# Static analysis

## What is Static analysis

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Static analysis** is a method of examining software **without executing it**.

It means analyzing the source code, bytecode, or even documentation to detect issues **before running the program**.

## Key points about static analysis:

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

* ✅ **No execution** → it checks the code itself, not how it behaves when running.
* ✅ **Automated tools** can scan code for potential bugs, security vulnerabilities, coding standard violations, or complexity issues.
* ✅ Helps in **early defect detection** → cheaper to fix than after testing or deployment.

## Static Analysis tools

### 🔹What is Static Analysis Tools

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Static Analysis Tools** are software programs that automatically check **code or documentation without executing it**, to detect:

* Bugs
* Security vulnerabilities
* Style/coding standard violations
* Ambiguity or incompleteness in requirements/tests

They save time by finding problems **early**, before testing or deployment.

### 🔹 How They Work

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

1. **Parse the artifact** (source code, requirements, test scripts, API docs).
2. **Apply rules or patterns** (e.g., "flag ambiguous words", "variables must be camelCase", "no buffer overflows").
3. **Report findings** (warnings, errors, severity levels).
4. **Suggest fixes** (some tools even auto-correct).

### 🔹 Types of Static Analysis Tools

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

1. **Code Quality & Style Checkers**
   * ESLint (JavaScript/TypeScript)
   * Pylint (Python)
   * Checkstyle (Java)
2. **Security-Focused Tools**
   * Bandit (Python security)
   * Fortify (enterprise security)
   * SonarQube (bugs + vulnerabilities)
3. **Safety-Critical Tools**
   * Polyspace (for aerospace/automotive, checks MISRA-C compliance)
   * Coverity (memory leaks, concurrency issues)
4. **Documentation/Requirements Analysis**
   * QVscribe (flags ambiguous requirements)
   * INCOSE rules/checklists
   * Natural language analyzers for clarity & consistency

### 🔹 Example in Practice

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Let’s say you have this C code:

char name[10];

strcpy(name, "ThisIsAVeryLongName");

* A **static analysis tool** like **Cppcheck** or **Coverity** scans the code.
* It sees that "ThisIsAVeryLongName" is longer than 10 characters.
* It flags a **buffer overflow risk** → without even running the program.

### real-world static analysis tools

Perfect 👍 — here’s a breakdown of **real-world static analysis tools** used in different domains, with examples of what they’re good for:

#### 🔹 General-Purpose Code Quality & Security

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

1. **SonarQube**
   * Supports: Java, C#, Python, C/C++, JavaScript, and many more.
   * **Use case:**
     + Finds code smells, bugs, and security vulnerabilities.
     + Tracks technical debt (how much effort is needed to fix poor code).
   * Example: In a **banking application**, SonarQube may flag a SQL query built with string concatenation as a **critical security risk**.
2. **ESLint** (for JavaScript/TypeScript)
   * Focuses on style, best practices, and preventing common mistakes.
   * **Use case:**
     + Detects unused imports in React.
     + Ensures variables are declared properly (let vs. var).
   * Example: In a web app, ESLint can warn if you forget to handle a Promise properly.

#### 🔹 Security-Focused

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

1. **Fortify (by Micro Focus)**
   * Scans code for **security vulnerabilities**.
   * **Use case:**
     + Identifies cross-site scripting (XSS) in web apps.
     + Detects insecure cryptographic usage.
   * Example: In a government portal, Fortify may flag the use of weak hashing (MD5) for storing passwords.
2. **Bandit (Python)**
   * Python-specific static analyzer.
   * **Use case:**
     + Finds use of eval() (dangerous if user input is passed).
     + Flags insecure random number generators.
   * Example: In a machine learning project, Bandit can highlight hardcoded API keys in the code.

#### 🔹 Safety-Critical Domains

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

1. **Polyspace (by MathWorks)**
   * Used for embedded systems (C/C++).
   * **Use case:**
     + Proves absence of runtime errors (like divide by zero).
     + Verifies MISRA-C compliance for automotive software.
   * Example: In an **aircraft control system**, Polyspace ensures no buffer overflow or null pointer dereference exists.
2. **Coverity (by Synopsys)**
   * Enterprise-level static analysis tool.
   * **Use case:**
     + Detects concurrency issues (race conditions).
     + Checks memory leaks in C/C++.
   * Example: In **medical devices**, Coverity can catch a race condition where two threads try to update the same patient data at once.

#### 🔹 Open-Source / Lightweight

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

1. **Pylint (Python)**
   * Checks code style and simple bugs.
   * **Use case:**
     + Detects unused imports, wrong indentation, naming violations.
   * Example: In a **Django web project**, Pylint warns if you define a variable but never use it.
2. **Cppcheck (C/C++)**
   * Lightweight tool for C/C++.
   * **Use case:**
     + Detects memory leaks and undefined behavior.
   * Example: In an **IoT device firmware**, Cppcheck can flag dereferencing of a possibly NULL pointer.

#### 🔹 Documentation & Requirements Analysis Tools

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

These check **requirements specs, test scripts, or API docs** for ambiguity, incompleteness, and consistency.

