7th June 2024

QA for Automotive Feature (Cruise Control System)

Capstone Project - Part A

Name of Testers:

1. Hrushikesh
2. Harshvardhan
3. Anupam
4. Krishna
5. Manish
6. Asha
7. Tahir

Trainer:

Prof. Abhishek Sengupta

Revision Sheet:

Version	Date	Author	Description of Change
1.0	04/06/2024	Hrushikesh	Initial Draft: Purpose, project overview, Test Strategy, Level of Testing
1.1	04/06/2024	Harshvardhan	Revised Draft: Test acceptance criteria, Test Deliverable, TDD
1.2	05/06/2024	Krishna	Revised Draft: Traceability Matrix, Risk Management
1.3	05/06/2024	Anupam	Revised Draft: Autosar Architecture, BOD, UDS, DOIP
1.4	05/06/2024	Manish	Revised Draft: Stress and Load Testing
1.5	06/06/2024	Asha	Revised Draft: Diagnostics of the Defects, Defects Error Code Analysis
1.6	06/06/2024	Tahir	Revised Draft: Diagnostics Error Codes for cruise Control System
1.7	07/06/2024	Group-1	Final Draft: Document Reviewed and Prepared

Table of Contents

- 11.5 Historical Analysis
- 11.6 Automated Error Resolution

13. Diagnostic Error Codes for Cruise Control System

- 12.1 Unit Test Cases
- 12.2 Integration Test Cases
- 12.3 System Test Cases
- 12.4 Regression Test Cases
- 12.5 Performance Test Cases
- 12.6 Compliance Test Cases
- 12.7 Potential Defects and Error Codes

1. Introduction

This document outlines a comprehensive test plan for the cruise control system's lane management functionality. It includes detailed testing strategies, test cases, and the necessary diagnostic analysis to ensure the system operates correctly and efficiently under various conditions. The plan aims to identify and rectify any potential issues, enhancing overall system reliability and performance.

1.1 Purpose

The purpose of this test plan is to outline the testing strategy for the Cruise Control System integrated with a car, ensuring it meets the functional, integration, and performance requirements within the AUTOSAR (Automotive Open System Architecture) layered architecture.

1.2 Project Overview

The Cruise Control System is an advanced feature designed to automatically control the speed of a vehicle. By maintaining a set speed, the CCS enhances driver comfort on long trips and improves fuel efficiency. The system includes the following key components:

Speed Sensors: To detect and monitor the vehicle's speed.

Control Module: The brain of the system that processes inputs and sends commands.

Actuators: To adjust the throttle and maintain the set speed.

User Interface: Allows the driver to set and adjust the desired speed.

The system must be tested thoroughly to ensure that it functions correctly, reliably, and safely under various conditions. This document details the QA approach to be adopted to achieve these goals.

1.3 Audience

This document is intended for the following audiences:

Quality Assurance Team: To understand the QA processes and procedures to be followed.

Development Team: To be aware of the quality standards and testing requirements.

Project Managers: To oversee the QA activities and ensure that the project stays on track.

Stakeholders: To gain insights into the QA efforts and the measures taken to ensure a high-quality product.

Regulatory Bodies: To verify that the system complies with safety and performance regulations.

2. Test Strategy

2.1 Test Objectives

The primary objectives of this test plan are to validate the functional and non-functional requirements of the cruise control system, ensure its integration with other vehicle systems, and confirm its compliance with industry standards. The goal is to deliver a robust and reliable cruise control system that meets user expectations and performs efficiently under different scenarios.

- Validate the functional requirements of the Cruise Control System.

- Ensure seamless integration with other vehicle systems.

- Test the system's performance under various conditions.

- Confirm the system's compliance with AUTOSAR standards.

2.2 Test Assumptions

- The hardware components are pre-tested and functioning correctly.

- The test environment mimics the real-world operating conditions of the car.

- Access to all necessary documentation and specifications.

2.3 Test Principles

- Adherence to AUTOSAR standards.

- Comprehensive coverage of all functional and non-functional requirements.

- Systematic and repeatable testing processes.

2.4 Data Approach

- Use of realistic data sets for cruise control scenarios and diagnostics.

- Test data should cover a range of use cases, including edge cases.

2.5 Scope and Levels of Testing

2.5.1 Unit Testing

The unit tests would focus on ensuring that each part of the system functions correctly in isolation.

Example Unit Tests

1. Speed Sensor Module

Test 1: Verify that the speed sensor correctly measures and reports the vehicle speed.

Test 2: Check the response of the speed sensor to various speed inputs (e.g., 0 km/h, 50 km/h, 100 km/h).

2. Control Algorithm

Test 1: Ensure that the control algorithm correctly maintains the set speed within a specified tolerance.

Test 2: Test the algorithm's response to sudden changes in vehicle speed (e.g., sudden acceleration or deceleration).

3. User Interface Module

Test 1: Validate that setting the cruise control to a specific speed via the user interface correctly stores the speed.

Test 2: Ensure that increasing and decreasing the set speed works as intended.

Test 3: Verify that the cruise control can be deactivated using the user interface.

Unit Testing Best Practices:

Isolation: Each unit test should test a single component or function in isolation to avoid dependencies on other parts of the system.

Automated Testing: Use automated testing frameworks to run unit tests frequently and consistently.

Mocking and Stubbing: Use mocks and stubs to simulate the behavior of complex dependencies.

Coverage: Aim for high code coverage to ensure that most, if not all, code paths are tested.

Repeatability: Ensure tests can be run repeatedly with the same results, providing a stable test environment.

Application Layer:

- **Set Speed Function:**

  - Test Case: Verify that the set speed function correctly maintains the desired speed.

  - Expected Result: The vehicle maintains the set speed accurately without fluctuations.

  - Potential Defect: The vehicle speed fluctuates instead of maintaining a constant speed.

  - Error Code: P0500 (Vehicle Speed Sensor Malfunction).

- User Interface Responses:

  - Test Case: Validate user interface responses to inputs.

  - Expected Result: The user interface responds correctly and promptly to all inputs.

  - Potential Defect: User inputs are not recognized or delayed.

  - Error Code: B1503 (Touch Screen Circuit Malfunction).

Runtime Environment (RTE):

- Service Call Execution:

  - Test Case: Validate that service calls from the Cruise Control application to other software components execute correctly.

- Expected Result: All service calls execute without errors and within the required time frame.

- Potential Defect: Service call delays leading to delayed throttle response.

- Error Code: N/A (Performance issue without specific DTC).

Basic Software (BSW) Modules:

- Communication Service Validation:

- Test Case: Test the data exchange between the Cruise Control system and other ECUs via the CAN network.

- Expected Result: Data packets are transmitted and received correctly without loss.

- Potential Defect: Data packets are lost or corrupted during transmission.

- Error Code: U0100 (Lost Communication with ECM/PCM).

- Memory Services Validation:

- Test Case: Validate memory services (NvM, MemStack).

- Expected Result: Memory services perform correctly, ensuring data integrity and availability.

