```
Start coding or generate with AI.
```

MINI PROJECT- PROMPT ENGINEERING FOR SOFTWARE QUALITY ASSURANCE

1.Introduction

Software Quality assurance(QA) is a crucial phase in software development where testers validate that the product meets functional and non functional requirements. With the rise of LLMs, prompt engineering can help automate parts of qA tasks like test case generation, bug summarization, and requirement analysis.

This project explores how LLMs can assist in:

- generating test cases from requirements
- summarizing bug reports
- automating test documentation

2.Prompt design

The following prompt types are used:

- Instructional
- Role based
- Few shot
- chain-of-thought

Prompt 1

Test case generation

Input text:

Generate 5 functional test cases for a login feature where users enter a username and password.

Result

Test Case 1: Valid Username and Valid Password Objective: Verify that a user can log in with valid credentials. Steps:

Enter a valid username.

Enter the correct corresponding password.

Click the "Login" button. Expected Result: User is successfully logged in and redirected to the home/dashboard page.

Test Case 2: Valid Username and Invalid Password Objective: Verify that the system prevents login with an incorrect password. Steps:

Enter a valid username.

Enter an incorrect password.

Click the "Login" button. Expected Result: User is not logged in. An error message is displayed (e.g., "Invalid username or password.").

Test Case 3: Invalid Username and Any Password Objective: Verify that the system prevents login with a non-existent username. Steps:

Enter an invalid or non-registered username.

Enter any password.

Click the "Login" button. Expected Result: User is not logged in. An error message is displayed (e.g., "Invalid username or password.").

Test Case 4: Empty Username and/or Password Fields Objective: Verify the system handles empty input fields gracefully. Steps:

Leave the username field empty and/or the password field empty.

Click the "Login" button. Expected Result: User is not logged in. A validation message is displayed (e.g., "Username and password are required.").

Test Case 5: Username and Password with Special Characters Objective: Verify that the system properly handles special characters in input fields. Steps:

Enter a username with special characters (if allowed) or normal username.

Enter a password containing special characters (e.g., P@ssw0rd!#).

Click the "Login" button. Expected Result: If the credentials are valid, the user is logged in successfully. If invalid, an appropriate error message is displayed.

---

prompt 2

Edge case discovery

Input text

List edge test cases for a mobile app's file upload feature that accepts PDF files up to 20MB.

Result

File Size Boundary Tests File size = 0 bytes (empty PDF)

Expected: Upload rejected with an appropriate error message.

File size = 1 byte (corrupted/invalid PDF)

Expected: Upload rejected or handled gracefully with a message.

File size = just under 20MB (e.g., 19.99MB)

Expected: Upload succeeds.

File size = exactly 20MB

Expected: Upload succeeds.

File size = just over 20MB (e.g., 20.01MB)

Expected: Upload rejected with an appropriate error message (e.g., "File exceeds size limit.").

🧩 File Format/Extension Tests File with .pdf extension but non-PDF content (e.g., renamed .exe)

Expected: Upload rejected after validating file content (not just extension).

File with valid PDF content but incorrect extension (e.g., .txt)

Expected: Upload rejected or corrected if the app strictly enforces .pdf extension.

File with mixed-case extension (e.g., .PDF, .Pdf)

Expected: Upload accepted if content is valid PDF.

📱 Device/Network Edge Tests Upload while on slow mobile network (e.g., 2G or unstable Wi-Fi)

Expected: App shows progress and retries gracefully or shows timeout error.

Upload interrupted by app backgrounding (user switches apps mid-upload)

Expected: Upload resumes if supported, or user is notified and can retry.

Device storage nearly full when selecting file

Expected: App does not crash; appropriate error if temp storage is required.

🔐 Security/Corruption Edge Cases Upload PDF with malicious content (e.g., embedded scripts)

Expected: App rejects the file or sanitizes input to prevent security risks.

Upload PDF with password protection

Expected: App accepts or rejects based on app policy; if rejected, provide clear message (e.g., "Password-protected files are not supported.").

🛠️ Other Edge Cases Upload same file multiple times in a row

Expected: App handles gracefully (e.g., shows duplicates or warns user if not allowed).