1. **QVscribe** (by QRA Corp)

* Focus: Requirements quality.
* Use case:
  + Flags ambiguous words (*fast, user-friendly, adequate*).
  + Detects incomplete or unverifiable requirements.
* Example: In a government RFP, QVscribe may flag *“system must be reliable”* as unverifiable and suggest measurable alternatives.

1. **INCOSE Checklists** (Industry standard)

* Focus: Requirements writing guidelines.
* Use case:
  + Detects passive voice, missing actors (*“data shall be encrypted” → by whom?*).
  + Ensures requirements are clear, consistent, testable.
* Example: In an aerospace project, a requirement written as *“doors shall close quickly”* would be flagged for ambiguity.

#### 🔹 DevOps / CI Integration Tools

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

These are designed to plug into CI/CD pipelines for automated static analysis.

1. **CodeClimate**

* Focus: Maintainability & code health.
* Use case:
  + Rates code complexity, duplication, and test coverage.
* Example: In a startup’s CI pipeline, CodeClimate may flag a method that’s too complex and suggest refactoring.

1. **DeepSource**

* Focus: Automated code review.
* Use case:
  + Checks security, performance, and anti-patterns during CI.
* Example: In a fintech app, DeepSource might warn against using pickle (unsafe deserialization in Python).

#### 🔹 Mobile Development Tools

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Specialized analyzers for Android/iOS apps.

1. **Lint (Android Lint)**

* Focus: Android code quality.
* Use case:
  + Detects unused resources, missing translations, layout performance issues.
* Example: In an Android app, Lint may flag that a string resource exists in English but not in Arabic, causing missing UI text.

1. **Infer (by Meta/Facebook)**

* Focus: Java, C, Objective-C, C++ (used in mobile apps).
* Use case:
  + Detects null pointer exceptions, memory leaks, resource leaks.
* Example: In a mobile banking app, Infer can flag that a file stream wasn’t properly closed, which could cause crashes.

#### 🔹 Academic / Research-Oriented Tools

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Used for advanced program verification or research.

1. **Frama-C**

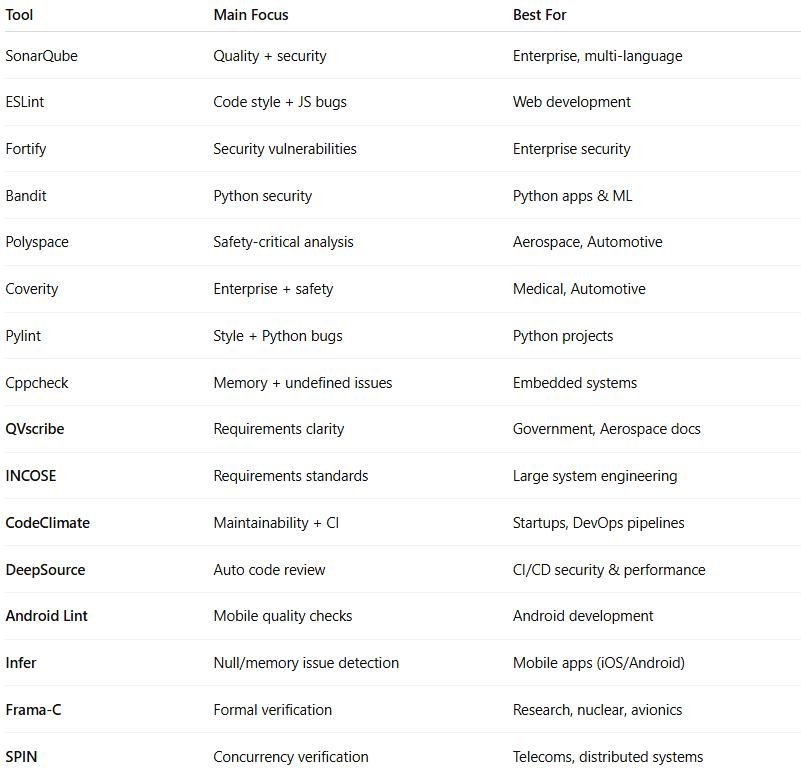
* Focus: C programs, formal verification.
* Use case:
  + Proves correctness of safety-critical code using mathematical methods.
* Example: In nuclear plant control software, Frama-C can mathematically prove that no integer overflow will ever occur.

1. **SPIN Model Checker**

* Focus: Concurrent systems, protocol verification.
* Use case:
  + Checks for deadlocks, race conditions in system models.
* Example: In a telecom protocol, SPIN can prove that two processes won’t get stuck waiting forever.

