# Static Testing applies to Verification and Validation

## How Static Testing applies to verification and validation

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Static Testing plays a role in **both Verification and Validation**, but mostly it’s associated with **Verification**.Here’s the breakdown:

### 1. Static Testing in Verification

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Verification** = *"Are we building the product right?"*

* **Static Testing** checks artifacts *without running the software* to ensure they meet specifications, standards, and consistency.
* Activities:  
  + **Manual Reviews**: Requirements review, design review, walkthroughs, inspections.
  + **Automated Static Analysis**: Code analysis, compliance checks, security scans.

**Example:**

* Reviewing the **requirements document** to ensure it matches the business rules.
* Using **SonarQube** to find coding standard violations before compilation.

🔍 **Why here?** Because verification is about checking correctness *before execution* — exactly what static testing does.

#### FULL EXAMPLES FOR Static Testing in Verification

##### 1. Requirements Stage (Early Verification)

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

📄 **Artifact:** *Software Requirements Specification (SRS)* 🔍 **Static Testing Activity:** *Manual review / inspection* **Example:**

* Check that all functional requirements are written in a clear, unambiguous way.
* Identify missing acceptance criteria for a “Forgot Password” feature.
* Ensure “login” is defined consistently across the document (not “sign in” in some places and “login” in others).

💡 **Tool support:** Grammar checkers, requirement management tools with traceability reports.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Alright — let’s turn that into a **real-world example** with an actual snippet of an SRS document and show how **Static Testing** (manual review) would detect issues.

**Sample SRS excerpt (Login Feature)**

**Requirement RQ-1: User Login**

The system shall allow the user to log in using their registered credentials.  
 The login page will contain fields for username and password, and a “Sign in” button.  
 Password must be at least 8 characters.  
 If login fails, the user will be redirected to the main page.

**Static Testing – Manual Review Findings**

1. **Ambiguity in wording**
   * *Issue:* “registered credentials” is vague — does it mean email and password, username and password, or both are supported?
   * *Recommendation:* Specify exactly which credentials are accepted:  
       
       
      “The system shall allow the user to log in using a registered email address and password.”
2. **Missing acceptance criteria**
   * *Issue:* No requirement for “Forgot Password” behavior, even though it’s part of the login page in the UI mockups.
   * *Recommendation:* Add:  
       
       
      “The login page shall provide a ‘Forgot Password’ link, allowing the user to reset their password via email verification.”
3. **Inconsistent terminology**
   * *Issue:* The requirement says “log in” in one place and “Sign in” button in another — inconsistent wording might confuse both developers and translators.
   * *Recommendation:* Use “Log in” consistently throughout the document.
4. **Logical error in failure handling**
   * *Issue:* “If login fails, the user will be redirected to the main page” — redirecting to main page after failure may lock the user out. This likely contradicts UX expectations.
   * *Recommendation:* “If login fails, the user shall remain on the login page and be shown an error message.”

✅ This is **Static Testing in Verification** because we haven’t executed any code — we just reviewed the requirement document **against clarity, completeness, and consistency rules**.

##### 2. Design Stage

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

📄 **Artifact:** *System design document / UML diagrams* 🔍 **Static Testing Activity:** *Technical review* **Example:**

* Review a sequence diagram to ensure it follows the intended business flow.
* Check that every function in the design has a corresponding requirement.
* Detect mismatches between database ER diagram and the data model described in the requirements.

💡 **Tool support:** UML validation tools, modeling tools like Enterprise Architect.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

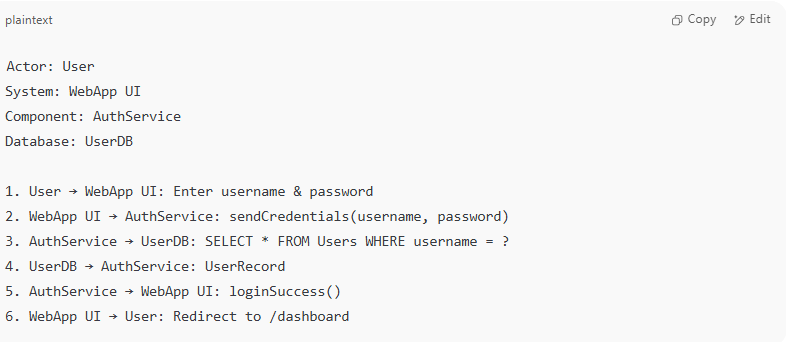
Alright — let’s walk through a **realistic static testing example** using a **System Design Document** with **UML diagrams**.

