Line Coverage - Improving Code Quality with Line Coverage

1. **Scenario** - Improving Code Quality with Line Coverage
   1. Context - Imagine a development team working on a web application that includes a feature for user authentication. The team has implemented several test cases to ensure the functionality works as intended. However, they notice an increase in bug reports related to user login failures.
   2. Initial Testing - The team has written test cases that cover the following scenarios:
      1. Successful login with valid credentials.
      2. Failed login with invalid credentials.

After running their test suite, they check their line coverage report and find that only **60%** of the lines in the authentication module were executed during testing.

* 1. **Analysis**

The line coverage report indicates that certain critical paths in the authentication logic were never reached during testing. For instance, there may be lines handling specific error conditions, such as:

* + 1. Account lockout after multiple failed attempts.
    2. Password reset functionality
  1. **Action Steps**

To enhance code quality and reduce bugs, the team decides to:

* + 1. **Add Test Cases:** Implement additional test cases to cover scenarios such as:
* Multiple failed login attempts leading to account lockout.
* Validating password reset requests.
  + 1. **Re-run Tests:** After adding these tests, they re-run their suite and find that line coverage has increased to **85%**
  1. **Outcome**

With improved line coverage, the team identifies and fixes several bugs related to untested paths in their code. The additional tests not only enhance the robustness of their application but also lead to a reduction in user-reported issues.

* 1. **Conclusion**

This scenario illustrates how line coverage can guide software teams in identifying untested areas within their codebase. By focusing on increasing line coverage through comprehensive testing, teams can significantly improve code quality and performance, ultimately leading to a more reliable product. This metric helps developers identify parts of the code that are not being tested, thus guiding them to improve their test cases for better coverage and reliability.

* 1. Question:
     1. If you have a codebase with 100 lines and your tests execute 80 of those lines, the line coverage would be \_\_\_\_\_\_\_\_\_\_\_ ?
     2. **What does line coverage measure in software testing?**

A) The number of test cases executed

B) The percentage of lines of code executed by tests

C) The number of bugs found during testing

D) The time taken to execute tests  
**Correct Answer:** B

* + 1. **If a codebase has 150 executable lines and your tests execute 120 of those lines, what is the line coverage percentage?**

A) 70%

B) 80%

C) 90%

D) 100%  
**Correct Answer:** B

* + 1. **Which of the following statements is true regarding line coverage?**

A) High line coverage guarantees that all bugs are found.

B) Line coverage can be improved by adding more test cases.

C) Line coverage does not consider the quality of the test cases.

D) All of the above  
**Correct Answer:** D

* + 1. **Which of the following scenarios could lead to high line coverage but still have untested critical paths?**

A) Writing tests for all public methods only

B) Having multiple test cases that cover different inputs

C) Not testing error handling and edge cases

D) Both A and C  
**Correct Answer:** D

* 1. True/False Questions
     1. Line coverage is the only metric that should be used to evaluate code quality.  
        **Correct Answer:** False
     2. Achieving 100% line coverage means that all potential bugs in the code have been identified and fixed.  
        **Correct Answer:** False
  2. **Interactive Questions / Q&A**
     1. What are some limitations of relying solely on line coverage as a metric for code quality?
     2. How can developers improve line coverage in their projects? List at least three strategies.
  3. Counter Question: Can high line coverage indicate sufficient testing?
  4. Scenario based explanation of a Transaction Processing System
  5. Context: A development team is tasked with creating a payment processing system that handles credit and debit transactions. To ensure reliability, the team aims for high line coverage and achieves **95%** coverage across their codebase.
  6. Initial Testing - The team has written tests that cover:
  + Successful credit transactions.
  + Successful debit transactions.
  + Basic error handling for invalid inputs.

After running their test suite, they are pleased with the high line coverage percentage. However, they overlook several critical aspects of the application.

* 1. **Key Issues**
     1. Lack of Edge Case Testing: The tests do not address edge cases such as:
  + Transactions exceeding account limits.
  + Handling network failures during transaction processing.
  + Invalid data types (e.g., strings instead of numbers).
    1. No Assertions in Tests: Some tests are written merely to execute lines of code without any assertions to validate outcomes. For example, a test might call a method to process a transaction but does not check whether the transaction was successful or if the correct error message was returned.
    2. Ignoring Business Logic: The tests focus on executing methods rather than verifying that business rules are enforced correctly. For instance, there may be rules about transaction limits based on account types that are not tested at all.
    3. Critical Paths Untested: The coverage report shows that many lines are executed, but essential paths—like those handling exceptions or specific business logic—remain untested.
  1. **Outcome** - When the application goes live, users encounter several issues:
* Transactions fail without appropriate error messages due to untested exception handling.
* Users exceed transaction limits without any checks in place, leading to financial discrepancies.
* The application experiences crashes during network failures because those scenarios were never tested.

1. **Conclusion**

In this scenario, despite achieving high line coverage, the lack of meaningful tests and failure to address critical business logic resulted in a flawed and unreliable application. This illustrates that line coverage alone cannot guarantee comprehensive testing; it must be complemented by well-designed test cases that validate functionality and business requirements effectively.