- Potential Defect: Memory data is lost or corrupted.

- Error Code: N/A (Memory issue without specific DTC).

- Diagnostic Services Validation:

- Test Case: Check diagnostic services (DEM, DCM).

- Expected Result: Diagnostic services correctly identify and report faults.

- Potential Defect: Diagnostic errors are not reported or are inaccurately reported.

- Error Code: Varies based on the specific diagnostic issue.

Microcontroller Abstraction Layer (MCAL):

- Peripheral Drivers Validation:

- Test Case: Verify peripheral drivers (CAN, Ethernet).

- Expected Result: Peripheral drivers function correctly, allowing proper communication.

- Potential Defect: Peripheral drivers fail, causing communication issues.

- Error Code: U0100 (Lost Communication with ECM/PCM).

2.5.2 Integration Testing

Integration tests for the cruise control system will focus on ensuring that the interactions and data flow between these components are functioning correctly.

Example Integration Tests

1. Speed Sensor Module and Control Algorithm

Test 1: Verify that the speed sensor module correctly transmits speed data to the control algorithm.

Test 2: Check that the control algorithm correctly adjusts its calculations based on the speed data received from the speed sensor.

2. User Interface Module and Control Algorithm

Test 1: Verify that commands from the user interface (e.g., setting speed, adjusting speed, deactivating cruise control) are correctly received and processed by the control algorithm.

Test 2: Ensure that the control algorithm updates the user interface with current speed and set speed information accurately.

3. Full System Integration

Test 1: Validate the entire workflow from setting the cruise control speed through the user interface to maintaining that speed via the control algorithm and throttle control module, and disengaging the cruise control when the brake is applied.

Test 2: Test the system's behavior during various real-world driving scenarios, such as sudden deceleration, gradual speed changes, and continuous speed maintenance.

Integration Testing Best Practices

Incremental Integration: Combine units and test incrementally (e.g., pair-wise testing, then gradually integrating more units) to isolate issues effectively.

Continuous Integration: Use continuous integration tools to automate the process of integrating and testing changes frequently.

Comprehensive Test Coverage: Ensure that tests cover all possible interactions between units, including edge cases and error conditions.

Clear Interface Contracts: Define clear contracts for interfaces between modules to ensure consistent and expected communication.

Mocking and Stubbing: Use mocks and stubs for external systems or modules that are not yet integrated or are difficult to test.

Communication Interfaces:

- CAN and Ethernet Communication:

- Test Case: Validate CAN and Ethernet communication between cruise control components.

- Expected Result: Communication is seamless, with no data loss or corruption.

- Potential Defect: Data loss or corruption during communication.

- Error Code: U0100 (Lost Communication with ECM/PCM).

- Head Unit Data Exchange:

- Test Case: Test data exchange between head unit and external devices (e.g., diagnostic tools).

- Expected Result: Data exchange is accurate and timely.

- Potential Defect: Incorrect or delayed data exchange.

- Error Code: N/A (Communication issue without specific DTC).

Sensor Integration:

- Speed Sensors Functionality:

- Test Case: Verify functionality of speed sensors and radar/lidar for adaptive cruise control.

- Expected Result: Sensors function correctly, providing accurate data.

- Potential Defect: Sensor data is inaccurate or delayed.

- Error Code: P0500 (Vehicle Speed Sensor Malfunction).

- Braking and Steering Systems Integration:

- Test Case: Check integration with vehicle's braking and steering systems.

- Expected Result: Integration is seamless, with no delays or errors.

- Potential Defect: Delays or errors in braking and steering responses.

- Error Code: P0571 (Brake Switch "A" Circuit Malfunction).

User Interface Integration:

- Display and Controls:

- Test Case: Verify display and controls for cruise control settings and alerts.

- Expected Result: Displays and controls are accurate and responsive.

- Potential Defect: Displays and controls are inaccurate or unresponsive.

- Error Code: B1503 (Touch Screen Circuit Malfunction).

- User Controls Responsiveness:

- Test Case: Check responsiveness and accuracy of user controls.

- Expected Result: User controls respond accurately and without delay.

- Potential Defect: User controls are slow or inaccurate.

- Error Code: N/A (User control issue without specific DTC).

2.5.3 System Testing

system testing involves verifying the entire cruise control functionality in a real or simulated environment to ensure it behaves as expected under various conditions.

System testing for a cruise control system will focus on the following:

Functional Requirements: Verifying that the cruise control system performs all specified functions correctly.

Performance Requirements: Ensuring the system operates within performance constraints such as response time and speed maintenance accuracy.

Safety Requirements: Confirming that the system adheres to safety standards, especially regarding disengagement during braking and emergency conditions.

User Experience: Testing the user interface for ease of use and clarity.

System Testing Best Practices:

Comprehensive Test Plans: Develop detailed test plans that cover all functional and non-functional requirements.

Realistic Test Environment: Perform tests in an environment that closely simulates real-world driving conditions.

Automated Testing: Use automated testing tools to conduct repetitive tests and ensure consistency.

Cross-Functional Collaboration: Involve various stakeholders (e.g., developers, testers, safety engineers) to ensure comprehensive coverage and understanding.

Regression Testing: Regularly perform regression testing to ensure new changes do not introduce new issues.

System Boot and Shutdown:

- Startup Sequence:

- Test Case: Verify system start-up and shutdown sequences.

- Expected Result: The system starts up and shuts down without errors.

- Potential Defect: Errors during startup or shutdown.

- Error Code: P0650 (Malfunction Indicator Lamp (MIL) Control Circuit).

Functional Scenarios:

- Maintaining Set Speed:

- Test Case: Test common use cases (e.g., maintaining set speed, adaptive speed regulation).

- Expected Result: The vehicle maintains the set speed accurately.

- Potential Defect: Vehicle fails to maintain the set speed.

- Error Code: P0500 (Vehicle Speed Sensor Malfunction).

- Adaptive Speed Regulation:

- Test Case: Validate adaptive speed regulation in response to traffic conditions.

- Expected Result: Adaptive speed regulation functions correctly, adjusting speed as needed.

- Potential Defect: Adaptive speed regulation fails or responds inappropriately.

- Error Code: N/A (Adaptive speed regulation issue without specific DTC).

Stress and Load Testing:

- High Load Conditions:

- Test Case: Assess system performance under high load conditions.

- Expected Result: System performs correctly under high load.

- Potential Defect: System performance degrades under high load.

- Error Code: N/A (Performance issue without specific DTC).

Failover and Recovery:

- Component Failures:

- Test Case: Test system response to component failures.

- Expected Result: System responds appropriately to failures and recovers correctly.

- Potential Defect: System fails to recover or respond correctly.

- Error Code: Varies based on specific component failure.

2.5.4 User Acceptance Testing (UAT)

- User Feedback:

- Test Case: Conduct UAT with real users to ensure the system meets their needs and expectations.

- Expected Result: Users find the system meets their needs and expectations.

- Potential Defect: User feedback indicates issues or unmet needs.