**Sample Sequence Diagram (Login Flow)**

**Intended business flow**:

1. User enters credentials in the UI.
2. UI sends credentials to Authentication Service.
3. Authentication Service queries the database.
4. Database returns result.
5. Authentication Service sends success/failure to UI.
6. UI redirects user accordingly.

**Excerpt from System Design Document – Sequence Diagram:**

****

Actor: User

System: WebApp UI

Component: AuthService

Database: UserDB

1. User → WebApp UI: Enter username & password

2. WebApp UI → AuthService: sendCredentials(username, password)

3. AuthService → UserDB: SELECT \* FROM Users WHERE username = ?

4. UserDB → AuthService: UserRecord

5. AuthService → WebApp UI: loginSuccess()

6. WebApp UI → User: Redirect to /dashboard

**Static Testing – Technical Review Findings**

1. **Mismatch with Requirements**
   * *Requirement says*: The system must lock an account after 5 failed login attempts.
   * *Design issue:* Sequence diagram does not include any interaction for tracking failed attempts or locking accounts.
   * *Recommendation:* Add a failedAttemptCounter and conditional branch to lock the account.
2. **Missing Validation Step**
   * *Requirement says*: The system must hash passwords before comparing with the database.
   * *Design issue:* The diagram shows sending plain text passwords from UI → AuthService and querying directly.
   * *Recommendation:* Insert a hashPassword() step in AuthService before database query.
3. **Database Design Mismatch**
   * *ER diagram says*: Table Users has fields email, hashed\_password.
   * *Design issue:* Sequence diagram query uses username, which does not exist in the ER diagram.
   * *Recommendation:* Change query to WHERE email = ?.
4. **Business Flow Deviation**
   * *Intended behavior:* On login failure, user should stay on the login page with an error message.
   * *Design issue:* Diagram does not show a failure path — only a success path exists.
   * *Recommendation:* Add an alternative path:  
       
       
      5b. AuthService → WebApp UI: loginFailure(errorMessage)  
      6b. WebApp UI → User: Display error, stay on /login.

💡 **Tool Support for Automated Checks:**

* **Enterprise Architect** – can check UML consistency with requirements.
* **Visual Paradigm** – can validate diagram syntax.
* **Modelio** – supports traceability links between UML elements and requirements.

##### 3. Coding Stage (Pre-Execution)

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

📄 **Artifact:** *Source code* 🔍 **Static Testing Activity:** *Automated static analysis + manual code review* **Example:**

* Use **SonarQube** to detect:  
  + Unused variables
  + Security vulnerabilities (e.g., SQL injection risk)
  + Code complexity exceeding threshold
* Manual review to confirm naming conventions and code readability.

💡 **Tool support:** SonarQube, ESLint, Pylint, StyleCop, Checkstyle.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Alright — here’s a **real-world static testing example** for **source code** using both automated analysis and manual review.

**Sample Source Code – Login Function (Java)**

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**



public class AuthService {

private Database db;

public AuthService(Database database) {

this.db = database;

}

public boolean login(String username, String password) {

String query = "SELECT \* FROM Users WHERE username = '" + username +

"' AND password = '" + password + "'";

User user = db.executeQuery(query);

if (user != null) {

return true;

}

return false;

}

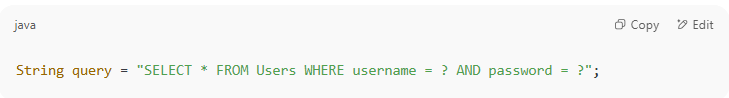
}

**Static Testing Findings**

**Automated Static Analysis (SonarQube / Checkstyle)**

1. **SQL Injection Vulnerability**
   * **Tool output:** *"Concatenating user input into SQL queries can lead to SQL injection"*

**Fix suggestion:** Use prepared statements:

  
  
 String query = "SELECT \* FROM Users WHERE username = ? AND password = ?";

1. **Hardcoded String Concatenation in Queries**
   * **Tool output:** *"Avoid using string concatenation for dynamic queries"*
   * **Fix suggestion:** Use parameterized queries.
2. **No Password Hashing**
   * **Tool output:** *"Passwords must be hashed before storage or comparison"*
   * **Fix suggestion:** Apply hashing (e.g., BCrypt).
3. **Method Complexity**
   * **Tool output:** *"Method 'login' is too complex"* (Cyclomatic complexity warning — even though here it’s simple, in real cases large if-else blocks trigger this).

