



Testing Fundamentals

Testing Types

| | |
|----------------------------|----------------------------------------------------------------------------------------------|
| Unit Testing | Tests individual components or functions in isolation. |
| Integration Testing | Tests the interaction between different components. |
| System Testing | Tests the entire system to ensure it meets requirements. |
| Acceptance Testing | Tests the system from the user's perspective to validate it meets their needs. |
| Regression Testing | Retests previously tested components after changes to ensure no new issues were introduced. |
| Performance Testing | Tests the system's responsiveness, stability, and scalability under various load conditions. |

Test-Driven Development (TDD)

1. **Write a failing test** before writing any code.
2. **Write the minimum amount of code** to pass the test.
3. **Refactor** the code to improve its structure and maintainability.

TDD promotes writing clean, testable code and ensures that all code is covered by tests.

Test Automation

Automated tests can be run repeatedly and consistently, saving time and reducing the risk of human error.

Popular tools include Selenium, JUnit, pytest, and Cypress.

Benefits include faster feedback, improved test coverage, and reduced testing costs.

Debugging Techniques

Debugging Strategies

| | |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Print Statements | Insert print statements to display variable values and track the program's execution flow. <code>print(f'Value of x: {x}')</code> |
| Debuggers | Use debuggers to step through code, inspect variables, and set breakpoints. Examples: <code>pdb</code> (Python), <code>gdb</code> (C/C++), Chrome DevTools (JavaScript). |
| Logging | Implement logging to record events, errors, and warnings for later analysis. Example (Python): <pre>import logging logging.basicConfig(level=logging.DEBUG) logging.debug('This is a debug message')</pre> |
| Code Reviews | Have peers review your code to identify potential bugs and improve code quality. |
| Rubber Duck Debugging | Explain the code to an inanimate object (e.g., a rubber duck) to help clarify your thinking and identify errors. |
| Divide and Conquer | Isolate the problem by systematically eliminating sections of code until the bug is found. |

Common Debugging Tools

- **IDEs (Integrated Development Environments):** Provide built-in debugging tools, code completion, and other features.
- **Debuggers:** Standalone tools for stepping through code and inspecting variables.
- **Linters:** Static analysis tools that identify potential code quality issues and bugs.

Analyzing Stack Traces

A stack trace shows the sequence of function calls that led to an error. Use it to identify the source of the error and understand the program's execution path.

Key information includes function names, line numbers, and file names.

Assertion and Error Handling

Assertions

| | |
|-------------------------|----------------------------------------------------------------------------------------------------------|
| Purpose | Verify assumptions in code during development. If an assertion fails, it indicates a bug. |
| Example (Python) | <pre>def divide(a, b): assert b != 0, 'Cannot divide by zero' return a / b</pre> |
| Usage | Use assertions to check preconditions, postconditions, and invariants. |

Exception Handling

| | |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | Handle unexpected errors gracefully to prevent program crashes. Use <code>try-except</code> blocks to catch and handle exceptions. |
| Example (Python) | <pre>try: result = 10 / 0 except ZeroDivisionError as e: print(f'Error: {e}')</pre> |
| Best Practices | Catch specific exceptions, log errors, and provide informative error messages. |

Error Reporting

| |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| Implement robust error reporting mechanisms to capture and log errors in production environments. This helps in identifying and fixing issues quickly. |
| Tools like Sentry, Rollbar, and Bugsnag can be used to track and manage errors. |

Advanced Testing Topics

Mocking

| | |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Definition | Creating simulated objects or functions to isolate and test specific parts of the code. This allows you to test in isolation without dependencies. |
| Example (Python) | <pre>from unittest.mock import Mock # Create a mock object mock_obj = Mock() # Set a return value for a method mock_obj.some_method.return_ value = 42 # Use the mock object in tests result = mock_obj.some_method() assert result == 42</pre> |

Fuzzing

| | |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------|
| Definition | A testing technique that involves feeding invalid, unexpected, or random data to a program to identify vulnerabilities and bugs. |
| Tools | AFL (American Fuzzy Lop), libFuzzer, and Peach Fuzzer. |

Static Analysis

| | |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------|
| Definition | Analyzing source code without executing it to identify potential errors, security vulnerabilities, and code quality issues. |
| Tools | SonarQube, FindBugs, and ESLint. |