- Error Code: N/A (User feedback issue without specific DTC).

- Adjustments Based on Feedback:

- Test Case: Collect feedback and make necessary adjustments.

- Expected Result: System is adjusted to meet user feedback.

- Potential Defect: Adjustments do not meet user expectations.

- Error Code: N/A (User feedback adjustment issue without specific DTC).

2.5.5 Performance Testing

Response Times:

- User Inputs and System Commands:

- Test Case: Measure response times for user inputs and system commands.

- Expected Result: Response times are within acceptable limits.

- Potential Defect: Delayed response times.

- Error Code: N/A (Performance issue without specific DTC).

Data Transfer Rates:

- Communication Latency:

- Test Case: Assess data transfer rates and communication latency.

- Expected Result: Data transfer rates and latency are within acceptable limits.

- Potential Defect: High latency or low data transfer rates.

- Error Code: N/A (Communication issue without specific DTC).

Resource Usage:

- CPU, Memory, and Network:

- Test Case: Monitor CPU, memory, and network usage.

- Expected Result: Resource usage is within acceptable limits.

- Potential Defect: High resource usage leading to performance issues.

- Error Code: N/A (Resource usage issue without specific DTC).

Varying Resource Availability:

- System Performance:

- Test Case: Validate system performance under varying resource availability.

- Expected Result: System performs correctly under different resource availability conditions.

- Potential Defect: System performance degrades with varying resource availability.

- Error Code: N/A (Performance issue without specific DTC).

2.5.6 Security Testing

Security Testing is a type of software testing that aims to uncover vulnerabilities, threats, and risks in a software application. The goal is to ensure that the software is protected from malicious attacks and unauthorized access, ensuring the integrity, confidentiality, and availability of the system.

Security testing for a cruise control system will focus on the following aspects:

Authentication: Ensuring only authorized users can access and control the system.

Authorization: Ensuring users have appropriate permissions for actions they can perform.

Data Integrity: Protecting the data from being altered or tampered with.

Data Confidentiality: Ensuring sensitive data is protected from unauthorized access.

Non-repudiation: Ensuring that actions taken cannot be denied. Resilience to Attacks: Protecting the system from various types of cyber-attacks, such as spoofing, tampering, information disclosure, and denial of service.

Example Security Tests

1. Authentication and Authorization

Test 1: Verify that only authenticated users can access and control the cruise control system.

Test 2: Ensure that users with different roles (e.g., driver, maintenance) have appropriate permissions and restrictions.

2. Data Integrity

Test 1: Test the system's ability to detect and prevent unauthorized changes to critical data (e.g., speed settings, user commands).

Test 2: Verify the integrity of data transmission between modules (e.g., speed sensor to control algorithm).

3. Data Confidentiality

Test 1: Ensure that sensitive data (e.g., speed settings, user credentials) is encrypted during storage and transmission.

Test 2: Test for vulnerabilities that could lead to data leaks or exposure (e.g., through communication channels).

Security Testing Best Practices:

Threat Modeling: Identify potential threats and vulnerabilities by analyzing the system's design and architecture.

Regular Security Audits: Conduct regular security audits and code reviews to identify and fix vulnerabilities.

Use of Security Tools: Utilize automated tools for vulnerability scanning, penetration testing, and static code analysis.

Security Training: Ensure that developers and testers are trained in secure coding practices and security testing techniques.

External Threats:

- System Security:
  - Test Case: Ensure the lane management system is secure from external threats.
  - Expected Result: System is secure with no vulnerabilities.
  - Potential Defect: System vulnerabilities are detected.
  - Error Code: N/A (Security issue without specific DTC).

Vulnerability Testing:

- Potential Breaches:
  - Test Case: Test for vulnerabilities and potential breaches.
  - Expected Result: No vulnerabilities or potential breaches are detected.
  - Potential Defect: Vulnerabilities or potential breaches are detected.
  - Error Code: N/A (Security issue without specific DTC).

2.5.7 Regression Testing

Regression testing for a cruise control system will focus on re-validating the system's core functionalities and ensuring that new changes have not introduced new bugs or issues.

Example Regression Tests

1. Core Functionalities

Test 1: Verify that the user can still set the desired speed using the user interface and that the system maintains this speed accurately.

Test 2: Ensure that the user can still increase and decrease the set speed and that the system adjusts accordingly.

2. System Response to Inputs

Test 1: Verify that the system responds correctly to user commands through the user interface, such as setting, adjusting, and deactivating the cruise control.

Test 2: Ensure the system disengages the cruise control promptly when the brake is applied.

3. Edge Cases and Stress Conditions

Test 1: Re-run tests for edge cases to ensure that the system handles unusual or extreme inputs correctly.

Test 2: Re-test the system's performance under stress conditions, such as rapid acceleration and deceleration.

Regression Testing Best Practices:

Automated Testing: Use automated testing tools to run regression tests frequently and efficiently. This ensures comprehensive coverage and quick feedback on recent changes.

Comprehensive Test Suite: Maintain a comprehensive suite of regression test cases that cover all major functionalities, interactions, and edge cases.

Continuous Integration: Integrate regression testing into the continuous integration (CI) pipeline to automatically run tests whenever changes are made to the codebase.

Prioritize Tests: Prioritize regression test cases based on critical functionalities and recent areas of change to focus on the most impacted parts of the system.

Regular Updates: Regularly update the regression test suite to include new test cases for new features and functionalities.

Previous Test Cases:

- Re-run Test Cases:

  - Test Case: Re-run previous test cases to ensure new updates do not break existing functionalities.

  - Expected Result: All previously passed test cases should pass again.

  - Potential Defect: Previously fixed issues reappear.

  - Error Code: Varies based on specific defect reoccurrence.

Bug Fixes and New Features:

- Validation:

  - Test Case: Validate bug fixes and new features.

  - Expected Result: Bug fixes and new features work correctly without affecting existing functionalities.

  - Potential Defect: New features or bug fixes cause issues in existing functionalities.

  - Error Code: Varies based on specific new feature or bug fix issue.

2.6 Test Effort Estimate

- Exploratory Testing: 5 days

- Functional Testing: 15 days

- Integration Testing: 10 days

- System Testing: 12 days

- UAT: 8 days

- Performance Testing: 5 days

- Security Testing: 5 days

- Regression Testing: 8 days

3. Test Acceptance Criteria

- All high and medium severity defects are resolved.

- Functional test cases achieve a pass rate of 95%.

- Integration and system test cases achieve a pass rate of 98%.

- Performance benchmarks for response time, speed regulation accuracy, and user interface responsiveness are met.

- Security tests show no vulnerabilities.

4. Test Deliverables

- Test Plan Document

- Test Cases and Test Scripts

- Test Execution Reports

- Defect Logs and Resolution Reports

- UAT Feedback Reports

- Final Test Summary Report

5. Test Driven Development (TDD)

5.1 What is Test Driven Development

Test-driven development (TDD) is a method of coding in which you first write a test and it fails, then write the code to pass the test of development, and clean up the code. This process recycled for one new feature or change. In other methods in which you write either all the code or all the tests first, TDD will combine and write tests and code together into one.