**Manual Code Review Findings**

1. **Naming Conventions**
   * Variable db is fine but might be clearer as databaseConnection.
   * Method name login is OK, but documentation missing.
2. **Readability**
   * No comments explaining how authentication works.
   * Password handling logic is unclear — needs in-code documentation.
3. **Security Practice**
   * Password comparison done in plain text → violates security best practices.
   * No account lockout after multiple failed attempts.
4. **Error Handling**
   * Method just returns false without logging the reason for failure.

💡 **Tool Support Used in Example**

* **SonarQube** – detected SQL injection, complexity, and missing hashing.
* **Checkstyle** – flagged missing JavaDoc and naming issues.
* **PMD** – could detect unused variables if present.
* **Manual Review** – spotted missing comments and lockout feature.

##### 4. Test Documentation Stage

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

📄 **Artifact:** *Test plan, test cases, test scripts* 🔍 **Static Testing Activity:** *Peer review* **Example:**

* Check that all test cases have clear preconditions and expected results.
* Verify traceability matrix to ensure each requirement is covered by at least one test case.
* Spot that a test case for “checkout process” doesn’t include testing of discount codes.

💡 **Tool support:** Test management tools (Jira Xray, TestRail).

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Alright — here’s a **real-world static testing example** for a **test plan, test cases, and test scripts** with a peer review focus.

**📄 Artifact: Test Plan & Test Cases – *E-commerce Checkout Feature***

**Test Plan Extract**

**Objective:** Verify the checkout process from cart to order confirmation.  
 **Scope:**

* Adding products to cart
* Applying discount codes
* Payment processing (credit card, PayPal)
* Order confirmation email

**Sample Test Case (Before Review)**

**Test Case ID:** TC-CHK-005  
 **Title:** Verify checkout process  
 **Preconditions:** User is logged in.  
 **Steps:**

1. Add product to cart.
2. Click checkout.
3. Enter payment details.
4. Place order.

**Expected Result:** Order is successfully placed.

**Static Testing – Peer Review Findings**

**1. Missing Preconditions Details**

* **Issue:** “User is logged in” is too vague — doesn’t specify *which* user type (guest, registered, admin).
* **Correction:** “User is logged in as a registered customer with at least one product in the cart and a valid shipping address on file.”

**2. Missing Expected Results**

* **Issue:** “Order is successfully placed” is too broad — missing specific validation points.
* **Correction:**
  + Order confirmation page is displayed.
  + Order ID is generated.
  + Confirmation email is sent to the registered email address.
  + Payment transaction is recorded in the payment gateway logs.

**3. Missing Requirement Coverage**

* **Traceability Matrix Check:** Requirement *REQ-CHK-003* — *"System must apply valid discount codes during checkout"* — has no corresponding test case.
* **Correction:** Add a new test case **TC-CHK-010** for “Checkout with valid discount code.”

**4. Missing Negative Scenarios**

* No test case covers:  
  + Expired discount code
  + Payment failure handling
  + Invalid shipping address
* **Correction:** Add test cases **TC-CHK-011**, **TC-CHK-012**, and **TC-CHK-013**.

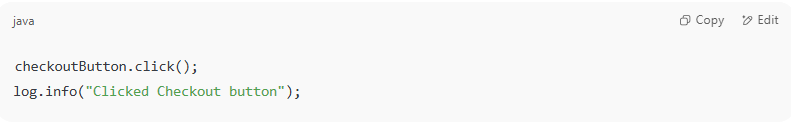
**5. Script Readability (For Automated Test Script)**

**Original snippet in Selenium:**

****

driver.findElement(By.id("checkoutBtn")).click();

**Reviewer Comment:** Add meaningful variable names & logging.  
 **Improved:**

****

checkoutButton.click();

log.info("Clicked Checkout button");

💡 **Tool Support Used in Example:**

* **TestRail** – to review coverage and precondition/expected result clarity.
* **Jira Xray** – to cross-check requirements coverage with the traceability matrix.