#### 🔹 ✅ Expanded Summary Table

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**



| **Tool** | **Main Focus** | **Best For** |
| --- | --- | --- |
| SonarQube | Quality + security | Enterprise, multi-language |
| ESLint | Code style + JS bugs | Web development |
| Fortify | Security vulnerabilities | Enterprise security |
| Bandit | Python security | Python apps & ML |
| Polyspace | Safety-critical analysis | Aerospace, Automotive |
| Coverity | Enterprise + safety | Medical, Automotive |
| Pylint | Style + Python bugs | Python projects |
| Cppcheck | Memory + undefined issues | Embedded systems |
| **QVscribe** | Requirements clarity | Government, Aerospace docs |
| **INCOSE** | Requirements standards | Large system engineering |
| **CodeClimate** | Maintainability + CI | Startups, DevOps pipelines |
| **DeepSource** | Auto code review | CI/CD security & performance |
| **Android Lint** | Mobile quality checks | Android development |
| **Infer** | Null/memory issue detection | Mobile apps (iOS/Android) |
| **Frama-C** | Formal verification | Research, nuclear, avionics |
| **SPIN** | Concurrency verification | Telecoms, distributed systems |

### ✅ Summary:

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**  
Static Analysis Tools = automated checkers that analyze **without execution**, applying **rules, standards, and best practices** to ensure quality, security, and clarity.

## Examples:

### Code

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

* Detecting unused variables, unreachable code, or memory leaks.
* Finding security issues like SQL injection risks or buffer overflows.
* Ensuring compliance with coding standards (e.g., MISRA C for automotive software).

👉 **Opposite of static analysis** = **Dynamic analysis**, where the program is executed and observed (e.g., during testing).

Got it 👍 — let me walk you through **several detailed examples of static analysis** so you can see how it works in practice:

#### 1. Detecting Unused Variables

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Imagine this C code:



int add(int a, int b) {

int result = a + b;

int temp; // unused variable

return result;

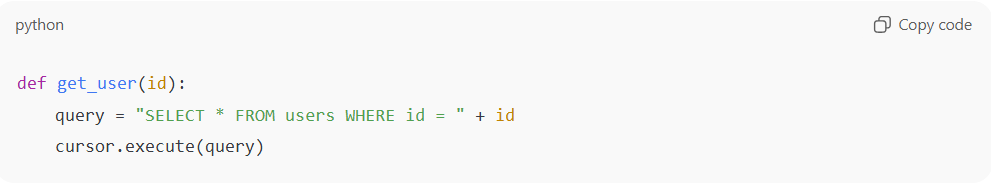
}

* A **static analysis tool** scans the code.
* It detects that temp is declared but never used.
* This doesn’t cause a failure now, but it may indicate **bad coding practice** or **unfinished logic**.

#### 2. Security Check – SQL Injection Risk

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Python code:



def get\_user(id):

query = "SELECT \* FROM users WHERE id = " + id

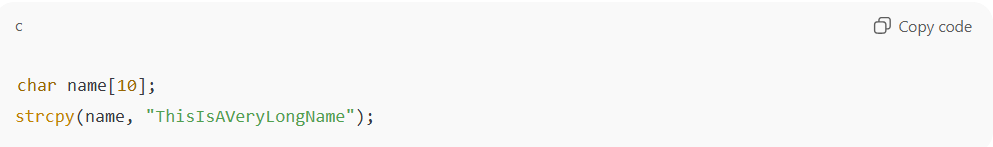
cursor.execute(query)

* A static analysis tool will flag this, because id is concatenated directly into the SQL query.
* If an attacker enters id = "1 OR 1=1", it can expose all records.
* The tool suggests using **parameterized queries** instead.

#### 3. Buffer Overflow Detection

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

C code:



char name[10];

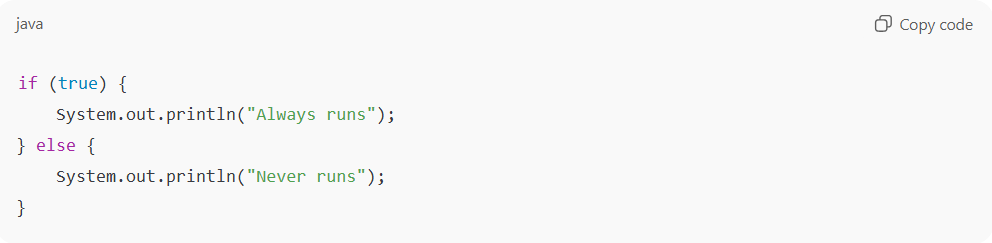
strcpy(name, "ThisIsAVeryLongName");

* Static analysis detects that "ThisIsAVeryLongName" is longer than the name array (10 characters).
* This would cause **buffer overflow** at runtime.
* The tool raises a **high-severity warning**, since this can crash the program or be exploited.

#### 4. Dead Code / Unreachable Code

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Java code:



if (true) {

System.out.println("Always runs");

} else {

System.out.println("Never runs");

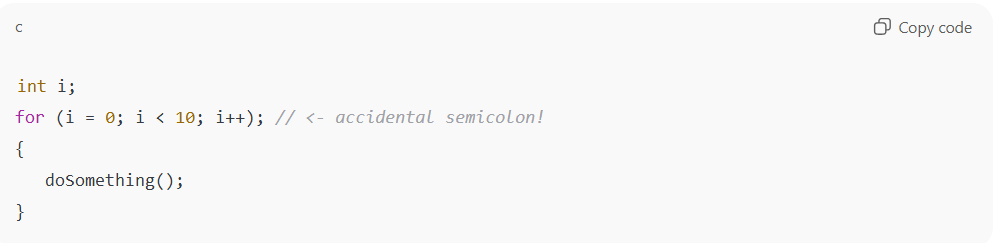
}

* The tool sees that the else branch is **never reachable**.
* It marks it as **dead code**, which bloats the program and may hide mistakes.

