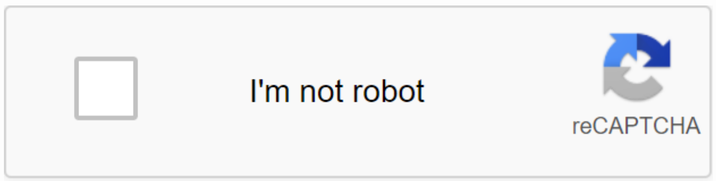

Continue

White box testing pdf

Software testing can be classified into two categories: Black Box Testing is a software test method where the internal structure/design/implementation of the element being tested is not known to the White Box Testing tester is a software test method in which the internal structure/design/implementation of the element being tested is known to the tester. Differences between Black Box Testing vs. White Box Testing: Black Box Testing White Box Testing It is a way of testing software where the internal structure or program or code is hidden and nothing is known about it. It is a way to test the software where the tester has knowledge of the internal structure or code or software program. It's mostly made by software testers. It is mostly done by software developers. No knowledge of implementation is required. Knowledge of implementation is required. It can be referred to as external or external software testing. It is the internal or internal software test. It's a functional test of the software. These are structural software tests. This test can be started based on the requirements specification document. This type of software test starts after the detail design document. No knowledge of programming is required. It is mandatory to have a knowledge of programming. It's the software behavior test. It's the logical test of the software. It is applicable to the highest levels of software testing. It is generally applicable to lower levels of software testing. It's also called a closed test. It is also called as a clear box test. It's less time-consuming. It's more time-consuming. It is not suitable or preferred for testing the algorithm. It is suitable for testing the algorithm. It can be done by trial and error ways and methods. Data domains along with internal or internal boundaries can be better tested. Example: Search for something on google using the keywords Example: for input to control and verify cycles Types of black box Testing: A. Functional tests B. Non-functional tests C. White Box Testing regression test types: A. Path Testing B. Loop Testing C. Condition test Attention reader! Don't stop learning now. Discover all the important concepts of CS Theory for SDE interviews with the Cs Theory Course at a student-friendly price and become industry-ready. Recommended posts: If you like GeeksforGeeks and want to contribute, you can also write an article using contribute.geeksforgeeks.org or send your article to contribute@geeksforgeeks.org. See your article that appears on the geeksforgeeks main page and help other Geeks.Si please improve this article if you find something wrong by clicking the Improve article button below. Improved by : aaqibgahamed, DipankarMajumder White box testing is an approach that allows testers inspect and verify the internal operation of a software system, i.e. code, infrastructure and integrations with external systems. The test of the white box is an essential part of automated build processes in a modern Continuous Integration/Continuous Delivery (CI/CD) development pipeline. White box testing is often referenced in the context of static application security testing (SAST), an approach that automatically checks source code or binary files and provides feedback on bugs and possible vulnerabilities. The white box test provides input and examines the outputs, considering the internal operation of the White Box Testing Pros and Contro Pros 1 code. The ability to achieve full code coverage requires a lot of effort to automate 2. Easy to automate Sensitive to code base changes, automation requires expensive maintenance 3. Reduces communication overhead between testers and developers Cannot test expected functionality that does not exist in codebase 4. It allows continuous improvement of the code and the development procedures It is impossible to test from the point of view of the user Black Box and White Box Testing White box test is often at odds with the black box test, which involves testing an application from the user's point of view without any knowledge of its implementation: White box testing can discover structural problems, hidden errors and problems with specific components. The black box test checks that the system as a whole works as expected. Grey Box Testing White box testing involves a complete knowledge of the internal operation of a system under test and the black box does not imply any knowledge. Testing the gray box, however, is a trade-off: testing a system with partial knowledge of its interior. It is most commonly used in integration tests, end-to-end system tests, and penetration tests. Gray box testing combines developer and tester inputs and can lead to more effective test strategies. Reduces the overhead required to perform functional testing of a large number of user paths, focusing testers on the locations that most likely affect users or results in a flaw. The gray box test combines the advantages of testing black boxes and white boxes: ensuring that tests are performed from the user's point of view, as in black box tests. Leverage internal knowledge to focus on the problems that matter most and to identify and address internal system weaknesses, such as white box testing. In the world of application security testing, the gray box testing approach is called Interactive Application Security Testing (IAST). IAST combines: SAST, which tests the white box by evaluating the static code of the application. Dynamic Application Security Testing (DAST), which runs black box testing, interacting with running applications and identifying and vulnerabilities as an external attacker or user would. Types of White Box Testing White box tests can take several forms: unit tests, tests written as part of application code, which test that each component works as expected. Mutation testing: a type of unit test that controls the solidity and consistency of code by defining make small random changes to your code and see if the tests are still passed. Integration testing: Tests specifically designed to control integration points between

