Current Position	Current position is 2
Guide URL	https://www.browserstack.com/guide/static-testing
Keywords	static testing
Secondary keywords	- static testing in software testing - static testing techniques types - types of static testing
Reference	- https://www.cprime.com/resources/blog/static-testing-what-you-need-to-know/ - https://www.geeksforgeeks.org/software-engineering/software-testing-static-testing/ - https://www.techtarget.com/searchsoftwarequality/definition/static-testing - https://testsigma.com/blog/static-testing/
TOC	● Introduction ● What is Static Testing? ● Why is Static Testing required? ● Types of Static Testing ○ Manual Methods of Static Testing ○ Automation Method of Static Testing ● When is Static Testing required? ● What components are tested in Static Testing? ● Static Testing Techniques ○ 1. Review ○ 2. Static Analysis ● How is Static Testing performed? ● Who Performs Static Testing? ● Static Testing Tools ● Static Testing vs Dynamic Testing ● Real-World Examples of Static Testing ● Advantages of Static Testing ● Disadvantages of Static Testing ● Best Practices for Static Testing ● Conclusion
SEO Title	What is Static Testing?
SEO Recommendations	- Content can be trimmed - Cover these types of static testing (Software Inspection, Structured Walkthroughs, Technical Reviews) - Can outrank geeksforgeeks
Meta Description	Learn the importance of static testing in software development to catch defects early and improve code quality.

Related Posts	
Guide Banner	
Product	Code Quality
Image Link	

Static testing is a verification technique that examines code, design, and documentation without executing the program.

Importance of Static Testing
It enables early detection of errors, reducing rework and cost. Static testing helps ensure better quality and compliance with coding standards by analyzing development artifacts before runtime.

Key aspects of static testing:

- **Early defect detection:** Identifies issues early in the lifecycle, saving time and effort later.
- **No code execution:** Analyzes source code and documents without running the application.
- **Focus on artifacts:** Reviews requirements, design, source code, test plans, and more.
- **Verification, not validation:** Ensures the product is built correctly, aligning with specifications.
- **Various techniques:** Includes informal reviews, walkthroughs, and formal inspections.
- **Tools and automation:** Static analysis tools automatically flag code issues, security flaws, and style violations.

Static Testing Tools

- **BrowserStack Code Quality Tool:** Imports code from remote repos to detect anti-patterns, vulnerabilities, and design issues.
- **Checkstyle:** Java-specific tool that enforces coding standards and style conventions.
- **SourceMeter:** Supports multiple languages with in-depth analysis of maintainability and complexity.
- **Soot:** Java bytecode analysis framework used for control flow and optimization.
- **Lint:** Detects bugs and coding issues in C/C++ codebases.
- **SonarQube:** Multi-language code quality and security analyzer with CI/CD integration.
- **PMD:** Flags unused variables, empty catch blocks, and inefficient object use.
- **FindBugs:** Java tool that identifies runtime bug patterns like null dereferences and race conditions.

This article talks about the fundamentals of static testing, its importance in software development, key techniques, real-world examples, and the tools that help ensure code quality before execution.

What is Static Testing in Software Testing?

Static testing is a **software testing technique** that examines code, documentation, and design artifacts **without executing the program**. Conducted in the early phases of the software development lifecycle (SDLC), static testing helps detect **errors, gaps, and inconsistencies** before code is run, making it a cost-effective quality assurance practice.

This method primarily involves **reviews, walkthroughs, and inspections** to ensure that all project deliverables meet the required standards and specifications.

Documents reviewed during static testing include:

- Business Requirement Documents (BRD) and Customer Requirement Specifications
- Functional Requirements and Software Requirement Specifications (SRS)
- User Stories and Use Case Documents
- Prototypes and UI/UX Design Specifications
- High-Level and Low-Level Design Documents
- Test Artifacts such as Test Plans, Test Strategies, Test Scenarios, Test Cases, Test Data, and Traceability Matrix.

Static testing is an essential part of **early defect detection**. It reduces rework and improves the overall efficiency of the testing process.

Read More: Test Plan vs Test Case: Core Differences

Why is Static Testing required?

Types of Static Testing

Static testing is broadly categorized into **manual** and **automated** methods.

1. Manual Methods of Static Testing

Manual static testing involves reviewing project artifacts without executing the code. These reviews are typically carried out by team members such as architects, designers, reviewers, and managers.

