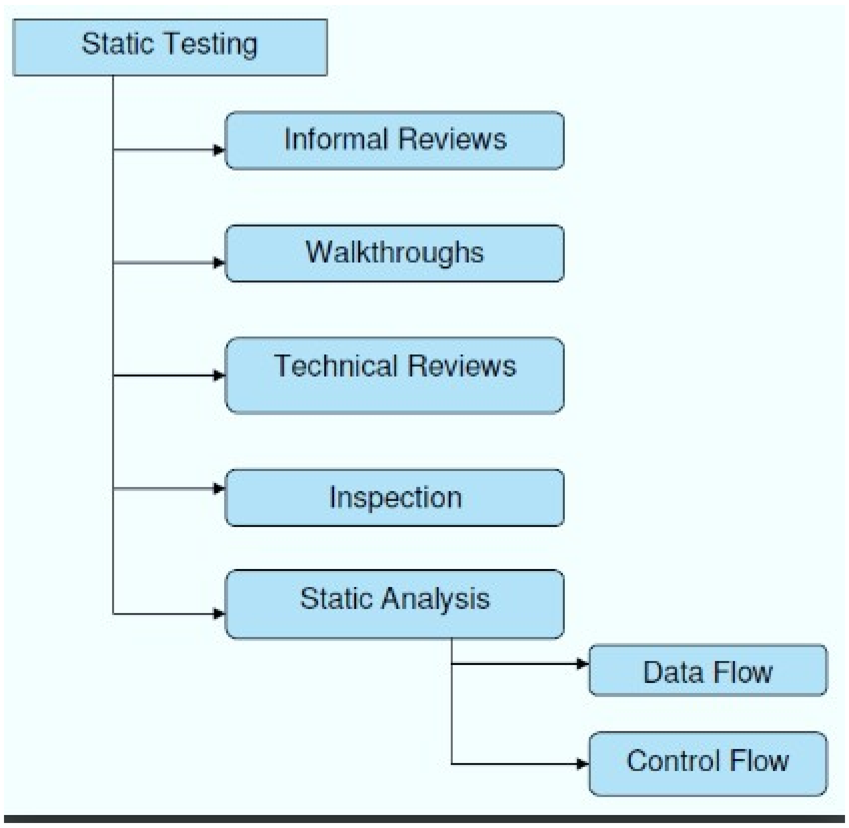
## What is Static Testing?

**Static Testing** is a software testing technique which is used to check defects in software application without executing the code. Static testing is done to avoid errors at an early stage of development as it is easier to identify the errors and solve the errors. It also helps finding errors that may not be found by Dynamic Testing.

The two main types of static testing techniques are

**Manual examinations**: Manual examinations include analysis of code done manually, also known as **REVIEWS.**

**Automated analysis using tools:** Automated analysis are basically static analysis which is done using tools.



## What is Testing Review?

A review in Static Testing is a process or meeting conducted to find the potential defects in the design of any program. Another significance of review is that all the team members get to know about the progress of the project and sometimes the diversity of thoughts may result in excellent suggestions.

Reviews can further be classified into four parts:

●Informal reviews (Informal reviews are casual, unstructured assessments or feedback sessions typically occurring spontaneously between peers or stakeholders, with minimal or no formal documentation )

●Walkthroughs (A walkthrough is a step-by-step demonstration or explanation of a process, task, or system, used to guide users or review work)

●Technical review (To find defects and opportunities for improvement in a technical document or product, like source code or design specifications. )

●Inspections (Software inspection is a formal, static review process to identify defects in software artifacts like code or documents before the software is executed )

**Types of defects which can be easier to find during static testing are:**

●Deviations from standards

●Non-maintainable code

●Design defects

●Missing requirements

●Inconsistent interface specifications

**Unreachable Code sample:**

```
public class UnreachableExample {

   public static void main(String[] args) {

      System.out.println("This will be printed.");

      return; // The method execution terminates here.

      System.out.println("This will never be printed (unreachable code)."); // Compiler error

   }

}


public class UnreachableExample {
    public static void main(String[] args) {
        for (;;) { // Infinite loop
            System.out.println("Inside the loop.");
            break; // Exits the loop unconditionally.
            System.out.println("This will never be printed (unreachable code)."); // Compiler error
        }
    }
}
```

