

# CASE STUDY/MINI PROJECT-SOFTWARE QUALITY ASSURANCE

## INTRODUCTION

Software Quality Assurance (QA) is all about building quality into software from the start, not just testing it at the end. It's a systematic process focused on preventing defects and ensuring the software is reliable, secure, and user-friendly. Ultimately, good QA saves money, makes customers happy, and boosts a company's reputation.

### Prompting can support QA teams by:

- Generating test cases from requirements
  - Explaining bugs in natural language
  - Summarizing crash logs or error messages
  - Classifying bug severity
  - Translating informal bug reports into structured templates
- 

## Functional Test Cases - Basic

### Prompt 1:

"Generate a list of basic functional test cases for an e-commerce product detail page. Focus on core functionalities like displaying product information, adding to cart, and image viewing."

#### Functional Test Cases for E-commerce Product Detail Page

##### = Product Information Display

- Test Case: Verify that the product name is correctly displayed.
- Test Case: Verify that the product price (including currency) is accurately shown.
- Test Case: Confirm the product description is fully visible and readable.
- Test Case: Check if the product SKU/model number (if applicable) is displayed correctly.
- Test Case: Ensure the product availability/stock status (e.g., "In Stock," "Out of Stock") is accurate.
- Test Case: Verify that product ratings or reviews summary (if present) are shown.

##### = Add to Cart Functionality

- Test Case: Add a single item to the cart: Verify clicking the "Add to Cart" button successfully adds the item to the shopping cart, and a confirmation message appears.

- Test Case: Add multiple quantities to the cart: Verify that specifying a quantity (e.g., 5) and clicking "Add to Cart" adds the correct number of items.
- Test Case: Verify that adding an item updates the cart count icon (if present).
- Test Case: Check if selecting different product variations (e.g., size, color) correctly updates the product details and allows adding to cart.

#### = Image Viewing

- Test Case: Verify the main product image loads correctly and is clearly displayed.
- Test Case: Confirm that thumbnail images (if multiple images exist) are present and clickable.
- Test Case: Test clicking a thumbnail to ensure it updates the main product image.
- Test Case: Validate image zoom functionality (if available) allows for a closer view of the product details.
- Test Case: Check that image navigation controls (e.g., arrows) correctly cycle through images.

---

## Negative Test Cases for "Add to Cart" Button

### Prompt 2:

#### = Adding with Zero Quantity:

- Test Steps: Navigate to a product detail page. In the quantity field, manually enter or select '0'. Click the "Add to Cart" button.
- Expected Result: An error message should clearly state that the quantity must be at least 1 or a valid positive number. The item should not be added to the cart. = Attempting to Add an Out-of-Stock Item:
- Test Steps: Find a product on the e-commerce site that is explicitly marked as "Out of Stock." Attempt to click the "Add to Cart" button.
- Expected Result: The "Add to Cart" button should be disabled or greyed out. If clickable, it should display an appropriate error message like "This item is currently out of stock" and prevent the item from being added.

#### = Adding a Product with a Missing Required Variant Selection:

- Test Steps: Navigate to a product page that requires a variant selection (e.g., size, color, storage capacity) before purchase. Without making any selection, click the "Add to Cart" button.
  - Expected Result: An error message should prompt the user to select a required variant (e.g., "Please select a size" or "Choose a color"). The item should not be added to the cart.
- = Adding Quantity Exceeding Maximum Order Limit (if applicable):

- **Test Steps:** Assume a product has a backend limit of 10 units per order. Navigate to its product detail page and attempt to enter '11' (or more) in the quantity field. Click "Add to Cart."
- **Expected Result:** An error message should display, indicating the maximum allowed quantity (e.g., "You can order a maximum of 10 units"). The system should prevent adding more than the limit.

= Adding to Cart After Session Expiration (Simulated):

- **Test Steps:** Navigate to a product detail page. Simulate a session timeout (e.g., by waiting for a period known to exceed the session limit if testing permits, or using developer tools to invalidate a session). Attempt to click

= "Add to Cart."

