Test Plan

ABSTRACT

This document provides a guide for creating and managing test cases that are based on system requirements and specifications. It covers methods for generating test cases, compares black-box and white-box testing, and explains how to link tests back to specific use cases. With a focus on quality, standards, and managing test changes, this document aims to help teams build reliable and effective test plans.

TABLE OF CONTENTS

INTRODUCTION

The purpose of this document is to outline a clear process for developing system-level test cases based on what the system is required to do. It provides guidance on creating quality tests, choosing between black-box and white-box testing methods, and ensuring that each test links back to a specific requirement or use case. This document also explains how to keep test plans organized under configuration management so that changes are tracked and controlled. Covered topics include quality standards for tests, handling constraints like time and resources, and the role of engineering standards. This guide is intended for engineers, quality teams, and managers looking to improve their testing practices and ensure that tests fully support the system's requirements.

REQUIREMENTS/SPECIFICATIONS-BASED SYSTEM LEVEL TEST CASES.

Test Case ID	**Test Description**	**Test Data**	**Expected Result**	**Pass/Fail**
TC01	Verify the app requests location permissions on startup	Launch the app	App prompts the user to allow location permissions.	Pass
TC02	Verify the map centers on the user's current location	Grant location permission	Map should display the user’s current location as the center of the screen.	Pass
TC03	Verify destination search displays marker on map	Enter a valid address	Marker appears on the map at the destination address.	Pass
TC04	Verify arrival notification at the destination	Set destination nearby and simulate driving toward it	App shows a notification indicating arrival at the destination.	Pass
TC05	Verify app’s behavior on invalid address entry	Enter invalid address	App shows an error message or prompt requesting a valid address.	
TC06	Verify map’s responsiveness and accuracy	Use simulated driving route	Map follows the route accurately and updates the user's position in real-time without delays	Pass

TC07	Verify that the supervisor can export reports in both PDF and CSV format	Supervisor logged into the admin dashboard, navigates to reporting section, selects a time period (e.g., weekly)	The report is generated and successfully exported as a PDF and CSV file	
TC08	Verify that the app detects nearby mobile devices	Service personnel within range of a customer's phone, Bluetooth or Wi-Fi enabled on both devices	The system detects and lists the customer's phone as a nearby device	
TC09	Verify the software download process to customer's phone	Wi-Fi connected on customer device, and sufficient storage space available	Software is downloaded successfully onto the customer's phone	
TC10	Verify route display from user to delivery personnel	Enter user and delivery personnel locations	App displays the route from the user's location to the delivery personnel's location	
TC11	Verify user receives estimated time of arrival (ETA) updates	Track delivery personnel and simulate changes in speed or route	App updates the ETA to the user based on the current location and route	
TC12	Verify app's behavior when switching between different network connections	Switch from Wi-Fi to mobile data during tracking	App maintains real-time updates and notifies the user of the network change without interrupting the tracking process	

TECHNIQUES FOR TEST GENERATION

In designing test cases for our location-based application, we applied the following techniques for test generation:

Requirements/Specifications-Based Testing:

We designed test cases directly from the app's documented requirements to ensure its core functionality. Examples include:

- TC01: Verifies that the app requests location permissions when it starts.
- TC02: Ensures that the map centers on the user's location once permission is granted.
- TC04: Checks that an arrival notification appears when the user reaches their destination.

Boundary Value Analysis:

In this technique, we focused on testing edge cases to see how the app behaves at its limits. Examples include:

- TC04: Tests if the app correctly triggers notifications as the user gets close to their destination.
- TC05: Tests entering an invalid address to make sure the app prompts the user to provide a valid one.

Error Guessing:

We used error guessing to think of common mistakes users might make and created test cases for these possible failure points. Here are some examples:

- TC05: Tests how the app handles invalid addresses. We expect the app to show an error or prompt the user to enter a valid address.
- TC12: Switches network connections while tracking the user's location. This tests whether the app stays responsive and continues to update the location data correctly even when the network changes.

Code Coverage Analysis and Path Testing:

We used code coverage analysis to make sure we tested all important parts of the app's code. Path testing helped us confirm that the app's logic flows correctly.