##### 5. Compliance & Standard Checks

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

📄 **Artifact:** *Entire project documentation and code* 🔍 **Static Testing Activity:** *Automated compliance scanning* **Example:**

* Run static security analysis for compliance with **OWASP Top 10**.
* Check for licensing issues in open-source libraries.
* Validate code style against industry guidelines (e.g., MISRA for automotive software).

💡 **Tool support:** WhiteSource, Black Duck, Fortify SCA.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Here’s a **real-world static testing example** for **entire project documentation + code** with **automated compliance scanning** applied.

**📄 Artifact: Entire Project Documentation & Source Code – *Online Banking Web Application***

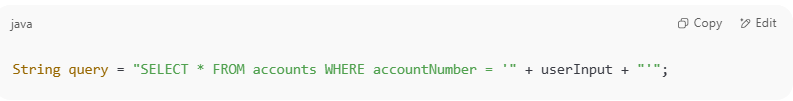
**Static Testing Activity: Automated Compliance Scanning**

**1. OWASP Top 10 Security Check**

**Tool Used:** Fortify SCA + SonarQube Security Rules

**Finding:**

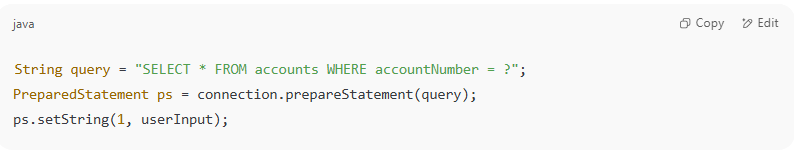
* **Vulnerability:** *SQL Injection Risk*
* **Location:** AccountController.java, Line 125
* **Code Detected:**

****

String query = "SELECT \* FROM accounts WHERE accountNumber = '" + userInput + "'";

* **Risk:** User-provided userInput is concatenated directly into SQL query without parameterization.

**Correction:**

****

String query = "SELECT \* FROM accounts WHERE accountNumber = ?";

PreparedStatement ps = connection.prepareStatement(query);

ps.setString(1, userInput);

**2. Open-Source License Compliance**

**Tool Used:** Black Duck

**Finding:**

* Library: commons-email-1.4.jar
* License: Apache 2.0 ✅ (allowed)
* Library: some-old-crypto-lib.jar
* License: GPL v3 ❌ (incompatible with commercial license of the project)

**Action Taken:**

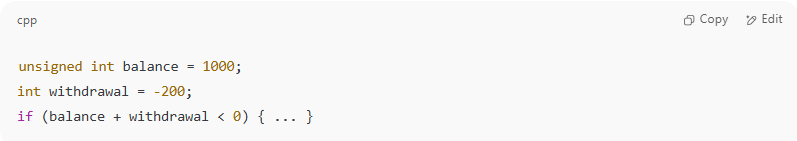
* Replaced with MIT-licensed alternative Bouncy Castle library for encryption.

**3. Code Style & Industry Guideline Check**

**Tool Used:** MISRA C++ Compliance Checker (for embedded banking kiosk code)

**Finding:**

* **Violation:** MISRA C++ Rule 7–1–1 – *Implicit conversions between signed and unsigned integers*
* **Original Code:**

****

unsigned int balance = 1000;

int withdrawal = -200;

if (balance + withdrawal < 0) { ... }

* **Correction:** Explicit type handling and validation before arithmetic.

**4. Documentation Compliance**

**Tool Used:** Documentation linter + internal policy checklist

**Finding:**

* No clear description of encryption algorithms in *Security Design Document*.
* Missing API versioning details in *API Specification PDF*.

**Action Taken:**

* Updated docs to include AES-256-GCM encryption scheme details.
* Added REST API versioning section with example URIs.