#### 5. Coding Standard Violation (Style/Rules)

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

In safety-critical domains (like aviation or automotive), strict rules apply (e.g., MISRA-C).  
Example:



int i;

for (i = 0; i < 10; i++); // <- accidental semicolon!

{

doSomething();

}

* A static analyzer spots the **empty loop** (because of ;).
* doSomething() is not inside the loop as the programmer intended.
* Without the tool, this bug might slip into production.

#### **Summary:** Static analysis catches:

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

* Style issues (unused variables, naming conventions).
* Logical errors (unreachable code, empty loops).
* Security risks (SQL injection, buffer overflow).
* Compliance issues (safety standards).

### Requirements Specification (Software Docs)

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Great — let’s go deeper into **Requirements Specification (Software Docs) examples** and how **static analysis** can detect problems:

#### 🔹 Example 1: Ambiguity

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Requirement:

“The system should provide a fast response to user queries.”

**Problem:**

* *Fast* is **ambiguous** (does it mean 1 sec? 5 sec?).

**Static Analysis Finding:**

* Flags words like *fast, user-friendly, efficient, adequate*.
* Suggests measurable requirements, e.g.:

“The system shall return 95% of queries within 2 seconds.”

#### 🔹 Example 2: Incompleteness

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Requirement:

“The system shall support mobile devices.”

**Problem:**

* Which devices? iOS, Android, tablets? Minimum OS version?

**Static Analysis Finding:**

* Detects missing specifics.
* Suggests rewriting:

“The system shall support Android version 12 and later, and iOS 15 and later.”

#### 🔹 Example 3: Inconsistency

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Requirement A:

“Password shall be at least 8 characters long.”

Requirement B:

“Password shall be at least 10 characters long.”

**Problem:**

* Contradiction between requirements.

**Static Analysis Finding:**

* Detects conflict.
* Suggests clarification before design starts.

#### 🔹 Example 4: Passive Voice / Unclear Responsibility

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Requirement:

“Data shall be encrypted before being sent.”

**Problem:**

* Who is responsible? The application, the database, or the middleware?

**Static Analysis Finding:**

* Flags passive voice.
* Suggests rewriting:

“The client application shall encrypt all data before sending it to the server.”

#### 🔹 Example 5: Unverifiable Requirement

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Requirement:

“The software shall be user-friendly.”

**Problem:**

* *User-friendly* cannot be tested objectively.

**Static Analysis Finding:**

* Flags as unverifiable.
* Suggests measurable usability criteria, e.g.:

“80% of first-time users shall complete the onboarding process within 3 minutes without external help.”

#### ✅ In short:

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Static analysis on requirements specification helps detect:

* Ambiguity
* Incompleteness
* Inconsistency
* Unclear responsibility
* Non-verifiable requirements

### Test Scripts

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

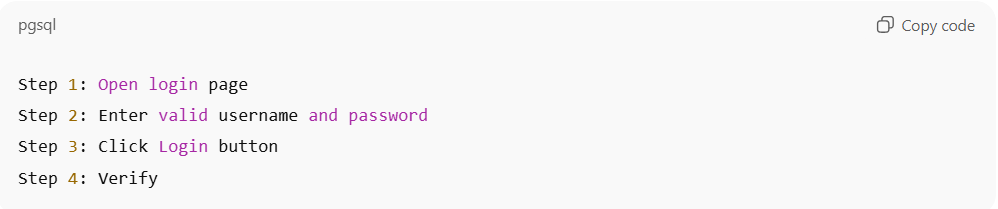
Perfect 👍 let’s go into **Test Scripts examples** and how **static analysis** can detect issues.

Test scripts are step-by-step instructions for testing software. Static analysis (without executing the tests) checks them for **clarity, completeness, and consistency**.

#### 🔹 Example 1: Missing Expected Result

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Test Script:



Step 1: Open login page

Step 2: Enter valid username and password

Step 3: Click Login button

Step 4: Verify

**Problem:**

* Step 4 says *Verify* but does not say **what** to verify.

**Static Analysis Finding:**

* Flags incomplete verification.
* Suggests:

“Verify that the user is redirected to the dashboard page.”

#### 🔹 Example 2: Ambiguity in Action

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**



Test Script:

Step 1: Enter user details

Step 2: Submit form

Step 3: Check system response

**Problem:**

* *Enter user details* is vague → which fields? name, email, phone?
* *Check system response* is unclear → check success message? check database?

