processes, with a goal of clear, complete, testable requirement
specifications or expectations.

139. What are 5 common problems in the software development process?

The 5 common problems in the software development process are as follows:

1) Poor requirements - if requirements are unclear, incomplete, too general, or not testable, there will be problems. 2) Unrealistic schedule - if too much work is crammed in too little time, problems are inevitable. 3) Inadequate testing - no one will know whether or not the program is any good until the customer complaints or systems crash. 4) Features - requests to pile on new features after development is underway; extremely common. 5) Miscommunication - if developers don't know what's needed or customer's have erroneous expectations, problems are guaranteed.

140. What are 5 common solutions to software development problems?

The 5 common solutions to software development problems as given below:

1) Solid requirements - clear, complete, detailed, cohesive, attainable, testable requirements that are agreed to by all players. Use prototypes to help nail down requirements. 2) Realistic schedules - allow adequate time for planning, design, testing, bug fixing, re-testing, changes, and documentation; personnel should be able to complete the project without burning out. 3) Adequate testing - start testing early on, re-test after fixes or changes, plan for adequate time for testing and bug-fixing. 4) Stick to initial requirements as much as possible - be prepared to defend against changes and additions once development has begun, and be prepared to explain consequences. If changes are necessary, they should be adequately reflected in related schedule changes. If possible, use rapid prototyping during the design phase so that customers can see what to expect. This will provide them a higher comfort level with their requirements decisions and minimize changes later on. 5)

communication - require walkthroughs and inspections when appropriate; make extensive use of group communication tools - e-mail, groupware, networked bug-tracking tools and change management tools, intranet capabilities, etc.; insure that documentation is available and up-to-date - preferably electronic, not paper; promote teamwork and cooperation; use prototypes early on so that customers' expectations are clarified.

141. What is a 'good code'?

'Good code' is a code that works, is bug free, and is readable and maintainable. Some organizations have coding 'standards' that all developers are supposed to adhere to, but everyone has different ideas about what's best, or what is too many or too few rules. There are also various theories and metrics, such as McCabe Complexity metrics. It should be kept in mind that excessive use of standards and rules can stifle productivity and creativity. 'Peer reviews', 'buddy checks' code analysis tools, etc. can be used to check for problems and enforce standards. For C and C++ coding, here are some typical ideas to consider in setting rules/standards; these may or may not apply to a particular situation: - minimize or eliminate use of global variables. - use descriptive function and method names

- use both upper and lower case, avoid abbreviations, use as many characters as necessary to be adequately descriptive (use of more than 20 characters is not out of line); be consistent in naming conventions. - use descriptive variable names - use both upper and lower case, avoid abbreviations, use as many characters as necessary to be adequately descriptive (use of more than 20 characters is not out of line); be consistent in naming conventions. - function and method sizes should be minimized; less than 100 lines of code is good, less than 50 lines is preferable. - function descriptions should be clearly spelled out in comments preceding a function's code.- organize code for readability.

- use whitespace generously - vertically and horizontally - each line of code should contain 70 characters max. - one code statement per line. - coding style should be consistent throughout a program (eg, use of brackets, indentations, naming conventions, etc.) - in adding comments, err on the side of too many rather than too few comments; a common rule of thumb is that there should be at least as many lines of comments (including header blocks) as lines of code. - no matter how small, an application should include documentation of the overall program function and flow (even a few paragraphs is better than nothing); or if possible a separate flow chart and detailed program documentation. - make extensive use of error handling procedures and status and error logging. - For C++, to minimize complexity and increase maintainability, avoid too many levels of inheritance in class hierarchies (relative to the size and complexity of the application). Minimize use of multiple inheritance, and minimize use of operator overloading (note that the Java programming language eliminates multiple inheritance and operator overloading.) - For C++, keep class methods small, less than 50 lines of code per method is preferable. - For C++, make liberal use of exception handlers

142. What is a 'good design'?

'Design' could refer to many things, but often refers to 'functional design' or 'internal design'. Good internal design is indicated by software code whose overall structure is clear, understandable, easily modifiable, and maintainable; is robust with sufficient error-handling and status logging capability; and works correctly when implemented. Good functional design is indicated by an application whose functionality can be traced back to customer and end-user requirements. For programs that have a user interface, it's often a good idea to assume that the end user will have little computer knowledge and may not read a user manual or even the on-line help; some common rules-of-thumb include: - the program should act in a way that least surprises the user - it should always be evident to the user what can be done next and how to exit - the program shouldn't let the users do something stupid without warning them. Know more about test design, test design technique, categories of test design technique and test design tools

143. What makes a good test engineer?

A good test engineer has a 'test to break' attitude, an ability to take the point of view of the customer, a strong desire for quality, and an attention to detail. Tact and diplomacy are useful in maintaining a cooperative relationship with developers, and an ability to communicate with both technical (developers) and non-technical (customers, management) people is useful. Previous software development experience can be helpful as it provides a deeper understanding of the software development process, gives the tester an appreciation for the

developers' point of view, and reduce the learning curve in automated test tool programming. Judgment skills are needed to assess high-risk areas of an application on which to focus testing efforts when time is limited.