💡 **Tool Support in Example:**

* **Fortify SCA** – detects OWASP Top 10 risks (insecure code patterns)
* **Black Duck** – checks license compatibility for open-source dependencies
* **MISRA Compliance Tool** – enforces industry-specific coding standards
* **DocLint** – ensures documentation follows compliance templates

##### Why this is Verification

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

In all these examples:

* We **haven’t executed the program**.
* We are **checking against requirements, standards, and specifications**.
* We are ensuring the product is being **built right** before runtime testing.

### 2. Static Testing in Validation

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Validation** = *"Are we building the right product?"*

* Static Testing supports validation indirectly by checking if artifacts reflect what the customer actually needs — before implementation.
* Activities:  
  + **Requirements review with stakeholders** (ensures the document matches customer expectations).
  + **Prototyping or model inspection** (ensures design matches user needs).

**Example:**

* Reviewing a **UI wireframe** to confirm it matches the customer’s workflow before coding.
* Checking **acceptance criteria** in user stories for clarity and completeness.

#### FULL EXAMPLES FOR Static Testing in Validation

Alright — let’s walk through **Static Testing in Validation** with full, realistic examples.  
 Remember:

* **Verification** checks **if we built the product right** (reviews, inspections, analysis).
* **Validation** checks **if we built the right product** (does it meet business/user needs).
* **Static Testing** can help in *validation* by reviewing artifacts **before execution** to see if they align with actual customer requirements.

##### 📄 Example 1 – Validating Requirements

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Artifact:** *Software Requirements Specification (SRS)* for an **E-commerce Website**

**Static Testing Activity:** Business stakeholder review session

**Finding:**

* Requirement states:  
    
    
   "The system shall allow users to purchase products online using credit card."
* Issue: Business stakeholders point out that **PayPal** and **Apple Pay** are also mandatory payment methods for the target market.

**Outcome:**

* Requirement updated to:  
    
    
   "The system shall allow users to purchase products online using credit card, PayPal, and Apple Pay."

**Validation Focus:** Ensuring the documented requirement meets actual customer expectations before any code is written.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Here’s a **real-world style example** of that scenario so it feels less abstract:

**📄 Artifact:** *Software Requirements Specification (SRS)* for **Amazon Marketplace – New Region Launch**

**🔍 Static Testing Activity:** Business stakeholder review session (Product Owner, Business Analyst, Marketing Lead, QA Lead)

**📌 Finding:**

* Original requirement in the SRS:  
    
    
   *"The system shall allow users to purchase products online using credit card."*
* During review, the **Marketing Lead** noted:  
    
    
   “In our target region (Germany), over 40% of customers use PayPal as their primary payment method, and Apple Pay adoption is growing quickly. Credit card only will cause significant loss in conversions.”

**🚩 Issue:** Missing critical payment methods for the local market.

**✅ Outcome:**

* Requirement updated to:  
    
    
   *"The system shall allow users to purchase products online using credit card, PayPal, and Apple Pay."*
* Additional acceptance criteria were added:  
  + PayPal checkout flow must redirect securely to PayPal sandbox/live environment.
  + Apple Pay option must appear only on Apple devices and comply with Apple’s Human Interface Guidelines.

**🎯 Validation Focus:** Ensuring the requirement matches **real customer payment preferences** *before* development starts — preventing costly rework after launch.

##### 📄 Example 2 – Validating UX Wireframes

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Artifact:** *UI Mockups & Wireframes* for a **Banking Mobile App**

**Static Testing Activity:** Usability review with customers & testers

**Finding:**

* Login screen shows **"Username"** field.
* Stakeholders clarify that **account number OR email** should be usable to log in.
* Customers find the “Forgot Password” link too small and not accessible enough.

**Outcome:**

* Wireframe updated to include:  
  + "Account Number / Email" placeholder.
  + Larger, clearly visible "Forgot Password?" link.

**Validation Focus:** Ensuring the UI design matches real-world usage and accessibility needs.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Here’s a **real-world style example** for that banking mobile app scenario:

**📄 Artifact:** *UI Mockups & Wireframes* for **HSBC Mobile Banking App – New Login Redesign**

**🔍 Static Testing Activity:** Usability review workshop with 6 customer representatives, 2 UX designers, 1 accessibility consultant, and QA team.