Upload when device battery is critically low

Expected: App handles interruption gracefully and does not corrupt user session.

---

prompt 3

Bug report summarization

input text

You are a QA lead. Summarize the following bug report in 3 bullet points: "User clicks 'Export CSV' on reports page. The app freezes for 15 seconds before generating an incomplete file with missing rows.

Result

As a QA lead, here is a summary of the bug report in 3 bullet points:

App freezes for ~15 seconds when user clicks 'Export CSV' on the reports page.

The generated CSV file is incomplete, with missing rows.

Issue impacts performance (UI unresponsiveness) and data integrity (missing export data).

Prompt 4

Suggest regression tests

input text

A recent update modified the user profile section. Suggest 5 regression test cases to ensure old functionality still works.

Result

Here are 5 regression test cases to ensure that the old functionality of the user profile section still works after the recent update:

1️⃣ View User Profile Information Objective: Ensure all user profile fields (name, email, phone, etc.) are correctly displayed. Steps:

Navigate to the user profile section.

Verify that all fields are visible and populated with correct data. Expected Result: Profile information is displayed accurately.

2️⃣ Edit and Save User Profile Information Objective: Verify that users can still update profile details. Steps:

Navigate to the profile edit screen.

Change fields such as name, phone number, or address.

Save the changes. Expected Result: Changes are saved and reflected correctly in the profile view.

3️⃣ Profile Picture Upload/Update Objective: Ensure that uploading or changing the profile picture still works. Steps:

Attempt to upload a new profile picture.

Save changes.

Refresh or re-open the profile section. Expected Result: New profile picture is displayed and persists after reload.

4️⃣ Email Verification Workflow Objective: Confirm that email verification link/process still functions correctly (if applicable). Steps:

Change the user's email address.

Trigger email verification flow.

Complete the verification through received email link. Expected Result: Email verification is successful, and new email is updated in profile.

5️⃣ Mobile Responsiveness of Profile Section Objective: Verify that the profile section UI remains responsive and functional on various mobile devices. Steps:

Access the profile section on different device sizes (phones, tablets).

Perform basic interactions (view, edit, upload picture). Expected Result: Profile section is fully usable and visually correct on all supported devices.

Prompt 5

Diagnose a bug

input text

A QA tester reports: "The 'Save' button is disabled even after all required fields are filled." Think step-by-step to diagnose the possible reasons for this issue.

Result

Here's a step-by-step diagnostic approach to analyze why the 'Save' button remains disabled even after required fields are filled:

🟢 1️⃣ UI/UX Verification Check required field validation visually Are all fields truly filled in correctly according to UI indicators (e.g., no red borders, no error messages)?

Hidden required fields Are there hidden fields or dynamic fields that the user may not see but are still required?

🔵 2️⃣ Form State Logic / Frontend Code Form validation logic Is the form's internal state (isFormValid or similar) properly updated when fields change?

Event listeners / bindings Are input fields correctly wired to update the form state? Sometimes incorrect onChange or debounce logic causes stale state.

JavaScript errors Are there any JavaScript errors in the console that prevent form updates?

🟣 3️⃣ Required Field Types & Format Data format issues Are fields filled with invalid formats (e.g., wrong email format, phone number too short), causing validation to fail silently?

Whitespace issues Are fields containing only spaces, which may not count as valid input?

Autofill handling Is browser autofill not triggering validation updates properly?

🟠 4️⃣ 'Save' Button Logic Enable/disable logic Is the button's enabled state dependent on a correct validation flag or state? Example: disabled={!isFormValid} — check if isFormValid is being computed correctly.

Timing / asynchronous validation Is asynchronous validation (e.g., checking username uniqueness) delaying form validation but not updating the button state correctly?

🔴 5️⃣ Backend/API Factors (if applicable) Server-side validation blocking the button Is there client-side blocking logic based on a server response that is stuck or failing silently?

API call race conditions Is the button state dependent on an API response that never arrives or is invalid?