5.2 History of Test-Driven Development

TDD shares similarities with test-first programming from extreme programming, which started in 1999. However, TDD has gained more widespread interest on its own. Programmers also use TDD to improve and fix old code written with different methods.

The idea of Test-Driven Development (TDD) which invented from an old book on programming. In this suggested method you will manually enter the expected output and then write a code until the actual output when matches it. After creating the first xUnit framework, we will remember this and give it a try which is related to the the invention of the TDD for me.

5.3 Test-Driven work in Test-Driven Development

TDD, or Test-Driven Development, is not just for software only. It is also used to create product and service teams as test-driven work. To make testing successful, it needs to be created at both small and big levels in test-driven development. This means testing every part of the work, like methods in a class, input data values, log messages, and error codes. Other side of software, teams use quality control (QC) will check before starting work. These will be checks to help plan and check the outcomes of the tests. They follow a similar process to TDD, with some small changes which are as follows:

- “Add a check” instead of “Add a test”
- “Run all checks” instead of “Run all tests”
- “Do the work” instead of “Write some code”
- “Run all checks” instead of “Run tests”
- “Clean up the work” instead of “Refactor code”
- Repeat these steps

5.4 Process of Test-Driven Development

Process of Test-Driven Development (TDD)

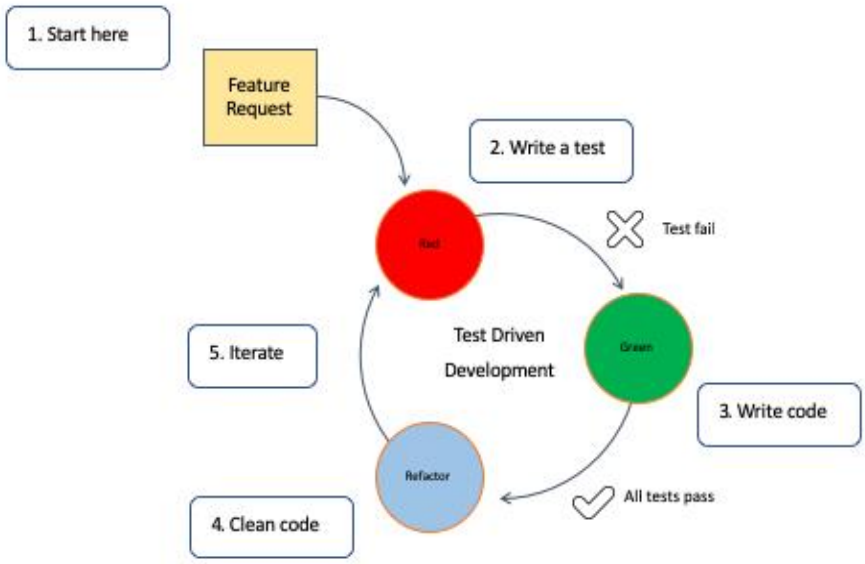
It is the process in which test cases are written before the code that validates those cases. It depends on the repetition of a concise development cycle. Test-driven Development is a technique in which automated Unit tests are used to drive the design and free decoupling of dependencies. The following sequence of steps is generally followed:

Write a complete test case describing the function. To make the test cases the developer must understand the features and requirements using user stories and use cases.

- Read, understand, and process the feature or bug request.
- Translate the requirement by writing a unit test. If you have hot reloading set up, the unit test will run and fail as no code is implemented yet.

- Write and implement the code that fulfills the requirement. Run all tests and they should pass, if not repeat this step.
- Clean up your code by refactoring.
- Rinse, lather and repeat.

The key principles of TDD include writing tests before code, writing only enough code to pass the tests, and continuously refactoring to improve code quality. By following these principles and the iterative process outlined above, developers can create well-tested, maintainable code that meets the needs of the stakeholders.



This workflow is sometimes called Red-Green-Refactoring, which comes from the status of the tests within the cycle.

- The red phase indicates that code does not work.
- The green phase indicates that everything is working, but not necessary in the most optimal way.
- The blue phase indicates that the tester is refactoring the code, but is confident their code is covered with tests which gives the tester confidence to change and improve our code.

5.5 TDD VS Traditional Testing

5.5.1 Approach:

Test Driven Development (TDD) it is a way of making software in that tests are written first after that the code is written. In traditional testing it the other way, It will making the code first and then start testing in it.

5.5.2 Testing Scope: TDD checks small parts of code one by one. Traditional testing checks the whole system, including how different parts work together.

5.5.3 Iterative: TDD is works in small steps. It will write a small code and tests, and then improve regularly to code until it passes all the tests which are required. Traditional testing tests the code one time and then fixing any problems which ate been find.

5.5.4 Debugging: TDD will try to find mistakes early in the process of code, which makes it will easier to fix them. Traditional testing will find the mistakes for after, which can be when held then it will harder to fix in the future.

5.5.5 Documentation: TDD will focuses on documentation of the tests and their results. Traditional testing might have been more clear information about how the testing made done and the system will have tested.

5.6 Advantages and Disadvantages of TDD

Advantages of Test-Driven Development (TDD)

- Unit test provides constant feedback about the functions.
- Quality of design increases which further helps in proper maintenance.
- Test driven development act as a safety net against the bugs.
- TDD ensures that your application actually meets requirements defined for it.
- TDD have very short development lifecycle.

Disadvantages of Test-Driven Development (TDD)

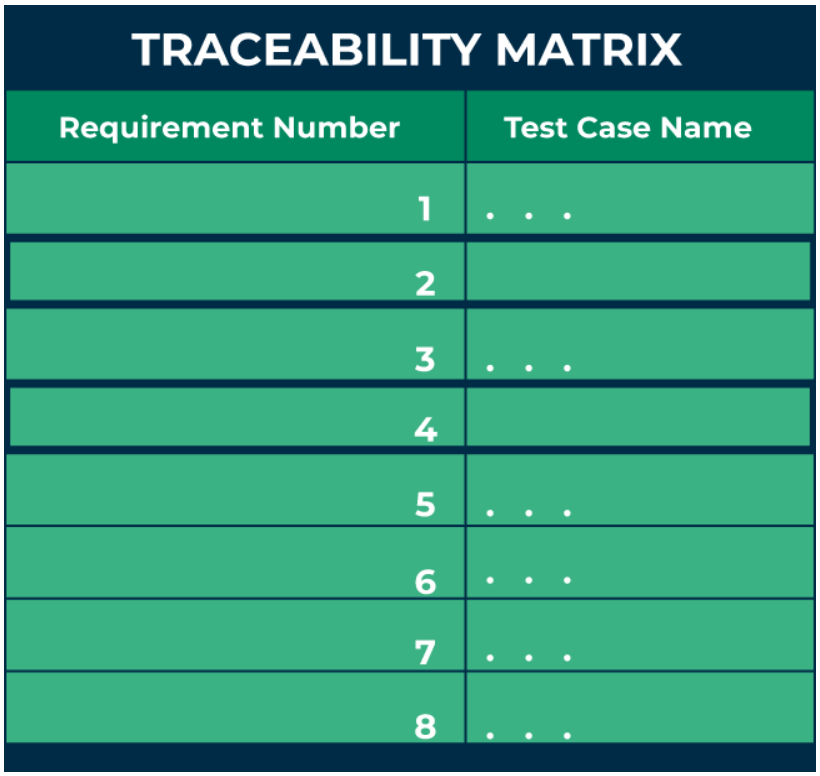
- Increased Code Volume: Using TDD means writing extra code for tests cases, which can make the overall codebase larger and more Unstructured.
- False Security from Tests: Passing tests will make the developers think the code is safer only for assuming purpose.
- Maintenance Overheads: Keeping a lot of tests up-to-date can be difficult to maintain the information and it's also time-consuming process.
- Time-Consuming Test Processes: Writing and maintaining the tests can take a long time.
- Testing Environment Set-Up: TDD needs to be a proper testing environment in which it will make effort to set up and maintain the codes and data.