internal components in a software system or integrations with external systems. White box penetration test - an ethical hacker acts as an experienced insider, trying to attack an application based on intimate knowledge of its code and environment. Static code analysis: Automatic identification of vulnerabilities or coding errors in static code, using predefined templates or machine learning analysis. What is the white box test focused on? White box tests can focus on identifying one of the following problems with an application's code: security gaps and vulnerabilities— verifying that security best practices are applied when encoding your application, and whether your code is vulnerable to known security vulnerabilities and threats. Broken or unavailablely structured paths that identify redundant, interrupted, or inefficient conditional logic. Output expected: Perform as many inputs as possible in a function to see if it always returns the expected result. Cycle testing: Controlling individual cycles, concatenated loops, and years cycles to verify efficiency, conditional logic, and proper management of local and global variables. Data flow (DFT) testing: Detect variables and their values as they pass through code to find variables that are not properly initialized, declared but never used or changed incorrectly. Test techniques and code coverage One of the main objectives of the white box test is to cover the source code as fully as possible. Code coverage is a metric that shows the amount of unit tests in an application that controls its functionality. Within code coverage, you can check the amount of logic of an application actually executed and tested by the unit test suite, using concepts such as instruction code coverage, branch code coverage, and path code coverage. These concepts are explained in more detail below. Instruction code coverage is a white box test technique that ensures that all executable statements in your code are executed and tested at least once. For example, if there are several conditions in a block of code, each of which is used for a certain input range, the test must run each input range, to ensure that all lines of code are actually executed. instruction code coverage helps you locate unused statements unused branches missing statement referenced by code and code remained from previous versions. Branch Coverage Branch maps code to branches of conditional logic and ensures that each branch is covered by unit tests. For example, if there are several nested conditional statements: if X then.. if Y then.. A B else if then. C more. D A, C and D are conditional branches, because they occur only if a condition is met. B is an unconditional branch, because it is always executed after A. In a branch branch , the tester identifies all conditional and unconditional branches and writes code to run as many branches as possible. The coverage of the route coverage path concerns linearly independent paths through the code. Testers draw a diagram of the code control flow, such as the following example. Control flowchart used to design tests in a path code coverage approach In this example, there are several possible paths in your code: 1, 2 1, 3, 4, 5, 6, 8 1, 3, 4, 7, 6, 8, and so on. In a path code coverage approach, the tester performs unit tests to run as many paths as possible through the program's control flow. The goal is to identify broken, redundant, or inefficient paths. Imperva Runtime Application Self Protection Runtime Application Self Protection (RASP) completes testing of white boxes and black boxes by adding an extra layer of protection when the application is already in production or in a realistic staging environment. RASP offers the following benefits: It allows you to test applications thoroughly during fast and agile development cycles. Check for unexpected inputs, inspect, and check the system response. It provides analytics and insights into weaknesses and vulnerabilities, helping you respond quickly to attacks. Imperva RASP offers these benefits, keeping applications secure and providing essential feedback to eliminate any additional risks. It requires no code changes and integrates easily with existing applications and DevOps processes, protecting you from known, zero-day attacks. In addition, Imperva provides tiered protection to ensure that websites and applications are available, easily accessible, and secure. The Imperva application security solution includes: DDoS security: maintaining uptime in all situations. Prevent any type of DDoS attack, of any size, from preventing access to the website and network infrastructure. CDN: Improves website performance and reduces bandwidth costs with a CDN designed for developers. Cache static resources to the limit, accelerating dynamic APIs and websites. WAF Cloud: Allows legitimate traffic and prevents invalid traffic. Save your applications to the margins with an enterprise-class cloud WAF. WAF Gateway: Keep applications and APIs safe within the network with Imperva Gateway WAF. Attack analysis: Myth and respond to real security threats efficiently and accurately with information that can be used on all levels of defense. Account capture protection: Use a purpose-based detection process to identify and defend against attempts to detect user accounts for malicious purposes. API Security: Protects APIs by ensuring that only desired traffic can access the API endpoint, as well as detecting and blocking vulnerability exploits. Bot Manager: Analyzes bot traffic to detect anomalies, identifies invalid bot behavior, and validates it through verification mechanisms that do not affect user traffic. Traffic. Traffic.