**Static Analysis Finding:**

* Flags vague wording.
* Suggests rewriting as:



Step 1: Enter Name = "John", Email = "john@test.com"

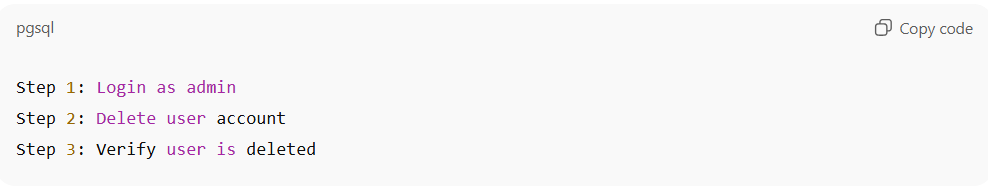
Step 2: Submit form

Step 3: Verify that the system displays "Registration successful"

#### 🔹 Example 3: Missing Preconditions

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Test Script:



Step 1: Login as admin

Step 2: Delete user account

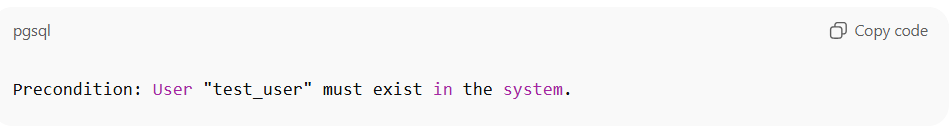
Step 3: Verify user is deleted

**Problem:**

* Preconditions not specified → Which user must exist to delete?

**Static Analysis Finding:**

* Suggests adding a precondition section:

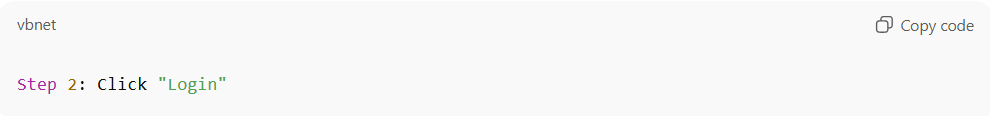


Precondition: User "test\_user" must exist in the system.

#### 🔹 Example 4: Inconsistent Terminology

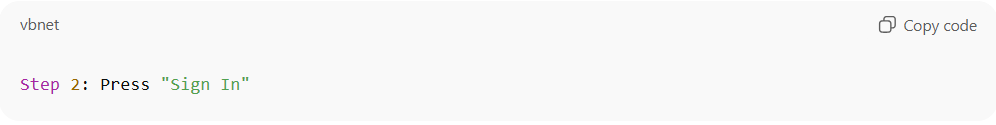
**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Test Script A:



Step 2: Click "Login"

Test Script B:



Step 2: Press "Sign In"

**Problem:**

* Both steps mean the same button, but with different names → leads to tester confusion.

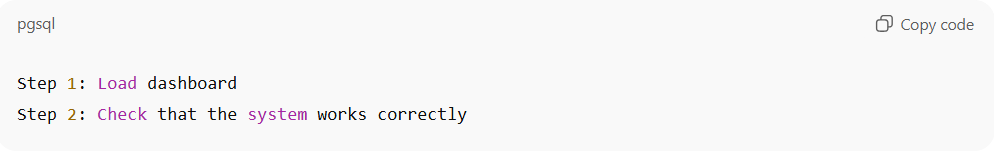
**Static Analysis Finding:**

* Flags inconsistent terminology.
* Suggests unifying terminology with the UI specification.

#### 🔹 Example 5: Non-Verifiable Step

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Test Script:



Step 1: Load dashboard

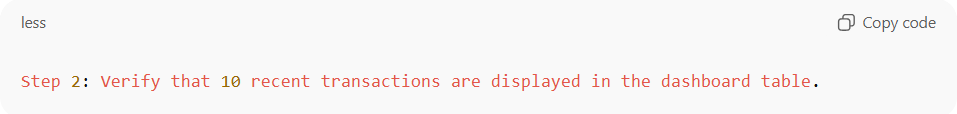
Step 2: Check that the system works correctly

**Problem:**

* *Works correctly* is not measurable.

**Static Analysis Finding:**

* Flags as unverifiable.
* Suggests defining objective checks, e.g.:



Step 2: Verify that 10 recent transactions are displayed in the dashboard table.

#### ✅ Summary

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**  
Static analysis of **test scripts** catches:

* Missing expected results
* Ambiguity in actions
* Missing preconditions
* Inconsistent terminology
* Non-verifiable steps

### API Documentation

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Perfect 👍 let’s go through **API Documentation examples** and how **static analysis** can detect problems in them.

API docs describe how developers should use an API (methods, inputs, outputs, constraints). If they are unclear or inconsistent, developers may misuse the API → leading to failures.

#### 🔹 Example 1: Missing Parameter Details

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

API Documentation:



POST /createUser

Parameters:

- username (string)

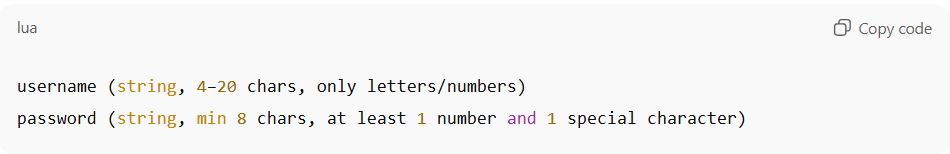
- password (string)

**Problem:**

* No info about length limits, allowed characters, or password rules.
* Developer doesn’t know if password must be ≥ 8 chars, contain symbols, etc.