144. What makes a good Software QA engineer?

The same qualities a good tester has are useful for a QA engineer. Additionally, they must be able to understand the entire software development process and how it can fit into the business approach and goals of the organization. Communication skills and the ability to understand various sides of issues are important. In organizations in the early stages of implementing QA processes, patience and diplomacy are especially needed. An ability to find problems as well as to see 'what's missing' is important for inspections and reviews.

145. What makes a good QA or Test manager?

A good QA, test, or QA/Test(combined) manager should: - be familiar with the software development process - be able to maintain enthusiasm of their team and promote a positive atmosphere, despite what is a somewhat 'negative' process (e.g., looking for or preventing problems) - be able to promote teamwork to increase productivity - be able to promote cooperation between software, test, and QA engineers - have the diplomatic skills needed to promote improvements in QA processes -have the ability to withstand pressures and say 'no' to other managers when quality is insufficient or QA processes are not being adhered to - have people judgment skills for hiring and keeping skilled personnel- be able to communicate with technical and non-technical people, engineers, managers, and customers. - be able to run meetings and keep them focused

146. What's the role of documentation in QA?

The role of documentation in QA is Critical. (Note that documentation can be electronic, not necessarily paper.) QA practices should be documented such that they are repeatable. Specifications, designs, business rules, inspection reports, configurations, code changes, test plans, test cases, bug reports, user manuals, etc. should all be documented. There should ideally be a system for easily finding and obtaining documents and determining what documentation will have a particular piece of information. Change management for documentation should be used if possible. Know more about documentation testing

147. What's the big deal about 'requirements'?

One of the most reliable methods of insuring problems, or failure, in a complex software project is to have poorly documented requirements specifications. Requirements are the details describing an application's externally-perceived functionality and properties. Requirements should be clear, complete, reasonably detailed, cohesive, attainable, and testable. A non-testable requirement would be, for example, 'user-friendly' (too subjective). A testable requirement would be something like 'the user must enter their previously-assigned password to access the application'. Determining and organizing requirements details in a useful and efficient way can be a difficult effort; different methods are available depending on the particular project. Many books are available that describe various approaches to this task. Care should be taken to involve ALL of a project's significant 'customers' in the requirements process. 'Customers' could be in-house personnel or out, and could include end-users, customer acceptance testers, customer contract officers, customer management, future software maintenance engineers, salespeople, etc. Anyone who could later derail the project if their expectations aren't met should be included if possible. Organizations vary considerably in their handling of requirements specifications. Ideally, the requirements are spelled out in a document with statements such as 'The product shall.....'. 'Design' specifications should not be confused with

'requirements'; design specifications should be traceable back to the requirements. In some organizations requirements may end up in high level project plans, functional specification documents, in design documents, or in other documents at various levels of detail. No matter what they are called, some type of documentation with detailed requirements will be needed by testers in order to properly plan and execute tests. Without such documentation, there will be no clear-cut way to determine if a software application is performing correctly.

148. What steps are needed to develop and run software tests?

The following are some of the steps to consider: - Obtain requirements, functional design, and internal design specifications and other necessary documents - Obtain budget and schedule requirements - Determine project-related personnel and their responsibilities, reporting requirements, required standards and processes (such as release processes, change processes, etc.) - Identify application's higher-risk aspects, set priorities, and determine scope and limitations of tests - Determine test approaches and methods - unit, integration, functional, system, load, usability tests, etc. - Determine test environment requirements (hardware, software, communications, etc.) -Determine testware requirements (record/playback tools, coverage analyzers, test tracking, problem/bug tracking, etc.) - Determine test input data requirements - Identify tasks, those responsible for tasks, and labor requirements - Set schedule estimates, timelines, milestones - Determine input equivalence classes, boundary value analyses, error classes - Prepare test plan document and have needed reviews/approvals - Write test cases - Have needed reviews/inspections/approvals of test cases - Prepare test environment and testware, obtain needed user manuals/reference documents/configuration guides/installation guides, set up test tracking processes, set up logging and archiving processes, set up or obtain test input data - Obtain and install software releases - Perform tests - Evaluate and report results - Track problems/bugs and fixes - Retest as needed - Maintain and update test plans, test cases, test environment, and testware through life cycle

149. What is 'configuration management'?

Configuration management covers the processes used to control, coordinate, and track: code, requirements, documentation, problems, change requests, designs, tools/compilers/libraries/patches, changes made to them, and who makes the changes.