6. Requirement Traceability Matrix

6.1 What is Requirement Traceability Matrix (RTM)?

RTM stands for Requirement Traceability matrix. RTM maps all the requirements with the test cases. By using this document one can verify test cases cover all functionality of the application as per the requirements of the customer.

- Requirements: Requirements of a particular project from the client.
- Traceability: The ability to trace the tests.
- Matrix: The data which can be stored in rows and columns form.



The main purpose of the requirement traceability matrix is to verify that the all requirements of clients are covered in the test cases designed by the testers.

In simple words, one can say it is a pen and pencil approach i.e., to analyze the two data information but here we are using an Excel sheet to verify the data in a requirement traceability matrix.

6.2 Why is Requirement Traceability Matrix (RTM) Important?

When business analysis people get the requirements from clients, they prepare a document called SRS (System/Software Requirement Specification) and these requirements are stored in this document. If we are working in the Agile model, we call this document Sprint Backlog, and requirements are present in it in the form of user stories.

When QA gets the SRS/Sprint backlog document they first try to understand the requirements thoroughly and then start writing test cases and reviewing them with the entire project team. But sometimes it may happen that in these test cases, some functionality of requirements is missing, so to avoid it we required a requirement traceability matrix

- Each test case is traced back to each requirement in the RTM. Therefore, there is less chance of missing any requirement in testing, and 100% test coverage can be achieved.
- RTM helps users discover any change that was made to the requirements as well as the origin of the requirement.
- Using RTM, requirements can be traced to determine a particular group or person that wanted that requirement, and it can be used to prioritize the requirement.
- It helps to keep a check between requirements and other development artifacts like technical and other requirements.
- The Traceability matrix can help the tester identify whether by adding any requirement previous requirements are affected or not.
- RTM helps in evaluating the effect on the QA team to reuse the test case.

6.3 Parameters of Requirement Traceability Matrix (RTM)

The below figure shows the basic template of RTM. Here the requirement IDs are row-wise and test case IDs are column-wise which means it is a forward traceability matrix.

The following are the parameters to be included in RTM:

- Requirement ID: The requirement ID is assigned to every requirement of the project.
- Requirement description: for every requirement a detailed description is given in the SRS (System/Software Requirement Specification) document.
- Requirement Type: understand the type of requirements i.e., banking, telecom, healthcare, traveling, e-commerce, education, etc.
- Test cases ID: the testing team designs test cases. Test cases are also assigned with some ID.

6.4 Types of Traceability Matrix

6.4.1 Forward traceability matrix

In the forward traceability matrix, we mapped the requirements with the test cases. Here we can verify that all requirements are covered in test cases and no functionality is missing in test cases. It helps you to ensure that all the requirements available in the SRS/ Sprint backlog can be traced back to test cases designed by the testers. It is used to check whether the project progresses in the right direction.

In forwarding the traceability matrix:

Rows = Requirement ID

Column = Test case ID

6.4.2 Backward traceability matrix

In the backward traceability matrix, we mapped the test cases with the requirements. Here we can verify that no extra test case is added which is not required as per our requirements. It helps you to ensure that any test cases that you have designed can be traced back to the requirements or user stories, and you are not extending the scope of the work by just creating additional test cases that cannot be mapped to the requirement. The backward traceability matrix is also known as the reverse traceability matrix.

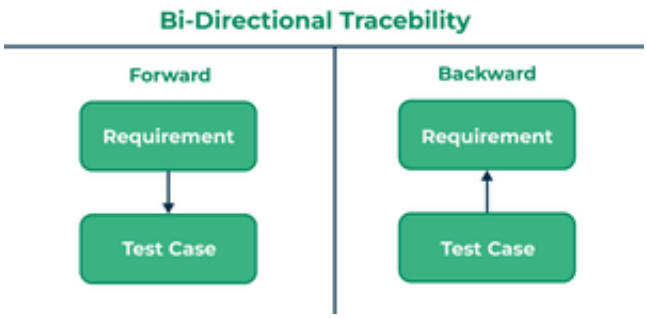
In the backward traceability matrix:

Rows = Test cases ID

Column = Requirement ID

6.4.3 Bi-directional traceability matrix

A bi-directional traceability matrix is a tool used in software development and project management to ensure that all requirements are linked to corresponding test cases and vice versa. It provides a systematic approach to trace the relationship between requirements, test cases, and other project artifacts. A bi-directional traceability matrix is a combination of a forward traceability matrix and a backward traceability matrix. Here we verify the requirements and test cases in both ways.



Bi-directional traceability matrix = Forward traceability matrix + Backward traceability matrix.

6.5 Traceability matrix for a QA (Quality Assurance) process for Cruise Control System

Creating a traceability matrix for a QA (Quality Assurance) process for an automotive feature like the Cruise Control System involves mapping various requirements, test cases, and other elements to ensure comprehensive coverage. Here's a simplified template for such a matrix:

A traceability matrix for the Quality Assurance (QA) process of an automotive feature such as the Cruise Control System is a vital tool for ensuring comprehensive coverage and adherence to requirements throughout the development lifecycle.

This matrix serves as a structured document that links various elements such as requirements, test cases, and other project artifacts. Each requirement is mapped to corresponding test cases, allowing for easy verification that all specified functionalities are adequately tested.

For instance, a requirement might specify that the Cruise Control System should maintain the vehicle's speed within +/- 1 mph of the set speed. In the traceability matrix, this requirement would be associated with one or more test cases designed to verify this functionality.

Conversely, each test case is linked back to the specific requirement it addresses. This bidirectional linkage ensures that every aspect of the system's behavior is accounted for and tested thoroughly.

Throughout the QA process, the traceability matrix serves as a reference point for testers, developers, and stakeholders. It facilitates efficient impact analysis, change management, and compliance verification. Additionally, it helps in identifying any gaps in test coverage or inconsistencies between requirements and test cases, enabling teams to address them promptly.