- **Expected Result:** The user should be prompted to log in again or receive a clear message indicating the session has expired. The "Add to Cart" action should fail, and the item should not be added to the cart without re-authentication.

---

## Usability Test Cases

### Prompt 3:

#### INPUT

"Propose three usability test cases for an e-commerce product detail page, considering aspects like navigation, clarity of information, and overall user experience."

Usability Test Cases for E-commerce Product Detail Page 1, Ease of Locating and Understanding Key Purchase Information:

- **Test Goal:** To ensure users can quickly find critical information needed to make a purchase decision.
- **Test Steps:** Observe a new user as they land on a product detail page. Ask them to identify the product price, availability/stock status, and the "Add to Cart" button. Ask if they feel confident about where to click to buy the product.
- **Expected Result:** The user should locate all three pieces of information within 5 seconds without significant scrolling or searching. The "Add to Cart" button should be prominently visible and clearly indicate its function.

2, Clarity and Interaction with Product Images/Gallery:

- **Test Goal:** To verify that users can effectively view and interact with all product images.
- **Test Steps:** Ask a user to browse through all available product images (if there's a gallery), and then attempt to zoom in on an image to see details (if this feature exists).
- **Expected Result:** The user should easily understand how to navigate between images (e.g., using arrows, clicking thumbnails) and successfully use the zoom functionality without

confusion or frustration. The image quality should remain high when zoomed.

### 3, Understanding and Selecting Product Options/Variants:

- **Test Goal:** To assess how easily users can understand and select different product variations (like size, color, or capacity).
  - **Test Steps:** For a product with multiple options, ask the user to select their preferred size, color, and/or other variants. Observe if they understand what each option means and if their selection is clearly reflected (e.g., by updating the main product image or price).
  - **Expected Result:** The user should correctly select their desired variants on the first attempt. The selected options should be clearly highlighted, and any associated changes (like price adjustments) should be immediately visible and understandable.
- 

## Performance Test Scenarios

### Prompt 4:

"Outline two high-level performance test scenarios for an e-commerce product detail page. Consider page load times under concurrent user loads and quick navigation."

#### High-Level Performance Test Scenarios for E-commerce Product Detail Page

##### 1, Spike Load on a "Hot" Product Page:

= Scenario:

- Simulate a sudden, sharp increase in traffic (a "spike") directed specifically at a single, highly popular product detail page (e.g., during a flash sale or a product launch announcement). This involves a rapid ramp-up of users viewing that specific page.

= Objective:

- To determine how the product detail page and underlying infrastructure (database queries for product info, image servers, content delivery network (CDN)) cope with an immediate and intense surge of interest for a single item. This checks for resilience and stability under sudden peak demand.

= Key Metrics:

- Response time for the specific "hot" product page during the spike, number of successful requests vs. errors, database query performance, and server resource utilization (CPU, memory, network I/O) specifically for that page's serving components.

##### 2, Long-Duration Endurance Test with Baseline Load & User Navigation:

- **Scenario:** Maintain a steady, moderate baseline load of concurrent users over an extended period (e.g., 4-8 hours) on the e-commerce site. Within this load, a subset of users are specifically navigating between multiple different product detail pages and interacting with common elements (like adding to cart, switching images).