**Static Analysis Finding:**

* Flags parameters with **missing constraints**.
* Suggests rewriting:



username (string, 4–20 chars, only letters/numbers)

password (string, min 8 chars, at least 1 number and 1 special character)

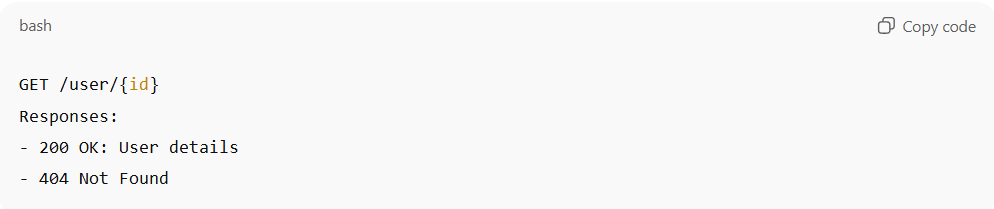
#### 🔹 Example 2: Inconsistent Response Codes

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

API Documentation:

GET /user/{id}

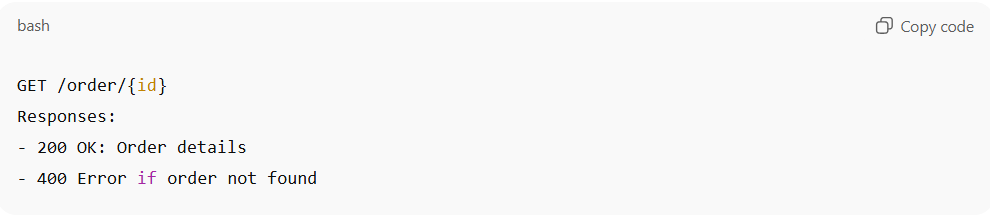
Responses:



- 200 OK: User details

- 404 Not Found

Another API endpoint says:



GET /order/{id}

Responses:

- 200 OK: Order details

- 400 Error if order not found

**Problem:**

* One endpoint uses **404** for not found, another uses **400**.
* Inconsistency confuses developers.

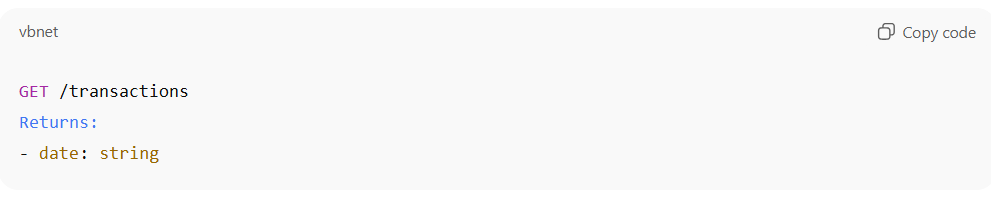
**Static Analysis Finding:**

* Detects inconsistency in error codes.
* Suggests aligning all “not found” cases to **404 Not Found**.

#### 🔹 Example 3: Ambiguous Data Format

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

API Documentation:



GET /transactions

Returns:

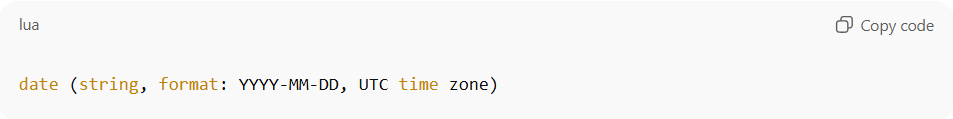
- date: string

**Problem:**

* “string” is ambiguous → is it YYYY-MM-DD? MM/DD/YYYY? with time zone?

**Static Analysis Finding:**

* Flags ambiguous data formats.
* Suggests rewriting:

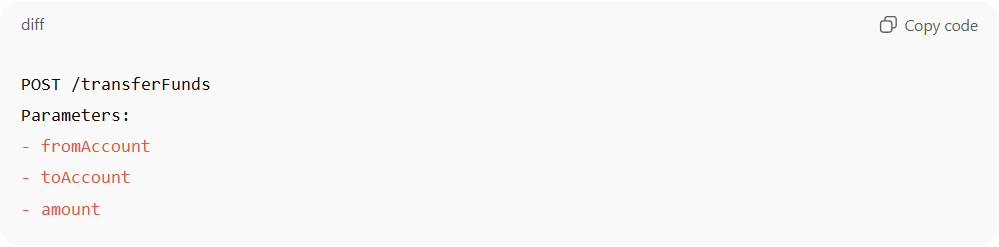


date (string, format: YYYY-MM-DD, UTC time zone)

#### 🔹 Example 4: Missing Authentication Info

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

API Documentation:



POST /transferFunds

Parameters:

- fromAccount

- toAccount

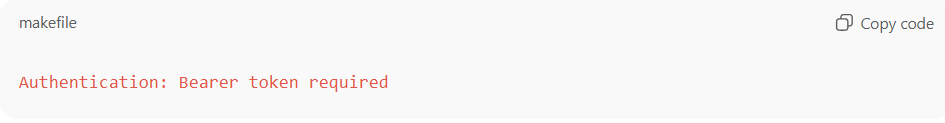
- amount

**Problem:**

* No mention of authentication or security (e.g., API keys, tokens).
* A developer might think it’s an open endpoint.