In essence, the traceability matrix plays a crucial role in maintaining the integrity and reliability of automotive features like the Cruise Control System, ultimately contributing to the delivery of high-quality software that meets user expectations and regulatory standards.

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Test Result
REQ-001	Cruise control shall maintain vehicle speed within +/- 1 mph of the set speed	TC-001	Verify that cruise control maintains speed within tolerance limits	Pass/Fail
REQ-002	Cruise control shall disengage when the brake pedal is pressed	TC-002	Test cruise control disengagement upon brake pedal activation	Pass/Fail
REQ-003	Cruise control shall resume the previously set speed after disengagement	TC-003	Test cruise control resumption after disengagement	Pass/Fail
REQ-004	Cruise control shall engage at speeds above 25 mph	TC-004	Verify cruise control engagement at speeds above threshold	Pass/Fail
REQ-005	Cruise control shall have a cancel button to deactivate the system	TC-005	Test cancel button functionality	Pass/Fail
REQ-006	Cruise control shall not engage if the vehicle is in a gear other than drive or neutral	TC-006	Verify cruise control engagement in valid gears	Pass/Fail
REQ-007	Cruise control shall not engage if the engine is not running	TC-007	Test cruise control behavior when engine is off	Pass/Fail

This matrix helps in tracking the fulfillment of each requirement through corresponding test cases and their results. Additionally, it ensures that all requirements are adequately tested, helping to validate the functionality and robustness of the Cruise Control System.

7. Risk Management

Identify Potential Risks

- Hardware failures
- Software bugs
- Communication breakdowns

Mitigation Strategies

- Implement robust error-handling and recovery mechanisms.
- Conduct thorough pre-testing of individual components.
- Regularly update and review test plans based on feedback and findings.
- Allocate additional time for unforeseen issues during test phases.

8. AUTOSAR Layered Architecture Explanation

AUTOSAR (Automotive Open System Architecture) is a standardized automotive software architecture that provides a framework for developing and integrating complex automotive software systems. It divides the software into several layers to simplify development, enhance modularity, and ensure interoperability.

Application Layer

- Contains the functional software components that implement specific application functionalities, such as cruise control.

Runtime Environment (RTE)

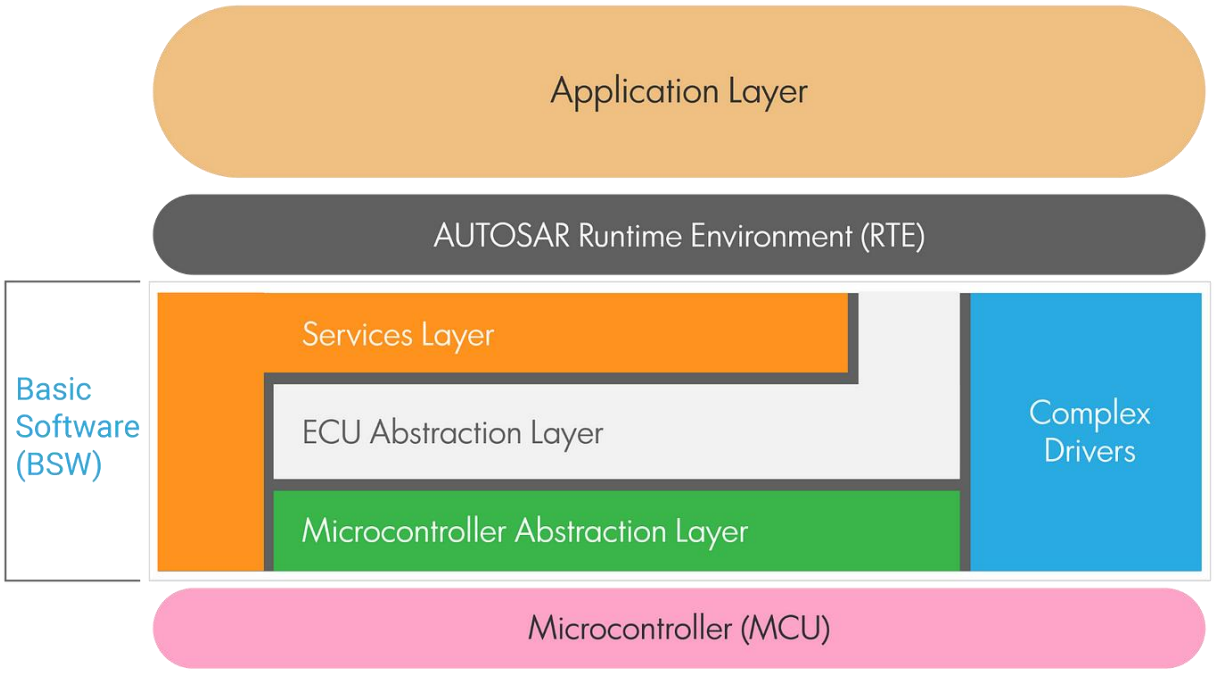
- Acts as a middleware layer, providing communication services between software components and between software components and the Basic Software (BSW) layer.

Basic Software (BSW)

- Includes standardized software modules that provide services for communication, memory management, diagnostic services, and operating system functionality.

Microcontroller Abstraction Layer (MCAL)

- Provides a hardware abstraction layer that allows the higher layers to be hardware-independent. It includes drivers for peripherals like CAN, Ethernet, and other communication interfaces.



AUTOSAR Architecture

1. Driver Input

Test Cases: Verify that all user inputs (e.g., set speed, set distance) are correctly captured and processed.

Testing Methods: automated input simulation.

2. Application Layer

User Interface Component:

Test Cases: Ensure accurate display and processing of user inputs.

Testing Methods: automated UI testing.

Speed Control Component:

Test Cases: Validate speed adjustment logic under various scenarios.

Testing Methods: Unit testing, integration testing, and hardware-in-the-loop (HIL) testing.

Distance Control Component:

Test Cases: Verify safe distance maintenance in different traffic conditions.

Testing Methods: Simulation and HIL testing.

3. Runtime Environment (RTE)

Test Cases: Check the correct routing of data and commands between components.

Testing Methods: Integration testing and end-to-end testing.

4. Basic Software Layer (BSW)

Services Layer:

Communication Service:

Test Cases: Verify reliable data exchange between ECUs.

Testing Methods: Network simulation and CAN/LIN bus

Diagnostic Service:

Test Cases: Ensure accurate fault detection and reporting.

Testing Methods: Fault injection testing.

Memory Service:

Test Cases: Validate proper data storage and retrieval.

Testing Methods: Stress testing

ECU Abstraction Layer:

Radar Sensor Interface:

Test Cases: Check accurate data retrieval from radar Sensors

Testing Methods: Sensor simulation.

Actuator Interface:

Test Cases: Verify correct actuator control signals.

Testing Methods: HIL testing.

5. Microcontroller Abstraction Layer (MCAL):

ADC Driver:

Test Cases: Ensure accurate reading of sensor data.