**📌 Finding:**

* Original login wireframe:  
  + Single input labeled **"Username"**.
  + Small “Forgot Password?” link in light gray at the bottom.
* **Stakeholder clarification (Product Manager):** “Many of our customers log in with their **account number** or **email**, not just a username. Restricting to username will break their usual workflow.”
* **Customer feedback:** “The *Forgot Password* link is too small and hard to tap, especially for older users or on smaller screens.”

**🚩 Issues Identified:**

1. Field label too restrictive → Doesn’t match real login behavior.
2. Accessibility issue with link size and contrast.

**✅ Outcome:**

* Placeholder text changed to **"Account Number / Email"**.
* Input field validation updated in requirements to accept both formats.
* *Forgot Password?* link enlarged, placed directly under login button, and given a high-contrast color meeting WCAG 2.1 AA standards.

**🎯 Validation Focus:** Confirmed that the **UI design aligns with actual user behavior** and **meets accessibility guidelines** *before coding starts*, reducing redesign costs later.

##### 📄 Example 3 – Validating Test Cases Against User Scenarios

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Artifact:** *Acceptance Test Cases* for a **Ride-Sharing App**

**Static Testing Activity:** Peer review with Product Owner

**Finding:**

* Test case:  
    
    
   "Verify user can book a ride from pickup to drop-off location."
* Stakeholder points out missing validation for:  
  + Choosing ride type (Economy, Premium, XL)
  + Adding multiple stops during the ride

**Outcome:**

* New test cases added to cover ride type selection and multi-stop feature.

**Validation Focus:** Ensuring test coverage aligns with actual business workflows.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Here’s a **real-world style example** for that ride-sharing app acceptance test case review:

**📄 Artifact:** *Acceptance Test Cases* for **Uber Ride-Booking Module – Release 5.2**

**🔍 Static Testing Activity:** Peer review session with Product Owner, QA Lead, and Senior Tester before sprint sign-off.

**📌 Finding:**

* **Original test case:** *TC-RIDE-001*: “Verify user can book a ride from pickup to drop-off location.”
* **Product Owner feedback:** “This is too basic. In real usage, customers choose a ride type and often add stops along the way. These flows must be validated before release.”

**🚩 Issues Identified:**

1. No test coverage for **ride type selection** (Economy, Premium, XL).
2. No validation for **multi-stop rides** (common in airport pickups, errands, etc.).

**✅ Outcome:**

* Added new test cases:  
  + *TC-RIDE-002*: Verify user can select ride type (Economy, Premium, XL) before confirming booking.
  + *TC-RIDE-003*: Verify user can add up to 2 extra stops before finalizing the ride.
* Updated acceptance criteria in **User Story RS-102** to explicitly require both features in test coverage.
* Linked new test cases to **business requirement BR-RIDE-07** in the requirements traceability matrix.

**🎯 Validation Focus:** Ensured acceptance tests now reflect **actual customer workflows** and **critical business scenarios**, reducing the risk of missed features in UAT.

##### 📄 Example 4 – Validating Business Process Models

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Artifact:** *Business Process Flow Diagram* for **Hospital Appointment Booking System**

**Static Testing Activity:** Walkthrough with medical staff

**Finding:**

* Diagram shows patient directly selecting a doctor.
* Hospital staff explain that selection must go **through department selection first** for insurance validation.

**Outcome:**

* Process diagram updated to:  
    
    
   Select Department → Verify Insurance → Choose Doctor → Confirm Appointment

**Validation Focus:** Ensuring business process reflects reality before system design is locked.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Here’s a **real-world style example** for that hospital appointment booking process:

**📄 Artifact:** *Business Process Flow Diagram* – Hospital Appointment Booking System (Project: MedBook v3.4)

**🔍 Static Testing Activity:** Walkthrough review session with **hospital front-desk staff, insurance coordinators, and system analyst** before finalizing the process model.

**📌 Finding:**

* **Original process diagram flow:** Patient → Select Doctor → Confirm Appointment
* **Issue raised by hospital staff:** “Patients must first select a department (e.g., Cardiology, Orthopedics) so that we can check insurance coverage before allowing them to choose a doctor. Otherwise, patients might book with doctors not covered by their insurance.”