- TC01: Tests if the app asks for location permissions when it starts. This checks if the permission request logic works the way it should.
- TC02: Verifies that the map centers on the user's location after permission is granted. This checks that the app correctly retrieves and displays the user's location.
- TC04: Makes sure the app sends an arrival notification when the user reaches the destination. This tests if the notification logic is triggered properly.

Scenario-Based Testing:

We tested the app based on how a real user might interact with it in different situations. For example:

- Scenario 1: We tested what happens when a user starts navigation, pauses it, and then resumes to make sure the app keeps tracking the location correctly and doesn't mess up.
- Scenario 2: We also tested how the app responds when the user quickly switches between locations, making sure the map updates smoothly and shows the correct location each time.

Black Box Vs White Box Testing

Black Box Testing

- Approach: Tests were designed based on the application's documented requirements to verify expected functionality from an end-user perspective.
- Techniques Used: Requirements/Specifications-Based Testing, Boundary Value Analysis, Error Guessing, and Scenario-Based Testing.
- Examples:
    - TC05: Testing how the app handles invalid addresses exemplifies black-box testing by ensuring proper behavior without insight into the internal code.

White Box Testing

- Approach: Focused on internal workings, analyzing code structure, logic, and execution paths for comprehensive coverage.
- Techniques Used: Code Coverage Analysis, Path Testing.
- Examples:
  - Use Case UC1: Tested specific methods like validateUser() and sendVerificationAlert(userId) for secure ID verification.
  - Use Case UC2: Verified internal functions like startTracking() and saveLocation(Location) for proper GPS tracking and data storage.

Quality Criteria for Tests

We focused on these key criteria to measure the quality of our test cases:

1. Accuracy
   Each test case is designed with clear and specific expected results, avoiding any ambiguity.
   - Example: TC02 specifies that the map should display the user's current location, centered on the screen, after location permission is granted.
2. Cross-Platform Compatibility
   Each test case is aligned with specific app requirements, ensuring a clear link between the app's specifications and the corresponding tests.
   - Example: TC07 directly maps to the requirement for supervisors to export reports in PDF and CSV formats.
3. Reliability
   Tests are designed to consistently produce the same results under the same conditions, ensuring stable app performance.
   - Example: TC06 verifies that the map updates in real-time along the user's route, with updates occurring without delay, even across different network conditions and tests.

TRACEABILITY OF TEST CASES TO USE CASES

Use Case 1 (UC1): Customer Verifies Service Personnel's Identity

Design Mapping:

1. **UserInput Class**: validateUser() handles verification requests using ID
2. **LocationTracking Class**: getCurrentLocation() fetches real-time GPS data to validate personnel's location.
3. **NotificationService Class**: sendVerificationAlert(userId) sends notifications to the customer

4. **Test Case 1**: Verify that validateUser() in userInput successfully handles ID verification
   - **Requirements Addressed**:
     - Verified Personnel Identity Using ID
     - Security (Multi-factor authentication for verification)
   - **Non-Functional Requirements**: Security (end-to-end encryption), User Interface (intuitive for easy biometric/ID verification)
5. **Test Case 2**: Verify that sendVerificationAlert(userId) in NotificationService sends a notification to the customer confirming the personnel's identity.
   - **Requirements Addressed**:
     - Sends Notification to the Customer Confirming Personnel's Identity

    - **Non-Functional Requirements**: Compatibility (cross-platform notifications), Performance (notification delivery within 2 seconds)
6. **Test Case 3**: Confirm that getCurrentLocation() in LocationTracking accurately fetches real-time GPS data to verify personnel's location.
    - **Requirements Addressed**:
        - Uses GPS to Validate Personnel's Location
    - **Non-Functional Requirements**: Performance (location fetched within 2 seconds), Reliability (accurate GPS updates)
7. **Test Case 4**: Validate that data used in identity verification and notifications is secure.
    - **Requirements Addressed**:
        - Security (Implements end-to-end encryption)
    - **Non-Functional Requirements**: Security (data encryption for all personal data)

Use Case 2 (UC2): Supervisor Tracks Service Personnel in Real-Time

Design Mapping:

- **LocationTracking Class**: startTracking() initiates location tracking; getCurrentLocation() fetches updated locations.
- **LocationDAO Class**: saveLocation(Location) stores location data.
- **NotificationService Class**: sendArrivalNotificaiton(userId) sends alerts for deviations in route or arrivals.