Testing Methods: Unit testing.

PWM Driver:

Test Cases: Validate proper control of actuators.

Testing Methods: Integration testing.

Timer Driver:

Test Cases: Check timing accuracy for control operations.

Testing Methods: Unit testing.

9. BOD, UDS, and DoIP Explanation

BOD (Base Operating Diagnostic)

BOD refers to the basic set of diagnostic functions provided by an automotive system. For a cruise control system, BOD encompasses the fundamental diagnostic operations required to monitor and maintain the system's performance.

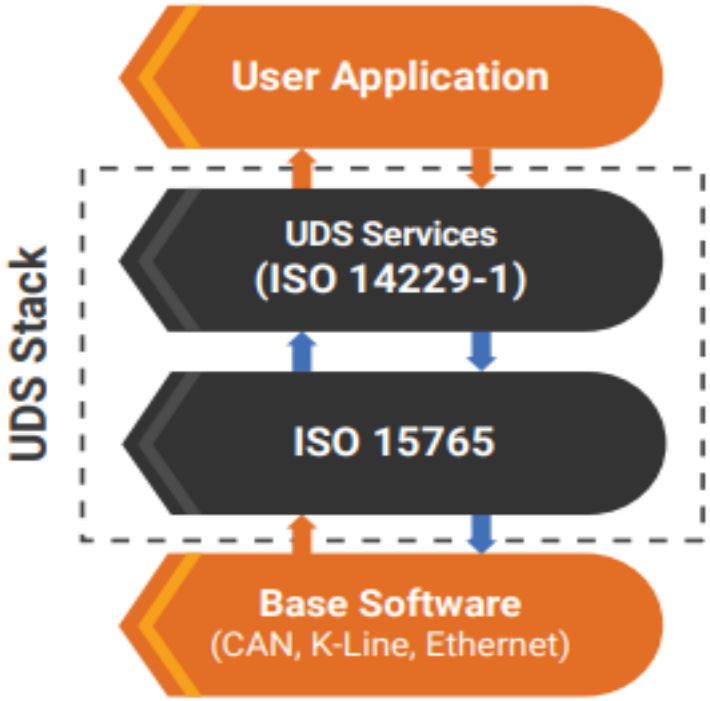
This includes:

Basic Health Monitoring: Continuous checks on the cruise control system's components, such as sensors, actuators, and control units, to ensure they are functioning correctly.

Error Detection: Identifying and logging basic faults or malfunctions within the system, providing essential diagnostic trouble codes (DTCs).

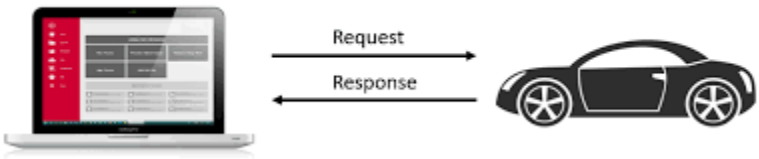
Status Reporting: Offering basic status reports and operational data about the cruise control system to aid in troubleshooting and maintenance.

Diagram:



UDS (Unified Diagnostic Services)

UDS is a comprehensive protocol used for diagnostic communication within a vehicle, allowing for in-depth diagnostics and service procedures. In the context of a cruise control system

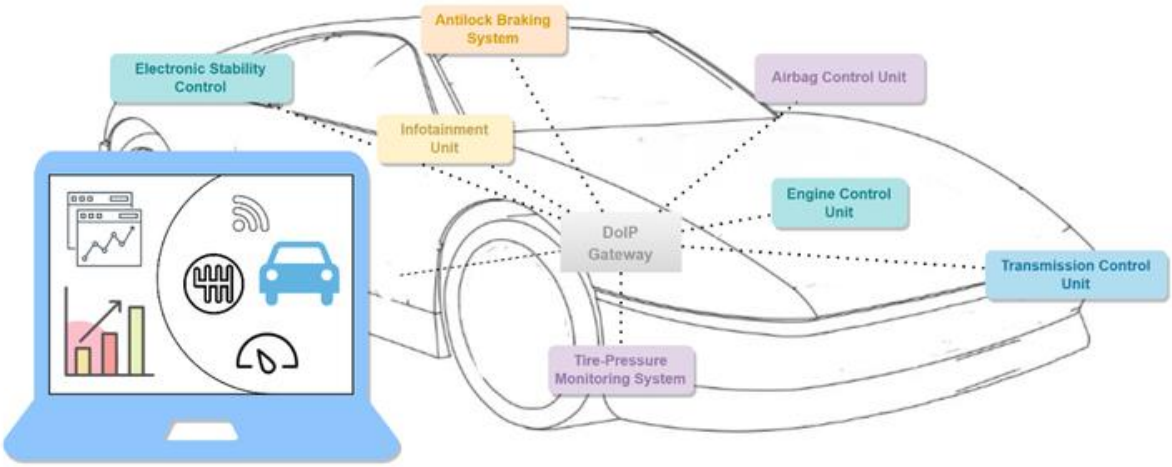


UDS supports:

Advanced Diagnostic Functions: Reading detailed sensor data, performing actuator tests, and retrieving specific DTCs related to the cruise control system.

Control and Configuration: Enabling technicians to perform actions like resetting components, updating software, or configuring parameters within the cruise control system.

Service Functions: Providing a wide range of services, such as firmware updates, system resets, and configuration adjustments to maintain optimal performance of the cruise control system



DoIP (Diagnostics over Internet Protocol)

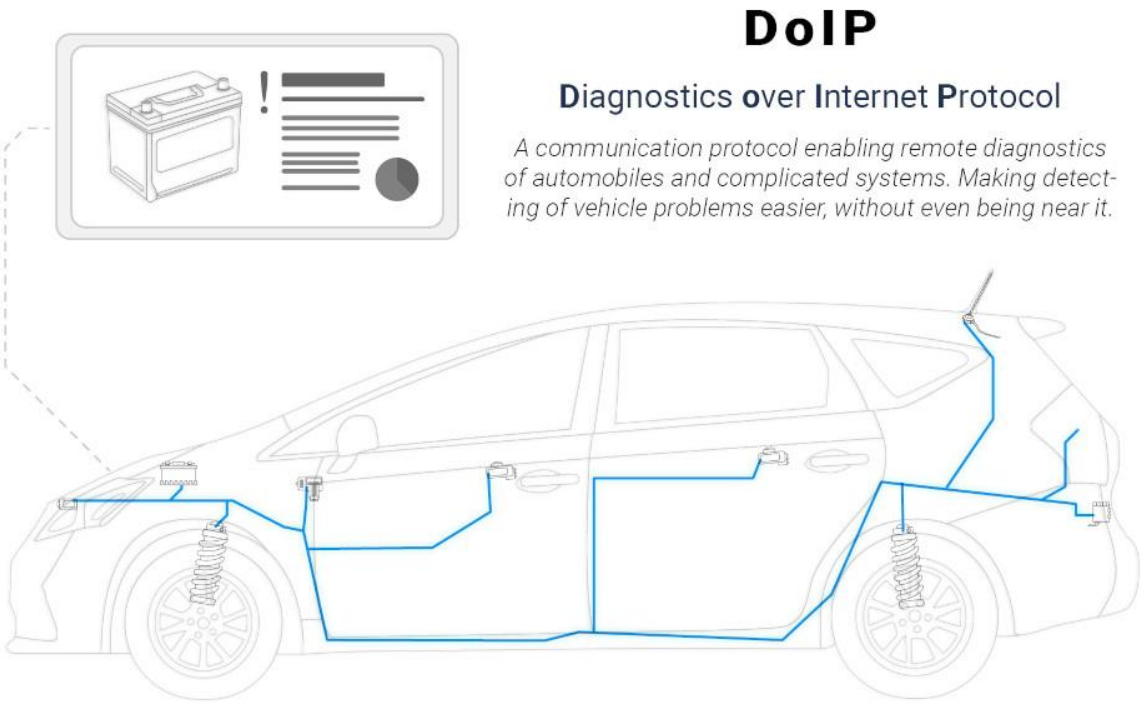
DoIP is an extension of the UDS protocol that facilitates diagnostic communication over IP networks. For a cruise control system,

