Appendix to Automatic High-Level Test Case Generation using Large Language Models

1 Prompting Strategy

The prompts were developed iteratively, starting with a straightforward task description, such as: "Generate comprehensive test cases for the given use case". During the refinement stage, the authors manually evaluated the outputs to assess their relevance, correctness, and alignment with the expected structure. Based on the evaluation, the prompts were iteratively refined by incorporating prompt specificity [2, 3], and one-shot learning [1].

1. **Prompt Specificity:** We specified the desired role of the LLM with instructions like "You are a tester tasked with creating comprehensive test cases for a given use case description below". We also added specific guidelines to ensure that the generated test cases comprehensively covered (i) basic and edge cases, (ii) both positive and negative scenarios, and (iii) valid and invalid inputs. We also specified that the generated test cases must be JSON-parsable to ensure consistency and seamless integration into our downstream technique.
2. **One-Shot Learning:** Each prompt included one example of a use case paired with its corresponding high-level test cases to guide the model. This one-shot example demonstrated the desired structure and style of the output.

Based on the nature of the task, we did not adopt complex prompting strategies such as Chain-of-Thought [4], Tree-of-Thought [6], or ReAct [5] because generating high-level test cases does not require the complex reasoning or multi-step derivations that these approaches are designed for. Additionally, there is a trade-off between cost and accuracy when using such strategies, as they typically consume a large number of tokens and lead to higher computational expenses.

2 Examples of Generated Test-cases

2.1 Example-1

2.1.1 Use-case

Use-case: Sign Up for Account
Scenario: Users can sign up for an account using their details.
Actor(s): New User
Preconditions: User does not have an existing account.

2.1.2 GPT-4 Generated Test-cases

1. Successful Sign Up
- Description: Verify that a new user can successfully sign up for an account with valid details.
- Input: 'fullName': 'Alice Smith', 'email': 'alice.smith@example.com', 'phoneNumber': '0123456789', 'password': 'SecurePass123', 'confirmPassword': 'SecurePass123'
- Expected Output: 'outcome': 'Sign up successful', 'status': 'Account Activated'

2. Sign Up with Existing Email
- Description: Verify that a user cannot sign up with an email that is already registered.
- Input: 'fullName': 'Bob Johnson', 'email': 'existing.email@example.com', 'phoneNumber': '0987654321', 'password': 'AnotherPass123', 'confirmPassword': 'AnotherPass123'

- Expected Output: 'outcome': 'Sign up failed', 'status': 'Email Already Registered'

3. Sign Up with Mismatched Passwords
- Description: Verify that a user cannot sign up if the password and confirm password do not match.
- Input: 'fullName': 'Charlie Brown', 'email': 'charlie.brown@example.com', 'phoneNumber': '0112233445', 'password': 'Password123', 'confirmPassword': 'Password321'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Password Mismatch'

4. Sign Up with Invalid Email Format
- Description: Verify that a user cannot sign up with an invalid email format.
- Input: 'fullName': 'David Green', 'email': 'david.green@com', 'phoneNumber': '0223344556', 'password': 'ValidPass123', 'confirmPassword': 'ValidPass123'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Invalid Email Format'

5. Sign Up with Empty Fields
- Description: Verify that a user cannot sign up if any required field is empty.
- Input: 'fullName': '', 'email': 'empty.fields@example.com', 'phoneNumber': '0334455667', 'password': 'EmptyPass123', 'confirmPassword': 'EmptyPass123'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Missing Information'

2.1.3 LLaMA Generated Test-cases

1. Successful Sign Up with Unique Email
- Description: Verify that a user can successfully sign up for an account with a unique email.
- Input: 'email': 'uniqueemail@example.com', 'name': 'John Doe', 'phoneNumber': '1234567890', 'password': 'SecurePassword123'
- Expected Output: 'outcome': 'Account created successfully', 'status': 'Verified'