1. **Test Case 5**: Verify that startTracking() in LocationTracking initiates GPS tracking successfully.
    - **Requirements Addressed**:
        - Tracks Personnel's Location via GPS
    - **Non-Functional Requirements**: Battery Optimization (efficient GPS use), Reliability (continuous tracking)
2. **Test Case 6**: Confirm that getCurrentLocation() in LocationTracking provides real-time location updates.
    - **Requirements Addressed**:
        - Tracks Personnel's Location in Real-Time
    - **Non-Functional Requirements**: Performance (location updates every 2 seconds), Scalability (handle multiple users)
3. **Test Case 7**: Verify that saveLocation(Location) in LocationDAO stores location data for future tracking and reporting.
    - **Requirements Addressed**:
        - Real-Time Location Tracking and Storage
    - **Non-Functional Requirements**: Data Handling (compliance with data retention laws), Reliability (consistent data storage)
4. **Test Case 8**: Confirm that sendArrivalNotification(userId) in NotificationService sends alerts for deviations or arrival events.
    - **Requirements Addressed**:
        - Sends Alerts for Deviations
    - **Non-Functional Requirements**: Compatibility (works on iOS and Android), Performance (alert within 2 seconds)

Use Case: UC3 - Supervisor Generates Service Report from the System

Requirements in UC3 and Corresponding Test Cases:

1. **Requirement:** Supervisor must be able to navigate to the reporting section.
   - **Mapped Test Case:** *TC01* - Ensures that location permissions are requested on startup, which is a prerequisite for tracking data that will be used in reporting.
2. **Requirement:** System compiles data related to location history and service completion times for a specific period.
   - **Mapped Test Case:** *TC02* - Confirms that the app centers the map on the current location, which ensures location data collection is accurate and can be used for reporting.
   - **Mapped Test Case:** *TC06* - Checks map responsiveness and accuracy to ensure data continuity for location history, critical for compiling reliable reports.
3. **Requirement:** Supervisor reviews and optionally exports the report as a PDF or CSV file.
   - **Mapped Test Case:** *TC07* - Verifies that the supervisor can export reports in both PDF and CSV formats, meeting the requirement for multiple export options.
4. **Requirement:** System should handle cases of insufficient data and notify the supervisor.
   - **Mapped Test Case:** *TC05* - Tests invalid address entry handling, which simulates scenarios of invalid or insufficient data, ensuring that the app provides an appropriate notification in these cases.
5. **Requirement:** System displays error messages on report generation failures and provides options to retry or select another time period.
   - **Mapped Test Case:** *TC05* - Also relevant here, as it verifies that error messages are correctly handled, simulating scenarios in which the system might need to notify the supervisor about unavailable reports.

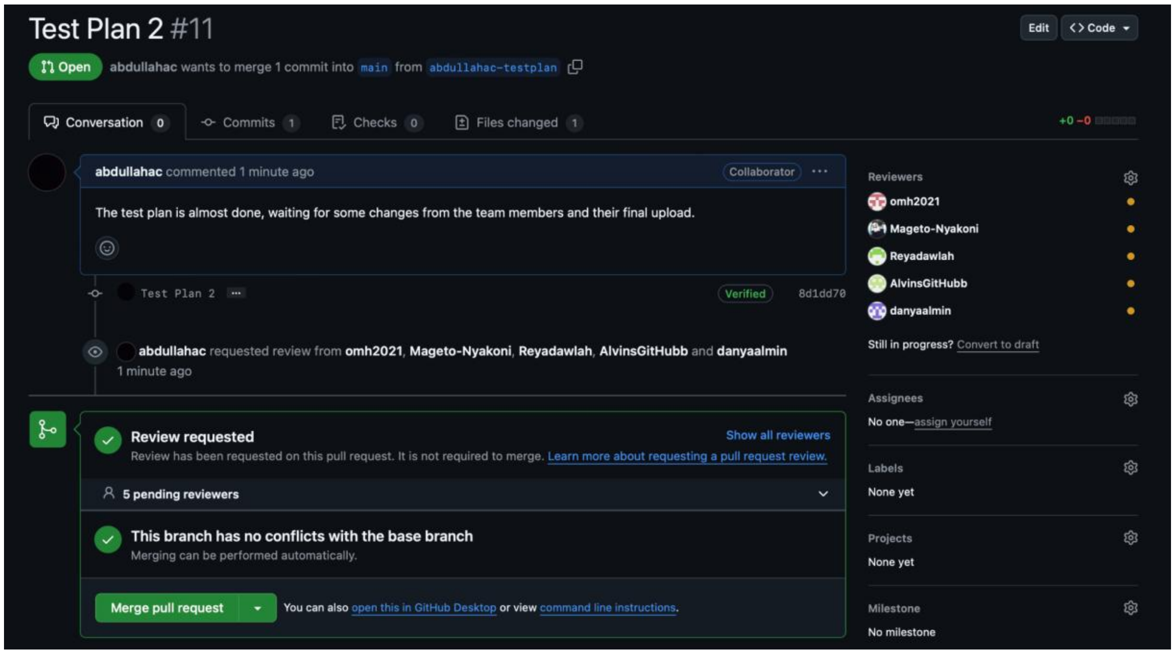
Use Case: UC4 - Service Personnel Downloads Software onto Customer Phone