**Static Analysis Finding:**

* Flags **sensitive operations with no authentication rules** documented.
* Suggests rewriting:

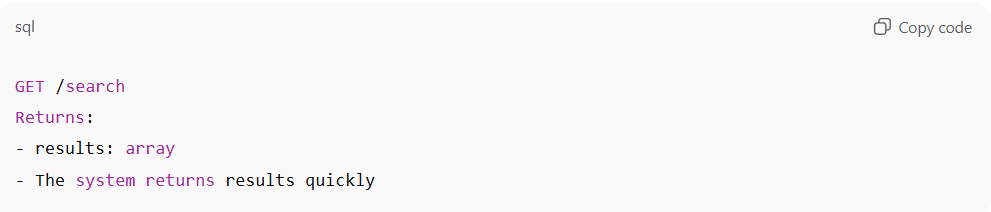


Authentication: Bearer token required

#### 🔹 Example 5: Unverifiable Requirement in Docs

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

API Documentation:



GET /search

Returns:

- results: array

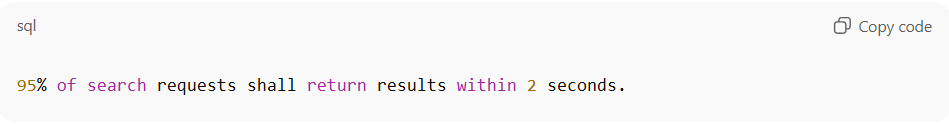
- The system returns results quickly

**Problem:**

* *quickly* is vague and non-testable.

**Static Analysis Finding:**

* Flags unverifiable terms.
* Suggests measurable wording:



95% of search requests shall return results within 2 seconds.

#### ✅ Summary:

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**  
Static analysis of API documentation catches:

* Missing parameter constraints
* Inconsistent response codes
* Ambiguous data formats
* Missing authentication/security details
* Unverifiable/vague requirements

### Coding Standards Documentation

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

Great 👍 let’s look at **Coding Standards Documentation examples** and how **static analysis** can be applied.

Coding standards are written rules a team or industry defines (for readability, safety, or security). Static analysis tools can automatically check if the **source code follows those documented rules**.

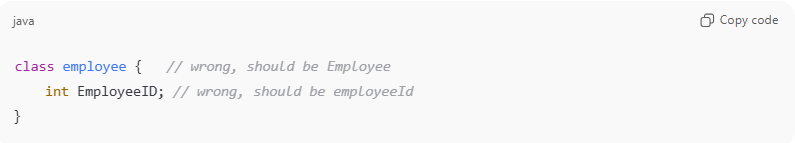
#### 🔹 Example 1: Naming Conventions

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Coding Standards Doc:**

* All class names must use **PascalCase**.
* All variables must use **camelCase**.

**Code:**



class employee { // wrong, should be Employee

int EmployeeID; // wrong, should be employeeId

}

**Static Analysis Finding:**

* Flags naming violations.
* Suggests corrections based on documented rules.

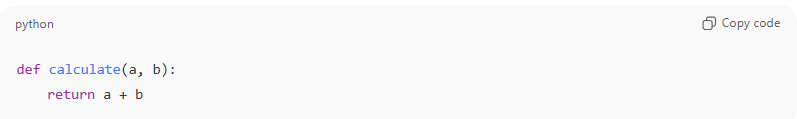
#### 🔹 Example 2: Commenting Rules

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Coding Standards Doc:**

* Every function must have a docstring explaining inputs, outputs, and purpose.

**Code:**



def calculate(a, b):

return a + b

**Static Analysis Finding:**

* Flags missing documentation.
* Suggests adding:



def calculate(a, b):

"""

Adds two integers.

:param a: first integer

:param b: second integer

:return: sum of a and b

"""

return a + b

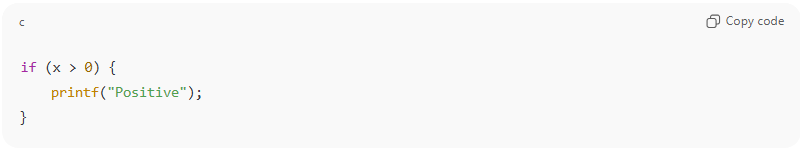
#### 🔹 Example 3: Braces and Indentation Style

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Coding Standards Doc:**

* In C, opening brace { must be on a new line.