Summary Possible root causes fall into these buckets: ✅ User not truly completing required fields ✅ Incorrect frontend state/validation logic ✅ Formatting/whitespace/autofill issues ✅ Faulty button enable/disable logic ✅ Pending API responses or asynchronous validations

---

Prompt 6

Translate Requirements to Test cases

input text

Convert the following requirement into 3 clear test cases: "Users must reset their password using a link that expires after 15 minutes."

Result

✅ Test Case 1: Password Reset Link Valid Within 15 Minutes Objective: Verify that the password reset link works correctly within the 15-minute validity period. Steps:

Request a password reset email.

Open the password reset link within 15 minutes of receiving it.

Enter a new password and submit. Expected Result: Password is successfully reset. User is notified of success.

✅ Test Case 2: Password Reset Link Expired After 15 Minutes Objective: Verify that the password reset link becomesprompt 7 invalid after 15 minutes. Steps:

Request a password reset email.

Wait for more than 15 minutes.

Attempt to open the password reset link and reset the password. Expected Result: Link is rejected with an appropriate message (e.g., "This link has expired. Please request a new one.").

✅ Test Case 3: Attempt to Reuse Expired Link After Successful Reset Objective: Ensure that an expired or already-used link cannot be reused. Steps:

Request a password reset email.

Use the link and successfully reset the password within 15 minutes.

Attempt to use the same link again after the password reset. Expected Result: Link is invalidated and an error message is shown (e.g., "This link is no longer valid.").

Prompt 7

Generate qa checklist

input text

As a senior QA engineer, create a basic test checklist for the release of a new e-commerce checkout flow.

Result

✅ 1️⃣ Cart & Checkout Initiation Items correctly added/removed from cart

Cart totals (subtotal, tax, shipping, discounts) correctly calculated

Cart persists across sessions (if expected)

"Proceed to Checkout" button enabled when cart has valid items

✅ 2️⃣ Guest vs. Logged-in User Flow Guest checkout flow works end to end

Logged-in user checkout flow works end to end

Login/register prompt appears where appropriate

Previously saved user info (address, payment) prepopulates correctly

✅ 3️⃣ Shipping Information User can enter/edit shipping address

Address validation works as expected (postal code formats, required fields)

Shipping methods are displayed based on address

Correct shipping fees calculated

✅ 4️⃣ Billing & Payment Information User can enter/edit billing address

User can select payment method (credit card, PayPal, wallet, etc.)

Payment gateway integration works (test with success/failure scenarios)

Validations work (e.g., expired credit card, incorrect CVV)

Promo codes/discounts applied correctly

✅ 5️⃣ Order Review & Confirmation Correct summary displayed (items, prices, discounts, tax, shipping)

User can review all entered data before confirming order

"Place Order" button enabled only when required info is complete

Successful order confirmation displayed with order ID

Confirmation email sent with correct order details

✅ 6️⃣ Edge & Negative Cases Attempt to checkout with empty cart — appropriate message shown

Network error during payment — graceful handling and user messaging

Duplicate order prevention (e.g., double-click "Place Order")

Session timeout handling during checkout

Browser refresh/back button behavior during checkout flow

✅ 7️⃣ Cross-device / Cross-browser Testing Responsive checkout flow works on desktop, tablet, mobile

Functional across major browsers (Chrome, Firefox, Safari, Edge)

Checkout flow maintains usability on slower networks

✅ 8️⃣ Accessibility & UX Checks All form fields accessible via keyboard navigation

Screen reader compatibility

Clear and consistent error messages for invalid inputs

Checkout steps clearly indicated (progress bar or breadcrumb)

✅ 9️⃣ Analytics & Tracking Key events tracked (cart start, checkout start, order placed)

No duplicate or missing tracking events in checkout flow

✅ 1 0️⃣ Security Sensitive data (payment info) not exposed in network traffic or local storage

CSRF and XSS protections verified

PCI compliance verified for payment components

Promt 8

Create a test summary report

input text

Write a short QA test summary report for a sprint that included 3 bug fixes and 2 new feature additions.