## Coding Standard Example:

**1. Formatting and Style:**

**Indentation:** Consistent use of spaces or tabs for indentation (e.g., 4 spaces per level).

**Brace Style:** Placement of curly braces (e.g., Allman style, K&R style).

**Line Length:** Limiting the maximum number of characters per line for readability.

**Whitespace:** Consistent use of spaces around operators, in function signatures, and between code blocks.

**2. Naming Conventions:**

**Variables:** Using descriptive names, often following camelCase (e.g., firstName) or snake_case (e.g., first_name).

**Functions/Methods:** Using descriptive names, often following camelCase (e.g., calculateTotal).

**Classes/Objects:** Using PascalCase (e.g., UserProfile).

**Constants:** Using all uppercase with underscores (e.g., MAX_RETRIES).

**3. Documentation:**

**Comments:** Using clear and concise comments to explain complex logic, non-obvious design choices, or potential pitfalls.

**Docstrings/Javadocs:** Providing comprehensive documentation for functions, classes, and modules, including parameters, return values, and exceptions.

**File Headers:** Including information like author, version, and copyright details.

**4. Code Structure and Design:**

**Modularity:** Breaking down code into smaller, reusable functions or modules.

**Error Handling:** Implementing robust error handling mechanisms (e.g., try-catch blocks, specific error codes).

**Security Considerations:** Adhering to secure coding practices to prevent vulnerabilities (e.g., input validation, avoiding insecure functions).

**Accessibility:** Ensuring code is written with accessibility in mind, especially for UI components.

**What is Tested in Static Testing?**

In Static Testing, following things are tested

●Unit Test Cases

●Business Requirements Document (BRD)

●Use Cases (A use case is a description of how a user interacts with a system to achieve a specific goal)

●System/Functional Requirements

●Prototype

●Prototype Specification Document

●Test Data

●Traceability Matrix Document

●User Manual/Training Guides/Documentation

●Test Plan Strategy Document/Test Cases

●Automation/Performance Test Scripts

**<u>1. Reviews:</u>**

Reviews are a broad category of static testing involving manual examination of documents to find anomalies.

**Informal Review:** This involves minimal documentation and typically includes other developers informally reviewing code or documentation.

**Walkthrough:** The author presents the work product (e.g., code, design) to a team, explaining it step-by-step. The team provides feedback and identifies potential issues.

**Peer Review:** Developers within a team examine each other's code or documentation to find defects and suggest improvements.

**Inspection:** A formal and structured review process led by a moderator. It involves a detailed examination of the work product by trained inspectors who follow a defined process to identify defects.

### 2. Static Code Analysis:

This technique uses automated tools to analyze source code without executing it.

**Data Flow Analysis:** Examines how data is defined, used, and modified throughout the program to detect issues like uninitialized variables or dead code.

**Control Flow Analysis:** Analyzes the execution paths within the code to identify potential problems like infinite loops or unreachable code.

**Coding Standards Compliance:** Checks if the code adheres to predefined coding conventions and style guidelines.

**Security Vulnerability Scanning:** Identifies potential security weaknesses and vulnerabilities in the code.

### 3.Other Static Techniques:

**Use Case Requirements Validation:** Ensures that all possible end-user actions are properly defined and covered in the requirements.

**Functional Requirements Validation:** Confirms that all necessary functional requirements for the software are identified and correctly specified.

**Architecture Review:** Analyzes the system's architecture to ensure it meets business-level processes and design principles.

**Field Dictionary Validation:** Reviews user interface fields and their definitions for consistency and correctness.

## Benefits of Static Testing:

**Below are some of the benefits of static testing:**

**Early defect detection:** Static testing helps in early defect detection when they are most easy and cost-effective to fix.

**Prevention of common issues:** Static testing helps to fix common issues like syntax errors, null pointer exceptions, etc. Addressing these issues early in development helps the teams to avoid problems later.

**Improved code quality:** Static testing helps to make sure that the code is easy to maintain and well-structured. This leads to a higher quality code.

**Reduced costs:** Early bug detection in static testing helps to fix them early in the development thus saving time, effort, and cost.

**Immediate feedback:** Static testing provides immediate evaluation and feedback on the software during each phase while developing the software product.

**Helps to find exact bug location:** Static testing helps to find the exact bug location as compared to dynamic testing.

**Static Testing Tools:**

- Checkstyle
- Soot
- SourceMeter
- Lint
- SonarQube