2. Successful Sign Up with Unique Phone Number
- Description: Verify that a user can successfully sign up for an account with a unique phone number.
- Input: 'phoneNumber': '9876543210', 'name': 'Jane Smith', 'email': 'janedoe@example.com', 'password': 'SecurePassword456'
- Expected Output: 'outcome': 'Account created successfully', 'status': 'Verified'

3. Sign Up with Registered Email
- Description: Verify that the system prevents sign up with an email that is already registered.
- Input: 'email': 'registeredemail@example.com', 'name': 'John Doe', 'phoneNumber': '1234567890', 'password': 'SecurePassword123'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Already Registered'

4. Sign Up with Registered Phone Number
- Description: Verify that the system prevents sign up with a phone number that is already registered.
- Input: 'phoneNumber': '9876543210', 'name': 'Jane Smith', 'email': 'janedoe@example.com', 'password': 'SecurePassword456'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Already Registered'

5. Sign Up with Weak Password
- Description: Verify that the system prevents sign up with a weak password.
- Input: 'email': 'uniqueemail@example.com', 'name': 'John Doe', 'phoneNumber': '1234567890', 'password': 'abcxyz'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Weak Password'

6. Sign Up with Invalid Email Format

- Description: Verify that the system prevents sign up with an invalid email format.
- Input: 'email': 'invalidemail', 'name': 'Jane Smith', 'phoneNumber': '9876543210', 'password': 'SecurePassword456'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Invalid Email Format'

7. Sign Up with Invalid Phone Number Format
- Description: Verify that the system prevents sign up with an invalid phone number format.
- Input: 'phoneNumber': 'abc123', 'name': 'John Doe', 'email': 'john@example.com', 'password': 'SecurePassword123'
- Expected Output: 'outcome': 'Sign up failed', 'status': 'Invalid Phone Number Format'

2.2 Example-2

2.2.1 Use-case

Use-case: Update payment method Scenario: A user wants to update their payment method for the subscription plan Actor(s): User Preconditions: User is loggged in

2.2.2 GPT-4 Generated Test-cases

1. Successful Payment Method Update
- Description: Verify that a user can successfully update their payment method
- Input: 'userId': 'user_67890', 'cardNumber': '4111111111111111', 'expiryDate': '12/25', 'cvv': '123', 'billingAddress': '123 Main St, City, Country'
- Expected Output: 'outcome': 'Payment method update successful', 'status': 'Payment Method Updated'

2. Failed Payment Method Update - Missing Card Number
- Description: Verify that a user cannot update their payment method if the card number is missing
- Input: 'userId': 'user_67890', 'cardNumber': None, 'expiryDate': '12/25', 'cvv': '123', 'billingAddress': '123 Main St, City, Country'
- Expected Output: 'outcome': 'Payment method update failed', 'status': 'Invalid Card Information'

3. Failed Payment Method Update - Invalid Expiry Date
- Description: Verify that a user cannot update their payment method if the expiry date is invalid
- Input: 'userId': 'user_67890', 'cardNumber': '4111111111111111', 'expiryDate': '13/25', 'cvv': '123', 'billingAddress': '123 Main St, City, Country'
- Expected Output: 'outcome': 'Payment method update failed', 'status': 'Invalid Expiry Date'

4. Failed Payment Method Update - Incorrect CVV
- Description: Verify that a user cannot update their payment method if the CVV is incorrect
- Input: 'userId': 'user_67890', 'cardNumber': '4111111111111111', 'expiryDate': '12/25', 'cvv': '12', 'billingAddress': '123 Main St, City, Country'
- Expected Output: 'outcome': 'Payment method update failed', 'status': 'Invalid CVV'

5. Failed Payment Method Update - Missing Billing Address
- Description: Verify that a user cannot update their payment method if the billing address is missing
- Input: 'userId': 'user_67890', 'cardNumber': '4111111111111111', 'expiryDate': '12/25', 'cvv': '123', 'billingAddress': None
- Expected Output: 'outcome': 'Payment method update failed', 'status': 'Missing Billing Address'