Result

📝 QA Test Summary Report Sprint: Sprint 12 Date: June 8, 2025 Prepared by: QA Team

📌 Scope of Testing 3 Bug Fixes:

Checkout button unresponsive on mobile

Incorrect discount calculation in cart

Broken image links on product detail pages

2 New Feature Additions:

Wishlist functionality

Product review submission form

✅ Testing Activities Functional Testing (Web + Mobile)

Regression Testing across impacted modules

Cross-browser Compatibility Testing

Basic Performance Checks

🚧 Test Results Item Status Bug Fixes Verification Passed ✅ Wishlist Functionality Passed ✅ Product Review Submission Passed ✅ Regression Tests Passed ✅ No Critical or Major Defects Confirmed ✅

🚩 Issues Observed Minor cosmetic issue in Wishlist pop-up (low priority, logged for next sprint)

No show-stoppers or blockers found in this sprint

📌 Conclusion All sprint deliverables have been tested and verified.

The build is considered stable and ready for release to staging/production.

---

prompt 9

Create Negative test cases

input text

Feature: User registration form Input: Email, password, confirm password Output: List of 5 negative test cases (invalid/missing input)

Result

🚫 1️⃣ Missing Email Field Input:

Email: empty

Password: validPassword123

Confirm Password: validPassword123

Expected Result:

Error message displayed: “Email is required.”

Form submission blocked.

🚫 2️⃣ Invalid Email Format Input:

Email: user@invalid (missing domain)

Password: validPassword123

Confirm Password: validPassword123

Expected Result:

Error message displayed: “Please enter a valid email address.”

Form submission blocked.

🚫 3️⃣ Password Too Short Input:

Email: user@example.com

Password: 123 (too short, assuming minimum length is 8)

Confirm Password: 123

Expected Result:

Error message displayed: “Password must be at least 8 characters long.”

Form submission blocked.

🚫 4️⃣ Password and Confirm Password Do Not Match Input:

Email: user@example.com

Password: validPassword123

Confirm Password: validPassword321

Expected Result:

Error message displayed: “Passwords do not match.”

Form submission blocked.

🚫 5️⃣ All Fields Empty Input:

Email: empty

Password: empty

Confirm Password: empty

Expected Result:

Error messages displayed for all required fields.

Form submission blocked.

Prompt 10

Explain test result discrepancy

input text

Test case expected behavior: "Email confirmation link redirects to dashboard." Actual result: "Link leads to 404 error page." Explain step-by-step how you would investigate this.

Result

🟢 1️⃣ Verify the Reproduction Re-test using a new email confirmation link.

Confirm whether the issue is consistent or intermittent.

Try on different browsers/devices to rule out client-side caching.

🔵 2️⃣ Check the Email Link Content Inspect the email and copy the confirmation link.

Check for obvious issues:

Malformed URL (e.g., missing token, broken characters, extra slashes).

Expired token.

Incorrect domain (staging link in production email, etc.).

🟣 3️⃣ Test the Link Manually Paste the confirmation link directly into the browser.

Observe the full redirect chain using browser dev tools (Network tab):

Does it start a redirect?

Is the correct endpoint hit?

Where does it fail (which HTTP status codes are returned)?

🟠 4️⃣ Review Server Logs Check backend logs for the endpoint handling email confirmation.

Look for errors:

Missing/invalid token.

Unauthorized access.

404s or 500s being returned from the confirmation API.

🔴 5️⃣ Check Routing & Configuration Verify the route for the confirmation link on the server and in the frontend app.

Is the URL path /confirm-email correctly mapped?

Is there a mismatch in the expected URL vs. what was sent in the email?

Check if the page component handling the redirect to dashboard is present and properly deployed.

🟤 6️⃣ Review Recent Code Changes Check recent commits/deployments related to:

Email template generation.

Backend token validation logic.

Frontend route handling or redirects.

Investigate whether the recent release may have introduced this issue.

Summary The investigation typically follows this flow:

Reproduce and confirm → 2. Analyze email link → 3. Trace request/response path →

Check backend logs → 5. Verify route and app config → 6. Review recent changes

---

SHORT REPORT ON PROMPT ENGINEERING FOR SOFTWARE QUALITY ASSURANCE

Project Overview

Software Quality Assurance (QA) is a critical component of the software development lifecycle. It ensures that applications behave as intended, meet user expectations, and maintain reliability over time. Traditional QA involves writing and executing test cases, identifying bugs, preparing test plans, and documenting results — all of which are time-consuming and manual.