150. What if the software is so buggy it can't really be tested at all?

The best bet in this situation is for the testers to go through the process of reporting whatever bugs or blocking-type problems initially show up, with the focus being on critical bugs. Since this type of problem can severely affect schedules, and indicates deeper problems in the software development
process (such as insufficient unit testing or insufficient integration testing, poor design, improper build or release procedures, etc.) managers should be notified, and provided with some documentation as evidence of the problem. Know more about Severity and Priority

151. How can it be known when to stop testing?

This can be difficult to determine. Many modern software applications are so complex, and run in such an interdependent environment, that complete testing can never be done. Common factors in deciding when to stop are: - Deadlines (release deadlines, testing deadlines, etc.)- Test cases completed with certain percentage passed - Test budget depleted - Coverage of code/functionality/requirements reaches a specified point - Bug rate falls below a certain level - Beta or alpha testing period ends

152. What if there isn't enough time for thorough testing?

Use risk analysis to determine where testing should be focused. Since it's rarely possible to test every possible aspect of an application, every possible combination of events, every dependency, or everything that could go wrong, risk analysis is appropriate to most software development projects. This requires judgement skills, common sense, and experience. (If warranted, formal methods are also available.) Considerations can include: - Which functionality is most important to the project's intended purpose? - Which functionality is most visible to the user? - Which functionality has the largest safety impact? - Which functionality has the largest financial impact on users? - Which aspects of the application are most important to the customer? - Which aspects of the application can be tested early in the development cycle? - Which parts of the code are most complex, and thus most subject to errors? - Which parts of the application were developed in rush or panic mode? - Which aspects of similar/related previous projects caused problems? - Which aspects of similar/related previous projects had large maintenance expenses? - Which parts of the requirements and design are unclear or poorly

thought out? - What do the developers think are the highest-risk aspects of the application? - What kinds of problems would cause the worst publicity? - What kinds of problems would cause the most customer service complaints?- What kinds of tests could easily cover multiple functionalities? - Which tests will have the best high-risk-coverage to time-required ratio?

153. What can be done if requirements are changing continuously?

This is a common problem and a major headache. - Work with the project's stakeholders early on to understand how requirements might change so that alternate test plans and strategies can be worked out in advance, if possible. - It's helpful if the application's initial design allows for some adaptability so that later changes do not require redoing the application from scratch. - If the code is well-commented and well-documented this makes changes easier for the developers.- Use rapid prototyping whenever possible to help customers feel sure of their requirements and minimize changes. - The project's initial schedule should allow for some extra time commensurate with the possibility of changes.- Try to move new requirements to a 'Phase 2' version of an application, while using the original requirements for the 'Phase 1' version. - Negotiate to allow only easily-implemented new requirements into the project, while moving more difficult new requirements into future versions of the application. - Be sure that customers and management understand the scheduling impacts, inherent risks, and costs of significant requirements changes. Then let management or the customers (not the developers or testers) decide if the changes are warranted - after all, that's their job. - Balance the effort put into setting up automated testing with the expected effort required to re-do them to deal with changes. - Try to design some flexibility into automated test scripts. - Focus initial automated testing on application aspects that are most likely to remain unchanged. - Devote appropriate effort to risk analysis of changes to minimize regression testing needs. - Design some flexibility into test cases (this is not easily done; the best bet might be to minimize the detail in the test cases, or set up only higher-level generic-type test plans) - Focus less on detailed test plans and test cases and more on ad hoc testing (with an understanding of the added risk that this entails).

154. What if the project isn't big enough to justify extensive testing?

Consider the impact of project errors, not the size of the project. However, if extensive testing is still not justified, risk analysis is again needed and the same considerations as described previously in 'What if there isn't enough time for thorough testing?' apply. The tester might then do ad hoc testing, or write up a limited test plan based on the risk analysis.