Key types:

- **Inspections**: A formal review process led by a moderator. It includes pre-defined roles (author, reviewer, manager), scheduled meetings, issue logs, and follow-ups. The goal is to detect defects and improve documentation quality.
- **Walkthroughs**: A semi-formal review led by the author to explain documentation and gather feedback. It promotes shared understanding, especially for high-level docs like requirement specs.
- **Technical Reviews**: Conducted by technical experts or moderators to validate technical accuracy, suggest improvements, and align the team on design concepts.
- **Informal Reviews**: Peer reviews without structured roles or documentation. Feedback is shared informally and implemented where relevant.

Also Read: Static testing – Tools and Techniques

2. Automation Method of Static Testing

Automated static testing involves **static code analysis** using tools, typically performed by developers.

Static Analysis Techniques:

- **Control Flow Analysis**: Examines logical execution paths using control flow graphs to validate function logic and process integrity.
- **Data Flow Analysis**: Checks variable definitions, usage, and data structure integrity without code execution.
- **Failure Analysis**: Identifies potential design flaws, unexpected behavior, and module-level failures.
- **Interface Analysis**: Reviews interfaces between modules or external systems to validate integration and communication logic.

Example Tools:

- **PyCharm**: A Python IDE with built-in static analysis and debugging features. The Space plugin supports code reviews and collaboration.
- **Checkstyle**: Used for enforcing coding standards in Java.
- **SourceMeter**: Performs in-depth source code analysis across multiple languages.

Also Read: Maintainability Testing – Importance, How to and Best Practices

When is Static Testing required?

Static testing is performed before the testing phase, early in the Software Development Life Cycle (SDLC), before dynamic testing begins.

- Helps detect defects in code and documentation that are hard to catch during runtime.
- Ensures adherence to coding standards and best practices.
- Identifies security vulnerabilities and potential cyber attack risks.
- Enables early mitigation planning for technical and security risks.
- Automated static analysis reduces testing time and overall project costs.

Read More: Differences between Black Box Testing and White Box Testing

Who Performs Static Testing?

Static testing is carried out by multiple stakeholders across the software development process:

- **Developers**: Review source code for syntax errors, logic flaws, and coding standard violations.
- **Testers/QA Engineers**: Analyze test cases, plans, and requirement documents for gaps and inconsistencies.
- **Business Analysts**: Validate requirement documents against business goals.
- **Designers/Architects**: Review design documents and technical specifications.
- **Project Managers/Moderators**: Oversee formal review processes like walkthroughs and inspections.

Static Testing Tools

Static Testing vs Dynamic Testing

Here are the key differences between static and dynamic testing:

Parameter	Static Testing	Dynamic Testing
Purpose	Evaluates documents, plans, specifications, and source code to identify issues.	Tests the actual behavior and functionality of the application under test (AUT).
Process Type	Part of the verification phase (before development).	Part of the validation phase (after development).
Code Execution	Performed without executing the code.	Requires code execution.
Timing of Defect Detection	Identifies defects early, reducing time and cost.	Detects issues post-development, often costlier to fix.
Techniques Used	Reviews, walkthroughs, inspections.	Unit testing, integration testing, system testing, acceptance testing.

Goal	To prevent defects before they occur.	To detect and fix existing defects.
Quality Assurance Approach	Achieved through early reviews and expert feedback.	Achieved through extensive functional and performance testing.
Participants	Involves business analysts, developers, designers, managers.	Involves QA testers and developers.

Real-World Examples of Static Testing

Static testing is widely adopted in real-world scenarios to catch defects early, reduce rework, and ensure better software quality before code execution begins. Here are some common examples:

- **Code Review in Agile Teams:** Developers review each other's code during pull requests to catch logic flaws, security issues, or violations of coding standards early.
- **Requirements Review:** Business analysts and QA leads assess requirement documents for accuracy, clarity, and alignment with business goals.
- **Design Document Review:** Architects evaluate design specs for scalability, maintainability, and adherence to best practices.
- **Test Plan Review:** QA teams verify test plans and cases to ensure full coverage and traceability back to requirements.
- **Static Code Analysis:** Automated tools like SonarQube, ESLint, or Checkstyle scan source code to detect bugs, vulnerabilities, and code quality issues without running the code.

Advantages of Static Testing

Disadvantages of Static Testing

Best Practices for Static Testing

Conclusion