**🚩 Problems Identified:**

1. Missing **insurance validation** step.
2. No enforcement of **department-based filtering** of available doctors.

**✅ Outcome:**

* Updated BPMN diagram to:  
   **Patient → Select Department → Verify Insurance → Choose Doctor → Confirm Appointment**
* Added process rule: “Doctor selection list is filtered based on department and insurance network.”
* Updated business requirement **BR-HOSP-12** to include insurance verification as a **mandatory step**.
* Shared revised diagram with both **IT and hospital management** for sign-off.

**🎯 Validation Focus:** Ensured the **business workflow matches real-world hospital procedures** and complies with insurance policy constraints, preventing costly redesigns later in the project.

##### 📄 Example 5 – Validating Compliance Documentation

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

**Artifact:** *Data Privacy Compliance Checklist* for a **Healthcare Management System**

**Static Testing Activity:** Legal & compliance team review

**Finding:**

* Requirement says:  
    
    
   "Store patient data securely."
* Legal team highlights missing HIPAA-mandated **audit logging** and **patient consent tracking**.

**Outcome:**

* Updated requirement to specify:  
  + Encryption method (AES-256 at rest, TLS 1.3 in transit)
  + Detailed logging and consent form storage procedures.

**Validation Focus:** Ensuring non-functional requirements meet regulatory standards before development.

###### Real-world example

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**

Here’s a **real-world style example** for that healthcare data privacy compliance case:

**📄 Artifact:** *Data Privacy Compliance Checklist* – Healthcare Management System (Project: MediSecure v5.2)

**🔍 Static Testing Activity:** Legal & Compliance Team Review (HIPAA & GDPR requirements alignment)

**📌 Finding:**

* **Original requirement:** “Store patient data securely.”
* **Issue raised by legal team:** This is too vague for HIPAA compliance — it does not specify encryption standards, audit logging for all PHI (Protected Health Information) access, or patient consent tracking.  
   GDPR also requires explicit consent storage and the ability to withdraw consent.

**🚩 Problems Identified:**

1. No mention of **encryption algorithms** or key management policies.
2. Missing **audit trail requirements** for data access/modification.
3. No process defined for **patient consent capture and withdrawal tracking**.

**✅ Outcome:**

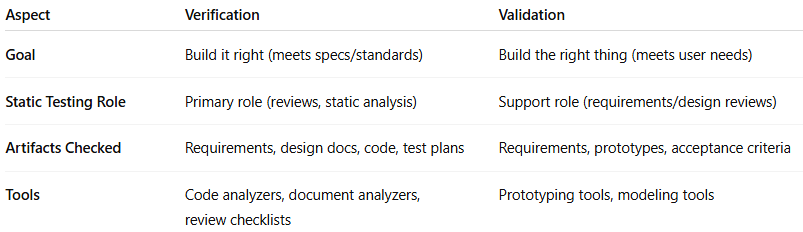
* Requirement updated to:  
    
    
   “All patient data shall be encrypted using AES-256 at rest and transmitted using TLS 1.3.  
   The system shall log all access, updates, and deletions of PHI with user ID, timestamp, and action taken.  
   Consent forms (digital or scanned) shall be securely stored and linked to the patient profile, with audit logs for consent changes.”
* Compliance checklist updated with specific HIPAA §164.312 and GDPR Article 7 references.
* Added **security acceptance criteria** to SRS.
* Development team briefed on the **NIST SP 800-53** security control mapping.

**🎯 Validation Focus:** Ensured **non-functional security requirements** are explicitly aligned with **regulatory and legal standards** before any coding, reducing risk of compliance violations and fines.

✅ **Key Point:** In **Static Validation**, you don’t run the code — you **review artifacts** to check that what’s *planned* to be built will truly satisfy the intended use and business needs.

## Summary Table

**# Source**: **Chatgpt (GPT-5-Standard) at [8/14/2025]**



| **Aspect** | **Verification** | **Validation** |
| --- | --- | --- |
| **Goal** | Build it right (meets specs/standards) | Build the right thing (meets user needs) |
| **Static Testing Role** | Primary role (reviews, static analysis) | Support role (requirements/design reviews) |
| **Artifacts Checked** | Requirements, design docs, code, test plans | Requirements, prototypes, acceptance criteria |
| **Tools** | Code analyzers, document analyzers, review checklists | Prototyping tools, modeling tools |