155. What if the application has functionality that wasn't in the requirements?

It may take serious effort to determine if an application has significant unexpected or hidden functionality, and it would indicate deeper problems in the software development process. If the functionality isn't necessary to the purpose of the application, it should be removed, as it may have unknown impacts or dependencies that were not taken into account by the designer or the customer. If not removed, design information will be needed to determine added testing needs or regression testing needs. Management should be made aware of any significant added risks as a result of the unexpected functionality. If the functionality only effects areas such as minor improvements in the user interface, for example, it may not be a significant risk.

156. How can Software QA processes be implemented without stifling productivity?

By implementing QA processes slowly over time, using consensus to reach agreement on processes, and adjusting and experimenting as an organization grows and matures, productivity will be improved instead of stifled. Problem prevention will lessen the need for problem detection, panics and burn-out will decrease, and there will be improved focus and less wasted effort. At the same time, attempts should be made to keep processes simple and efficient, minimize paperwork, promote computer-based processes and automated tracking and reporting, minimize time required in meetings, and promote training as part of the QA process. However, no one - especially talented technical types - likes rules or bureaucracy, and in the short run things may slow down a bit. A typical scenario would be that more days of planning and development will be needed, but less time will be required for late-night bug-fixing and calming of irate customers.

157. What if an organization is growing so fast that fixed QA processes are impossible?
This is a common problem in the software industry, especially in new technology areas. There is no easy solution in this situation, other than: - Hire good people - Management should 'ruthlessly prioritize' quality issues and maintain focus on the customer - Everyone in the organization should be clear on what 'quality' means to the customer

158. How does a client/server environment affect testing?

Client/server applications can be quite complex due to the multiple dependencies among clients, data communications, hardware, and servers. Thus testing requirements can be extensive. When time is limited (as it usually is) the focus should be on integration and system testing. Additionally, load/stress/performance testing may be useful in determining client/server application limitations and capabilities. There are commercial tools to assist with such testing.

159. How can World Wide Web sites be tested?

Web sites are essentially client/server applications - with web servers and 'browser' clients. Consideration should be given to the interactions between html pages, TCP/IP communications, Internet connections, firewalls, applications that run in web pages (such as applets, javascript, plug-in applications), and applications that run on the server side (such as cgi scripts, database interfaces, logging applications, dynamic page generators, asp, etc.). Additionally, there are a wide variety of servers and browsers, various versions of each, small but sometimes significant differences between them, variations in connection speeds, rapidly changing technologies, and multiple standards and protocols. The end result is that testing for web sites can become a major ongoing effort. Other considerations might include: - What are the expected loads on the server (e.g., number of hits per unit time?), and what kind of performance is required under such loads (such as web server response time, database query response times). What kinds of tools will be needed for performance testing (such as web testing tools, other tools already in house that can be adapted, web robot downloading tools, etc.)? - Who is the target audience? What kind of browsers will they be using? What kind of connection speed will they by using? Are they intra- organization (thus with likely high connection speeds and similar browsers) or Internet-wide (thus with a wide variety of connection speeds and browser types)? - What kind of performance is expected on the client side (e.g., how fast should pages appear, how fast should animations, applets, etc. load and run)? - Will down time for server and content maintenance/upgrades be allowed? How much? - What kinds of security (firewalls, encryptions, passwords, etc.) will be required and what is it expected to do? How can it be tested? - How reliable are the site's Internet connections required to be? And how does that affect backup system or redundant connection requirements and testing? - What processes will be required to manage updates to the web site's content, and what are the requirements for maintaining, tracking, and controlling page content, graphics, links, etc.? - Which HTML specification will be adhered to? How strictly? What variations will be allowed for targeted browsers? - Will there be any standards or requirements for page appearance and/or graphics throughout a site or parts of a site?? - How will internal and external links be validated and updated? how often? - Can testing be done on the production system, or will a separate test system be required? How are browser caching, variations in browser option settings, dial-up connection variability, and real-world internet 'traffic congestion' problems to be accounted for in testing?- How extensive or customized are the server logging and reporting requirements; are they considered an integral part of the system and do they require testing?- How are cgi programs, applets, java scripts, ActiveX components, etc. to be maintained, tracked, controlled, and tested? - Pages should be 3-5 screens max unless content is tightly focused on a single topic. If larger, provide internal links within the page. - The page layouts and design elements should be consistent throughout a site, so that it's clear to the user that they're still within a site. - Pages should be as browser-independent as possible, or pages should be provided or generated based on the browser-type. - All pages should have links external to the page; there should be no dead-end pages. - The page owner, revision date, and a link to a contact person or organization should be included on each page.