**Code:**



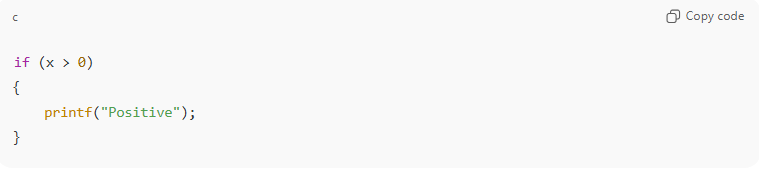
if (x > 0) {

printf("Positive");

}

**Static Analysis Finding:**

* Flags brace style violation.
* Suggests:



if (x > 0)

{

printf("Positive");

}

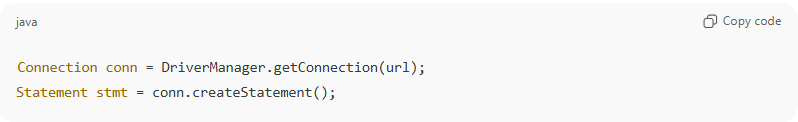
#### 🔹 Example 4: Error Handling Rules

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Coding Standards Doc:**

* All database calls must handle exceptions.

**Code:**



Connection conn = DriverManager.getConnection(url);

Statement stmt = conn.createStatement();

**Static Analysis Finding:**

* Flags missing try-catch.
* Suggests adding exception handling.

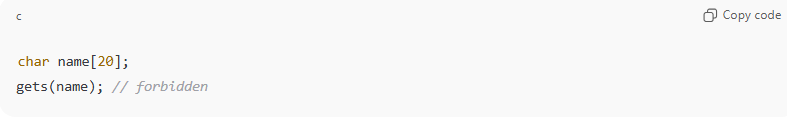
#### 🔹 Example 5: Security Rules (MISRA-C or CERT Standards)

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**Coding Standards Doc:**

* Never use gets() in C (unsafe, causes buffer overflows).

**Code:**



char name[20];

gets(name); // forbidden

**Static Analysis Finding:**

* Flags rule violation.
* Suggests using fgets() instead.

#### ✅ Summary:

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**  
Static analysis against coding standards detects:

* Naming convention violations
* Missing documentation/comments
* Wrong indentation or brace style
* Missing error handling
* Unsafe function usage

## Summary

**# Source**: **Chatgpt (GPT-5-Standard) at [9/3/2025]**

**🔹 Static Analysis for Documentation (recap)**

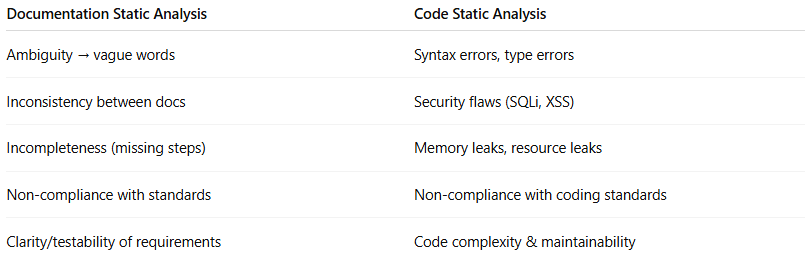
* **Ambiguity** → Vague words (*fast, efficient, user-friendly*).
* **Inconsistency** → Requirement ≠ Design doc.
* **Incompleteness** → Missing steps/results in test cases.
* **Non-compliance** → Docs don’t match coding standards.

**🔹 Static Analysis for Code**

Here’s what it typically detects (without running the program):

1. **Syntax Errors & Violations**
   * Example: Missing semicolon (;), undeclared variable.
   * Tool: ESLint, Pylint.
2. **Code Style & Standards Compliance**
   * Example: Variable naming doesn’t follow convention, indentation errors.
   * Tool: Pylint, StyleCop, ESLint.
3. **Unused / Dead Code**
   * Example: A function is defined but never called.
   * Tool: SonarQube, CodeClimate.
4. **Type & Data Issues**
   * Example: Assigning a string to an integer variable in statically typed languages.
   * Tool: MyPy (Python), TypeScript compiler.
5. **Security Vulnerabilities**
   * Example: SQL injection risk if query built with string concatenation.
   * Tool: Fortify, Bandit.
6. **Concurrency Issues**
   * Example: Race conditions, deadlocks in multi-threaded code.
   * Tool: Coverity, SPIN.
7. **Memory & Resource Leaks**
   * Example: Forgetting to free allocated memory (malloc in C).
   * Tool: Cppcheck, Polyspace.
8. **Code Complexity & Maintainability**
   * Example: Function too long (hundreds of lines), hard to maintain.
   * Tool: SonarQube, CodeClimate.
9. **Standards & Compliance Checking**
   * Example: MISRA-C compliance in automotive software.
   * Tool: Polyspace, LDRA.

✅ **Quick Contrast**



| **Documentation Static Analysis** | **Code Static Analysis** |
| --- | --- |
| Ambiguity → vague words | Syntax errors, type errors |
| Inconsistency between docs | Security flaws (SQLi, XSS) |
| Incompleteness (missing steps) | Memory leaks, resource leaks |
| Non-compliance with standards | Non-compliance with coding standards |
| Clarity/testability of requirements | Code complexity & maintainability |