2.2.3 LLaMa Generated Test-cases

1. Valid Payment Method Update
- Description: Verify that a user can successfully update their payment method for the subscription plan.
- Input: 'user_id': '2005006', 'current_method': 'Credit Card', 'new_method': 'PayPal', 'confirmation_status': 'Confirmed'
- Expected Output: 'outcome': 'Payment method updated successfully', 'status': 'Updated', 'confirmation_message': 'User receives a confirmation message along with an email', 'email_confirmation': "Email confirmation sent to the user's registered email"

2. Invalid Payment Method Update
- Description: Verify that the system handles the case where the user tries to update the payment method with an invalid or unsupported method.
- Input: 'user_id': '2005006', 'current_method': 'Credit Card', 'new_method': 'InvalidMethod', 'confirmation_status': 'Not Confirmed'
- Expected Output: 'outcome': 'Payment method update failed', 'status': 'Not Updated', 'confirmation_message': 'User does not receive a confirmation message', 'email_confirmation': 'No email confirmation sent to the user'

3 Prompt with Project Description

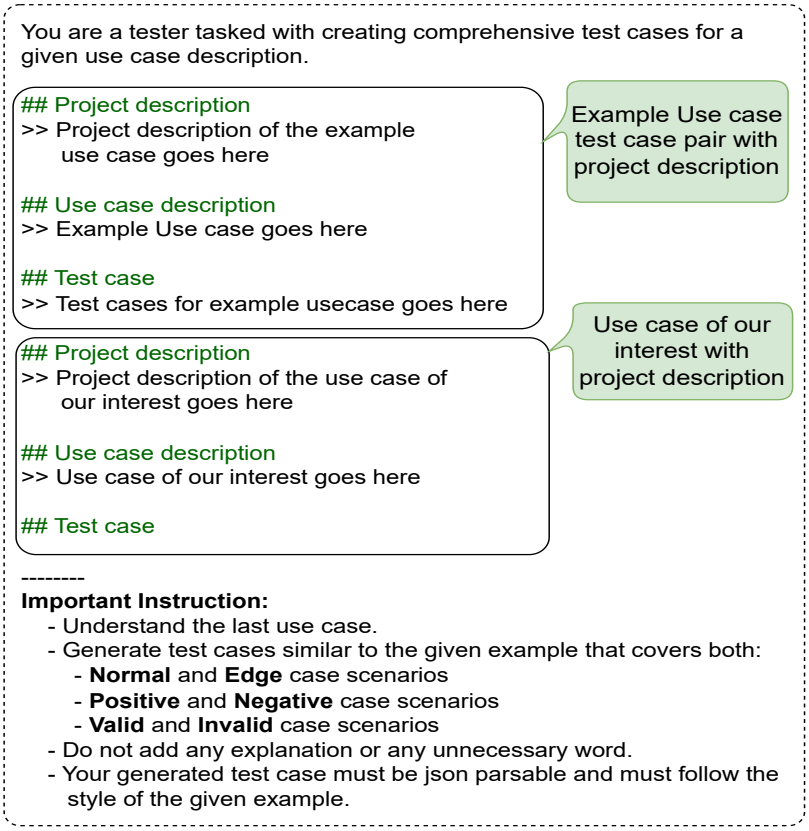

Figure 1: Prompt for generating test cases from a use case and the project description using pre-trained LLMs.

References

[1] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[2] Lincoln Murr, Morgan Grainger, and David Gao. "Testing LLMs on Code Generation with Varying Levels of Prompt Specificity". In: *arXiv preprint arXiv:2311.07599* (2023).

[3] Chenglei Si et al. "Prompting gpt-3 to be reliable". In: *arXiv preprint arXiv:2210.09150* (2022).

[4] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 24824–24837.

[5] Shunyu Yao et al. "React: Synergizing reasoning and acting in language models". In: *arXiv preprint arXiv:2210.03629* (2022).

[6] Shunyu Yao et al. "Tree of thoughts: Deliberate problem solving with large language models". In: *arXiv preprint arXiv:2305.10601* (2023).