160. How is testing affected by object-oriented designs?

Well-engineered object-oriented design can make it easier to trace from code to internal design to functional design to requirements. While there will be little affect on black box testing (where an

understanding of the internal design of the application is unnecessary), white-box testing can be oriented to the application's objects. If the application was well-designed this can simplify test design.

161. What is Extreme Programming and what's it got to do with testing?

Extreme Programming (XP) is a software development approach for small teams on risk-prone projects with unstable requirements. It was created by Kent Beck who described the approach in his book 'Extreme Programming Explained'. Testing ('extreme testing') is a core aspect of Extreme Programming. Programmers are expected to write unit and functional test code first - before the application is developed. Test code is under source control along with the rest of the code. Customers are expected to be an integral part of the project team and to help develop scenarios for acceptance/black box testing. Acceptance tests are preferably automated, and are modified and rerun for each of the frequent development iterations. QA and test personnel are also required to be an integral part of the project team. Detailed requirements documentation is not used, and frequent re-scheduling, re-estimating, and re-prioritizing is expected.

162. Will automated testing tools make testing easier?

- Possibly. For small projects, the time needed to learn and implement them may not be worth it. For larger projects, or on-going long-term projects they can be valuable. - A common type of automated tool is the 'record/playback' type. For example, a tester could click through all combinations of menu choices, dialog box choices, buttons, etc. in an application GUI and have them 'recorded' and the results logged by a tool. The 'recording' is typically in the form of text based on a scripting language that is interpretable by the testing tool. If new buttons are added, or some underlying code in the application is changed, etc. the application can then be retested by just 'playing back' the 'recorded' actions, and comparing the logging results to check effects of the changes. The problem with such tools is that if there are continual changes to the system being tested, the 'recordings' may have to be changed so much that it becomes very time-consuming to continuously update the scripts. Additionally, interpretation of results (screens, data, logs, etc.) can be a difficult task. Note that there

are record/playback tools for text-based interfaces also, and for all types of platforms.- Other automated tools can include: code analyzers - monitor code complexity, adherence to standards, etc. coverage analyzers - these tools check which parts of the code have been exercised by a test, and may be oriented to code statement coverage, condition coverage, path coverage, etc. memory analyzers - such as bounds-checkers and leak detectors. load/performance test tools - for testing client/server and web applications under various load levels. web test tools - to check that links are valid, HTML code usage is correct, client-side and server-side programs work, a web site's interactions are secure. other tools - for test case management, documentation management, bug reporting, and configuration management.

163. What's the difference between black box and white box testing?

Black-box and white-box are test design methods. Black-box test design treats the system as a “black-box”, so it doesn't explicitly use knowledge of the internal structure. Black-box test design is usually described as focusing on testing functional requirements. Synonyms for black-box include: behavioural, functional, opaque-box, and closed-box. White-box test design allows one to peek inside the “box”, and it focuses specifically on using internal knowledge of the software to guide the selection of test data. Synonyms for white-box include: structural, glass-box and clear-box. While black-box and white-box are terms that are still in popular use, many people prefer the terms 'behavioural' and 'structural'. Behavioural test design is slightly different from black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged. In practice, it hasn't proven useful to use a single test design method. One has to use a mixture of different methods so that they aren't hindered by the limitations of a particular one. Some call this 'gray-box' or 'translucent-box' test design, but others wish we'd stop talking about boxes altogether. It is important to understand that these methods are used during the test design phase, and their influence is hard to see in the tests once they're implemented. Note that any level of testing (unit testing, system testing, etc.) can use any test design methods. Unit testing is usually associated with structural test design, but this is because testers usually don't have well-defined requirements at the unit level to validate.

164. What kinds of testing should be considered?

Below are the kinds of testing which should be considered:

Black box testing can be considered which is not based on any knowledge of internal design or code. Tests are based on requirements and functionality. White box testing can also be considered which is based on knowledge of the internal logic of an application's code. Tests are based on coverage of code statements, branches, paths, conditions. Unit testing - the most 'micro' scale of testing; to test particular functions or code modules. Unit testing is typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. Not always easily done unless the application has a well-designed architecture with tight code; may require developing test driver modules or test harnesses. incremental integration testing - continuous testing of an application as new functionality is added; requires that various aspects of an application's functionality be independent enough to work separately before all parts of the program are completed, or that test drivers be developed as needed; done by programmers or by testers. integration testing - testing of combined parts of an application to determine if they function together correctly. The 'parts' can be code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems. functional testing - black-box type testing geared to functional requirements of an application; this type of testing should be done by testers. This doesn't mean that the programmers shouldn't check that their code works before releasing it (which of course applies to any stage of testing.) system testing - black-box type testing that is based on overall requirements specifications; covers all combined parts of a system. end-to-end testing - similar to system testing; the 'macro' end of the test scale; involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate. sanity testing or smoke testing - typically an initial testing effort to determine if a new software version is performing well enough to accept it for a major testing effort. For example, if the new software is crashing systems every 5 minutes, bogging down systems to a crawl, or corrupting databases, the software may not be in a 'sane' enough condition to warrant further testing in its current state. regression testing - re-testing after fixes or modifications of the software or its environment. It can be difficult to determine how much re-testing is needed, especially near the end of the development cycle. Automated testing tools can be especially useful for this type of testing. acceptance testing - final testing based on specifications of the end-user or customer, or based on use by end-users/customers over some limited period of time. load testing - testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails. stress testing - term often used interchangeably with 'load' and 'performance' testing. Also used to describe such tests as system functional testing while under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database system, etc. performance testing - term often used interchangeably with 'stress' and 'load' testing. Ideally 'performance' testing (and any other 'type' of testing) is defined in requirements documentation or QA or Test Plans. Usability testing - testing for 'user-friendliness'. Clearly this is subjective, and will depend on the targeted end-user or customer. User interviews, surveys, video recording of user sessions, and other techniques can be used. Programmers and testers are usually not appropriate as usability testers. install/uninstall testing - testing of full, partial, or upgrade install/uninstall processes. recovery testing - testing how well a system recovers from crashes, hardware failures, or other catastrophic problems. failover testing - typically used interchangeably with 'recovery testing' security testing - testing how well the system protects against unauthorized internal or external access, wilful damage, etc; may require sophisticated testing techniques. Compatibility testing - testing how well software performs in a particular hardware/software/operating system/network/etc. environment. Exploratory testing - often taken to mean a creative, informal software test that is not based on formal test plans or test cases; testers may be learning the software as they test it. Ad-hoc testing - similar to exploratory testing, but often taken to mean that the testers have significant understanding of the software before testing it. Context-driven testing - testing driven by an understanding of the environment, culture, and intended use of software. For example, the testing approach for life-critical medical equipment software would be completely different than that for a low-cost computer game. User acceptance testing is done to determine whether the software is satisfactory to an end-user or customer. Comparison testing is about comparing software weaknesses and strengths to competing products. Alpha testing - testing of an application when development is nearing completion; minor design changes may still be made as a result of such testing. Alpha testing is typically done by end-users or others, not by programmers or testers. Beta testing - testing when development and testing are essentially completed and final bugs and problems need to be found before final release. Beta testing is typically done by end-users or others, not by programmers or testers. mutation testing - a

method for determining if a set of test data or test cases is useful, by deliberately introducing various code changes ('bugs') and retesting with the original test data/cases to determine if the 'bugs' are detected. Proper implementation requires large computational resources.

165. Why is it often hard for management to get serious about quality assurance?

Solving problems is a high-visibility process; preventing problems is low-visibility. This is illustrated by an old parable: In ancient China there was a family of healers, one of whom was known throughout the land and employed as a physician to a great lord. The physician was asked which of his family was the most skillful healer. He replied, "I tend to the sick and dying with drastic and dramatic treatments, and on occasion someone is cured and my name gets out among the lords.""My elder brother cures sickness when it just begins to take root, and his skills are known among the local peasants and neighbours." "My eldest brother is able to sense the spirit of sickness and eradicate it before it takes form. His name is unknown outside our home."

166. Why does software have bugs?

1) Miscommunication or no communication - as to specifics of what an application should or shouldn't do (the application's requirements). 2) Software complexity - the complexity of current software applications can be difficult to comprehend for anyone without experience in modern-day software development. Multi-tiered applications, client-server and distributed applications, data
communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity. programming errors - programmers, like anyone else, can make mistakes. 3) Changing requirements (whether documented or undocumented)

- the end-user may not understand the effects of changes, or may understand and request them anyway - redesign, rescheduling of engineers, effects on other projects, work already completed that may have to be redone or thrown out, hardware requirements that may be affected, etc. If there are many minor changes or any major changes, known and unknown dependencies among parts of the project are likely to interact and cause problems, and the complexity of coordinating changes may result in errors. Enthusiasm of engineering staff may be affected. In some fast-changing business environments, continuously modified requirements may be a fact of life. In this case, management must understand the resulting risks, and QA and test engineers must adapt and plan for continuous extensive testing to keep the inevitable bugs from running out of control. 4) Poorly documented code - it's tough to maintain and modify code that is badly written or poorly documented; the result is bugs. In many organizations management provides no incentive for programmers to document their code or write clear, understandable, maintainable code. In fact, it's usually the opposite: they get points mostly for quickly turning out code, and there's job security if nobody else can understand it ('if it was hard to write, it should be hard to read'). 5) software development tools - visual tools, class libraries, compilers, scripting tools, etc. often introduce their own bugs or are poorly documented, resulting in added bugs. Also know about from where do defects and failures in software testing arise?