With the rise of large language models (LLMs) like GPT, there's an opportunity to use prompt engineering to automate or augment these processes.

This mini project explores how prompt engineering techniques can be applied to Software Quality Assurance (QA). With the increasing complexity of software systems and the rise of large language models (LLMs), prompt-based workflows can support QA teams by automating tasks such as test case generation, bug report summarization, edge case discovery, and root cause analysis.

The project involved crafting and testing 10 diverse prompts using different strategies—instructional, role-based, few-shot, and chain-of-thought—to simulate common QA workflows.

Objectives

- To design effective prompts tailored for typical QA activities.
- To evaluate the quality and usefulness of outputs from an LLM.
- To reflect on the limitations, bias, and reliability of using LLMs in QA.

Prompt Types & Tasks

Prompt Focus	Prompt Type
Functional test case generation	Instructional

Prompt Focus	Prompt Type
Edge case discovery	Instructional
Bug report summarization	Role-based
Regression testing suggestions	Instructional
Debugging with reasoning	Chain-of-thought
Translating requirements into test cases	Instructional
QA checklist creation	Role-based
Writing test summary reports	Instructional
Negative test case creation	Few-shot
Root cause analysis	Chain-of-thought

The tasks covered the following QA scenarios:

- Creating test cases for login, registration, and file upload
- Writing regression tests
- Summarizing bugs from logs and reproduction steps
- Diagnosing test discrepancies using reasoning steps
- Creating QA reports and test summaries

Key Observations

- *Prompt Clarity Matters* Direct, concise prompts with clear intent produced the best outputs. Including context (like feature details) improved accuracy.
- *Role-based Prompts Improve Tone* Asking the LLM to “act as a QA lead” made outputs more concise and professional.
- *Few-shot Prompts Boost Creativity* When showing examples of negative test cases, the LLM generated more diverse and thoughtful suggestions.
    - *Chain-of-Thought Reasoning Helps Debugging* In prompts where a test case failed, asking the model to think step-by-step helped it simulate logical QA workflows.

Prompt performance Analysis

- Prompts with clear task structure performed better
- Few shot and chain of thought formats improved response richness
- Mostly accurate, but occasional oversimplifications in bug analysis
- All prompts were easily tunable

Bias

- Bias towards comman platforms
- The model occasionally assumed standard behaviour not present in the described feature.

Limitations

- Not all QA edge cases are well coveres
- No ability to test actual software

Reliability

- Highly reliable for:

```
- Generating test cases
- Summarize long reports
```

- Less reliable for:

```
- Deep technical root cause analysis
- Specific environmental behavior prediction
```

Key findings

Strengths of Prompt-Based QA

- *Speed & Efficiency* The model quickly generated structured test cases and summaries that saved significant manual effort.
- *Clarity with Role Prompts* Prompts like “Act as a QA lead…” led to domain-appropriate tone and vocabulary.
- *Helpful Debug Simulation* Chain-of-thought prompts helped model simulate reasoning in bug diagnosis.
- *Adaptability Across Contexts* Prompts were flexible and could be adapted for different features, platforms, or test types.

Conclusions

Through this prompt design and experimentation exercise, I gained valuable insights into how different types of prompts influence the behavior of large language models (LLMs).
Basic prompts such as text summarization and code generation produced consistent results, while advanced techniques like chain-of-thought and role prompting allowed for more structured and nuanced responses.

Key learnings include:

- The importance of **precise and clear prompt wording**
- How adding specific instructions significantly improves output quality

- The role of **iterative refinement** in optimizing prompt effectiveness
- Limitations such as occasional lack of reasoning depth or repetitive outputs

Prompt engineering is a powerful skill that enables users to better control and utilize LLMs across a wide range of applications.
Continuous experimentation and thoughtful prompt design can further enhance the reliability and usefulness of AI-generated content.

Prompt engineering can support QA efforts in:

- Accelerating test design
- Summarizing complex bugs quickly
- Generating ideas for edge cases
- Assisting in documentation and reporting