Requirements in UC4 and Corresponding Test Cases:

1. **Requirement:** Service personnel's arrival is tracked, and the system confirms the arrival location.
   - **Mapped Test Case:** *TC04* - Specifically verifies arrival notifications, ensuring the app notifies when the service personnel reach the designated location. This aligns with the requirement that personnel's arrival location is recorded.
2. **Requirement:** System should detect nearby mobile devices.
   - **Mapped Test Case:** *TC08* - Ensures the system detects and lists nearby devices, enabling connection to the customer’s phone, which is necessary for the download process to begin.
3. **Requirement:** System downloads software onto the customer's phone.
   - **Mapped Test Case:** *TC09* - Verifies the complete software download process, ensuring that the app successfully transfers the software to the customer’s device, assuming necessary storage space and connectivity are available.
4. **Requirement:** Location is continuously tracked during the software download.
   - **Mapped Test Case:** *TC02* and *TC06* - Both of these test cases support continuous tracking. *TC02* confirms the initial location tracking, while *TC06* ensures real-time map updates, which are essential for continuous tracking during software download.
5. **Requirement:** Customers' devices must have enough storage space for the software.
   - **Mapped Test Case:** *TC09* - This test case verifies the software download process, specifically checking that the download only proceeds if there is sufficient storage on the customer’s device, fulfilling this requirement.

EVIDENCE THE TEST PLAN HAS BEEN PLACED UNDER CONFIGURATION MANAGEMENT

- The document was added to github, where each person downloaded it and worked on their parts.
- After everyone was done, they submitted their changes to github and created pull requests.
- https://github.com/AlvinsGitHubb/Location-Verification-App



Test Plan 2 #11

abdullahac wants to merge 1 commit into main from abdullahac-testplan

abdullahac commented 1 minute ago

The test plan is almost done, waiting for some changes from the team members and their final upload.

Test Plan 2 ... Verified 8d1dd70

abdullahac requested review from omh2021, Mageto-Nyakoni, Reyadawlah, AlvinsGitHubb and danyaalmin
1 minute ago

Review requested
Review has been requested on this pull request. It is not required to merge. Learn more about requesting a pull request review.

5 pending reviewers

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Reviewers
omh2021
Mageto-Nyakoni
Reyadawlah
AlvinsGitHubb
danyaalmin

Still in progress? Convert to draft

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

ENGINEERING STANDARDS AND MULTIPLE CONSTRAINTS

- IEEE Std 829-1983: Software Testing [pdf]
- ISO/IEC/IEEE Std 29119-1-(Revision-2022): Part 1 - Software Testing General Concepts [pdf]
- ISO/IEC/IEEE Std 29119-2-(Revision-2021): Part 2 - Test Process [pdf]
- ISO/IEC/IEEE Std 29119-3-(Revision-2021): Part 3 - Test Documentation [pdf]
- ISO/IEC/IEEE Std 29119-4-(Revision-2021): Part 4 - Test Techniques [pdf]

ADDITIONAL REFERENCES

- Jorgensen, P.C., 2013. *Software Testing: A Craftsman's Approach.* Auerbach Publications
- Mathur, A.P., 2013. *Foundations of Software Testing, 2/e.* Pearson Education