167. How can new Software QA processes be introduced in an existing organization?

A lot depends on the size of the organization and the risks involved. For large organizations with high-risk (in terms of lives or property) projects, serious management buy-in is required and a formalized QA process is necessary. Where the risk is lower, management and organizational buy-in and QA implementation may be a slower, step-at-a-time process. QA processes should be balanced with productivity so as to keep bureaucracy from getting out of hand. For small groups or projects, a more ad-hoc process may be appropriate, depending on the type of customers and projects. A lot will
depend on team leads or managers, feedback to developers, and ensuring adequate communications among customers, managers, developers, and testers. The most value for effort will often be in (a) requirements management processes, with a goal of clear, complete, testable requirement specifications embodied in requirements or design documentation, or in 'agile'-type environments extensive continuous coordination with end-users, (b) design inspections and code inspections, and (c) post-mortems/retrospectives.

168. How do the companies expect the defect reporting to be communicated by the tester to the

development team. Can the excel sheet template be used for defect reporting. If so what are the common fields that are included? Who assigns the priority and severity of the defect?

To report bugs in excel: S.no, Module Screen/ Section Issue detail, Severity, Priority, Issue status this is how to report bugs in excel sheet and also set filters on the Columns attributes. But most of the companies use the share point process of reporting bugs In this when the project came for testing a module wise detail of project is inserted to the defect management system they are using. It contains following field1) Date 2) Issue brief 3) Issue description (used for developer to regenerate the issue)

4) Issue status (active, resolved, on hold, suspend and not able to regenerate) 5) Assign to (Names of members allocated to project) 6) Priority (High, medium and low) 7) severity (Major, medium and low)

169. What are the tables in test plans and test cases?

Test plan is a document that contains the scope, approach, test design and test strategies. It includes the following:-1) Test case identifier 2) Scope 3) Features to be tested 4) Features not to be tested. 5) Test strategy. 6) Test Approach 7) Test Deliverables 8) Responsibilities. 9) Staffing and Training 10)

Risk and Contingencies 11) Approval While A test case is a noted/documented set of steps/activities that are carried out or executed on the software in order to confirm its functionality/behaviour to certain set of inputs.

170. What are the table contents in test plans and test cases?

Test Plan is a document which is prepared with the details of the testing priority. A test Plan generally includes: 1) Objective of Testing 2) Scope of Testing 3) Reason for testing 4) Timeframe 5) Environment 6) Entry and exit criteria 7) Risk factors involved 8) Deliverables

171. What automating testing tools are you familiar with?

Win Runner, Load runner, QTP , Silk Performer, Test director, Rational robot, QA run.

172. Why did you use automating testing tools in your job?

Automating testing tools are used because of the following reasons:

1) For regression testing 2) Criteria to decide the condition of a particular build

173. How do you plan test automation?

1) Prepare the automation Test plan 2) Identify the scenario 3) Record the scenario 4) Enhance the scripts by inserting check points and Conditional Loops 5) Incorporated Error Handler 6) Debug the script 7) Fix the issue 8) Rerun the script and report the result.

174. Can test automation improve test effectiveness?

Yes, automating a test makes the test process: 1) Fast 2) Reliable 3) Repeatable 4) Programmable 5) Reusable 6) Comprehensive

175. What are the main attributes of test automation?

software test automation attributes : Maintainability - the effort needed to update the test automation suites for each new release Reliability - the accuracy and repeatability of the test automation Flexibility - the ease of working with all the different kinds of automation test ware Efficiency - the total cost related to the effort needed for the automation Portability - the ability of the automated test to run on different environments Robustness - the effectiveness of automation on an unstable or rapidly changing system Usability - the extent to which automation can be used by different types of users

176. Does automation replace manual testing?

There can be some functionality which cannot be tested in an automated tool so we may have to do it manually. Therefore manual testing can never be replaced. (We can write the scripts for negative testing also but it is hectic task).When we talk about real environment we do negative testing manually.