- **Objective:** To uncover performance degradation over time, such as memory leaks, inefficient caching, or database connection pool exhaustion, that might not appear in short-duration tests. It also assesses the sustained performance and responsiveness of the page for typical user journeys involving product Browse.
  - **Key Metrics:** Consistent average page load times for product detail pages over the extended period, stability of server resource utilization, identification of any gradual increase in error rates, and the effectiveness of caching mechanisms for frequently accessed product data.
- 

## Security Test Cases - Basic

### Prompt 5:

#### 1,SQL Injection Attempt in Product ID/URL Parameter:

- **Test Steps:** Navigate to a product detail page. Identify the product ID in the URL (e.g., [www.example.com/product?id=123](http://www.example.com/product?id=123)). Attempt to append basic SQL Injection payloads to the id parameter (e.g., `?id=123' OR '1'='1` or `?id=123; DROP TABLE products;`). Observe the page's behavior and any error messages.
- **Expected Result:** The page should not display any database errors, sensitive information, or unexpected changes. The SQL query should either be properly sanitized, causing the page to return no product, an error page, or simply the intended product 123 (ignoring the injected part). The database should not be affected.
- **Vulnerability Focus:** Input sanitization, database security, error handling.

#### 2,Accessing Unlisted/Disabled Product Pages (IDOR - Insecure Direct Object Reference):

- **Test Steps:** Identify a valid product ID from a known product page (e.g., id=123). Then, attempt to manually change this product ID in the URL to a suspected unlisted, disabled, or internal product ID (e.g., id=99999 if you suspect a high ID, or id=archived\_product). Also, try sequential IDs.
  - **Expected Result:** If the product ID corresponds to an unlisted or disabled product, the page should return a "Product Not Found" (404) error, an "Access Denied" message, or redirect to a generic product listing page. It should not display details of products that are not publicly available or authorized for view.
  - **Vulnerability Focus:** Authorization, secure direct object references, access control.
- 

### Bias

Bias in LLM-generated test cases could manifest in several ways:

- **Overemphasis on common scenarios:** The LLM might prioritize test cases based on the prevalence of similar data in its training, potentially overlooking less common but critical user flows or edge cases.

- Reflecting "best practices" over specific context: The LLM might generate generic best-practice test cases that don't fully align with the unique requirements or design of the specific e-commerce page.
  - Missing accessibility considerations: If the training data doesn't heavily emphasize accessibility testing, the LLM might consistently omit or under-represent test cases related to screen readers, keyboard navigation, or color contrast.
- 

## Limitations

Several limitations exist when using LLMs for test case generation:

- Lack of Domain Expertise: LLMs don't understand the underlying business logic, system architecture, or specific product requirements. They can only generate test cases based on patterns in their training data, not true comprehension. This is particularly evident in security and performance testing where a deep understanding of system vulnerabilities or bottlenecks is crucial.
  - Inability to "Think Outside the Box": While good at pattern recognition, LLMs are not inherently creative in identifying novel failure modes or highly specific, unusual user behaviors. They won't explore scenarios that aren't "seen" in their training.
  - No Test Environment Awareness: LLMs are unaware of the actual test environment, data dependencies, or specific configurations, all of which are vital for designing executable test cases.
  - Requirement for Human Oversight: The generated test cases always require significant human review, refinement, and validation to ensure accuracy, completeness, and relevance. They cannot replace a skilled QA engineer.
- 

## Reliability

The reliability of LLM-generated test cases is a significant concern:

- Varies by Prompt Quality: The more precise and detailed the prompt, the more reliable the output is likely to be. Generic prompts often lead to generic or less useful test cases.
  - Consistency Issues: Running the same prompt multiple times might yield slightly different results, especially for more open-ended prompts, indicating a lack of deterministic behavior.
  - Accuracy of "Expected Results": While LLMs can suggest test steps, their accuracy in defining precise "expected results" (which are crucial for test case execution) can be low, as they don't know the system's exact behavior. This often requires significant human intervention.
  - Dependence on Training Data: If the training data contains biases or outdated testing practices, the LLM's output will reflect those issues, impacting reliability.
- 

## Conclusion

While LLMs show promising potential as a powerful assistant in software QA, particularly for brainstorming and generating routine test cases, they are far from a complete solution. Their role is to augment, not replace, human QA engineers. Leveraging LLMs for initial ideation and then relying on human expertise for critical thinking, deep analysis, and context-specific refinement will be key to effectively integrating them into the software testing lifecycle