177. How will you choose a tool for test automation?

Choosing of a tool depends on many things like 1) Application to be tested 2) Test environment 3) Scope and limitation of the tool. 4) Feature of the tool 5) Cost of the tool 6) Whether the tool is compatible with your application which means tool should be able to interact with your application 7) Ease of use

178. How you will evaluate the tool for test automation?

We need to concentrate on the features of the tools and how this could be beneficial for our project.

The additional new features and the enhancements of the features will also help.

179. What are main benefits of test automation?

FAST, RELIABLE, COMPREHENSIVE, REUSABLE

180. What could go wrong with test automation?

1. The choice of automation tool for certain technologies 2. Wrong set of test automated

181. How you will describe testing activities?

Testing activities start from the elaboration phase. The various testing activities are preparing the test plan, Preparing test cases, Execute the test case, Log the bug, validate the bug & take appropriate action for the bug, Automate the test cases.

182. What testing activities you may want to automate?

Automate all the high priority test cases which need to be executed as a part of regression testing for each build cycle.

183. Describe common problems of test automation.

The common problems are: 1. Maintenance of the old script when there is a feature change or enhancement 2. The change in technology of the application will affect the old scripts

184. What types of scripting techniques for test automation do you know?

5 types of scripting techniques: Linear Structured Shared Data Driven Key Driven

185. What are principles of good testing scripts for automation?

1) Proper code guiding standards 2) Standard format for defining functions, exception handler etc 3) Comments for functions 4.)Proper error handling mechanisms 5) The appropriate synchronisation techniques.

186. Can the activities of test case design be automated?

As I know it, test case design is about formulating the steps to be carried out to verify something about the application under test. And this cannot be automated. However, I agree that the process of putting the test results into the excel sheet.

187. What are the limitations of automating software testing?

Hard-to-create environments like “out of memory”, “invalid input/reply”, and “corrupt registry entries” make applications behave poorly and existing automated tools can’t force these condition - they simply test your application in “normal” environment.

188. What skills needed to be a good test automator?
1) Good Logic for programming. 2) Strong analytical skills 3) Pessimistic in Nature.

189. How to find that tools work well with your existing system?

1. Discuss with the support officials 2. Download the trial version of the tool and evaluate 3. Get suggestions from people who are working on the tool

190. Describe some problem that you had with automating testing tool

1) The inability of win runner to identify the third party control like infragistics controls 2) The change of the location of the table object will cause object not found error. 3) The inability of the win runner to execute the script against multiple languages

191. What are the main attributes of test automation?

Maintainability, Reliability, Flexibility, Efficiency, Portability, Robustness, and Usability - these are the main attributes in test automation.

192. What testing activities you may want to automate in a project?

Testing tools can be used for : Sanity tests(which is repeated on every build), stress/Load tests(U simulate a large no of users, which is manually impossible) & Regression tests(which are done after every code change)

193. How to find that tools work well with your existing system?

To find this, select the suite of tests which are most important for your application. First run them with automated tool. Next subject the same tests to careful manual testing. If the results are coinciding you can say your testing tool has been performing.

194. **How will you test the field that generates auto numbers of AUT when we click the button 'NEW" in the application?**

We can create a text file in a certain location, and update the auto generated value each time we run the test and compare the currently generated value with the previous one will be one solution.

195. How will you evaluate the fields in the application under test using automation tool?

We can use Verification points (rational Robot) to validate the fields .Ex Using object data, object data properties VP we can validate fields.

196. Can we perform the test of single application at the same time using different tools on the same

machine?

No. The Testing Tools will be in the ambiguity to determine which browser is opened by which tool.

197. What is 'configuration management"?

Configuration management is a process to control and document any changes made during the life of a project. Revision control, Change Control, and Release Control are important aspects of Configuration Management.

198. How to test the Web applications?

The basic difference in web testing is here we have to test for URL's coverage and links coverage. Using Win Runner we can conduct web testing. But we have to make sure that Web test option is selected in "Add in Manager". Using WR we cannot test XML objects.

199. What are the problems encountered during the testing the application compatibility on different

browsers and on different operating systems

Font issues, alignment issues

200. How exactly the testing the application compatibility on different browsers and on different

operating systems is done?
Compatibility testing is a type of software testing used to ensure compatibility of the system/application/website built with various other objects such as other web browsers, hardware platforms, users

201. How testing is proceeded when SRS or any other document is not given?

If SRS is not there we can perform Exploratory testing. In Exploratory testing the basic module is executed and depending on its results, the next plan is executed.

202. How do we test for severe memory leakages?

Severe memory leakages can be tested by using Endurance Testing . Endurance Testing means checking for memory leaks or other problems that may occur with prolonged execution.