DoIP offers significant advantages:

Remote Diagnostics: Allowing technicians to perform diagnostics and updates from a remote location, improving convenience and reducing the need for physical access to the vehicle.

High-Speed Communication: Leveraging the capabilities of IP networks to transfer diagnostic data quickly and efficiently, enabling more responsive and detailed diagnostic procedures.

Over-the-Air Updates: Facilitating OTA updates to the cruise control system, ensuring that the latest software and firmware can be deployed to enhance functionality and fix issues without requiring a visit to a service center.



10. Load And Stress Testing:

Load testing and stress testing are both types of performance testing used to evaluate the behavior and stability of systems under various conditions. For a cruise control system in a vehicle, these tests can ensure that the system performs reliably under different scenarios.

Load Testing

Objective:

To determine how the cruise control system performs under expected and peak usage conditions.

To assess the system's capability to handle a specific load without compromising performance.

Focus:

On the system's capacity to manage regular operational demands.

On checking system behavior under normal and peak load conditions.

Scenarios:

Simulating different driving conditions such as highway cruising, city driving, and mixed traffic conditions.

Evaluating the cruise control's response when multiple systems (like air conditioning, infotainment) are also active.

Ensuring the system can maintain a set speed consistently over an extended period.

Metrics:

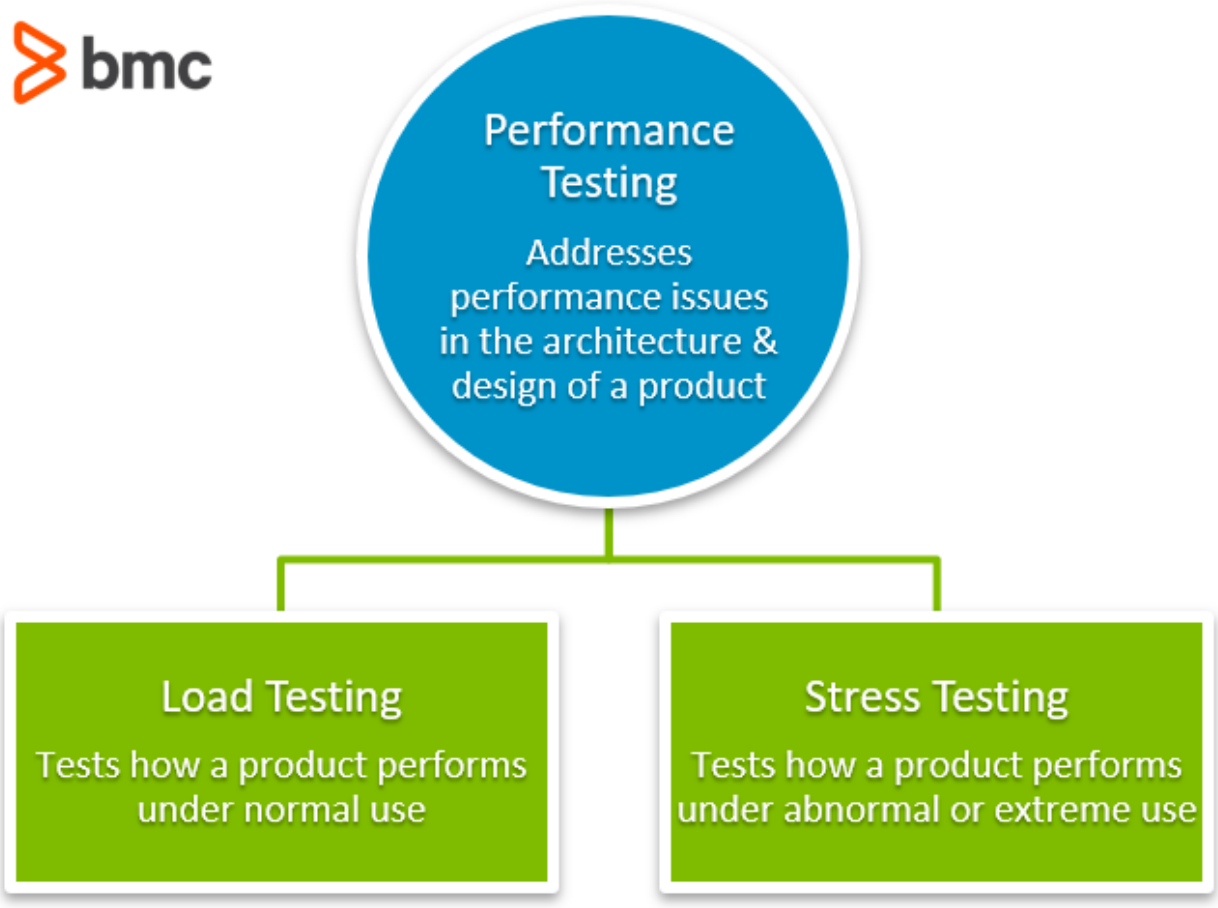
Response time to user inputs (e.g., setting or adjusting speed).

Stability and accuracy in maintaining the set speed.

System resource utilization (CPU, memory).

Detection of any performance bottlenecks.

Load Vs Stress Testing:



Stress Testing

Objective:

To determine the system's robustness and error-handling capabilities under extreme conditions.

To identify the breaking point or failure threshold of the cruise control system.

Focus:

On the system's behavior under extreme, often unrealistic, conditions.

On assessing how the system recovers from failure or handles overload conditions.

Scenarios:

Simulating rapid acceleration and deceleration repeatedly to test the system's response.

Introducing abnormal conditions, such as fluctuating power supply or extreme temperatures, to test the system’s resilience.

Overloading the system with numerous simultaneous inputs or commands to see if it can still function correctly.

Testing how the system handles component failures or sensor malfunctions.

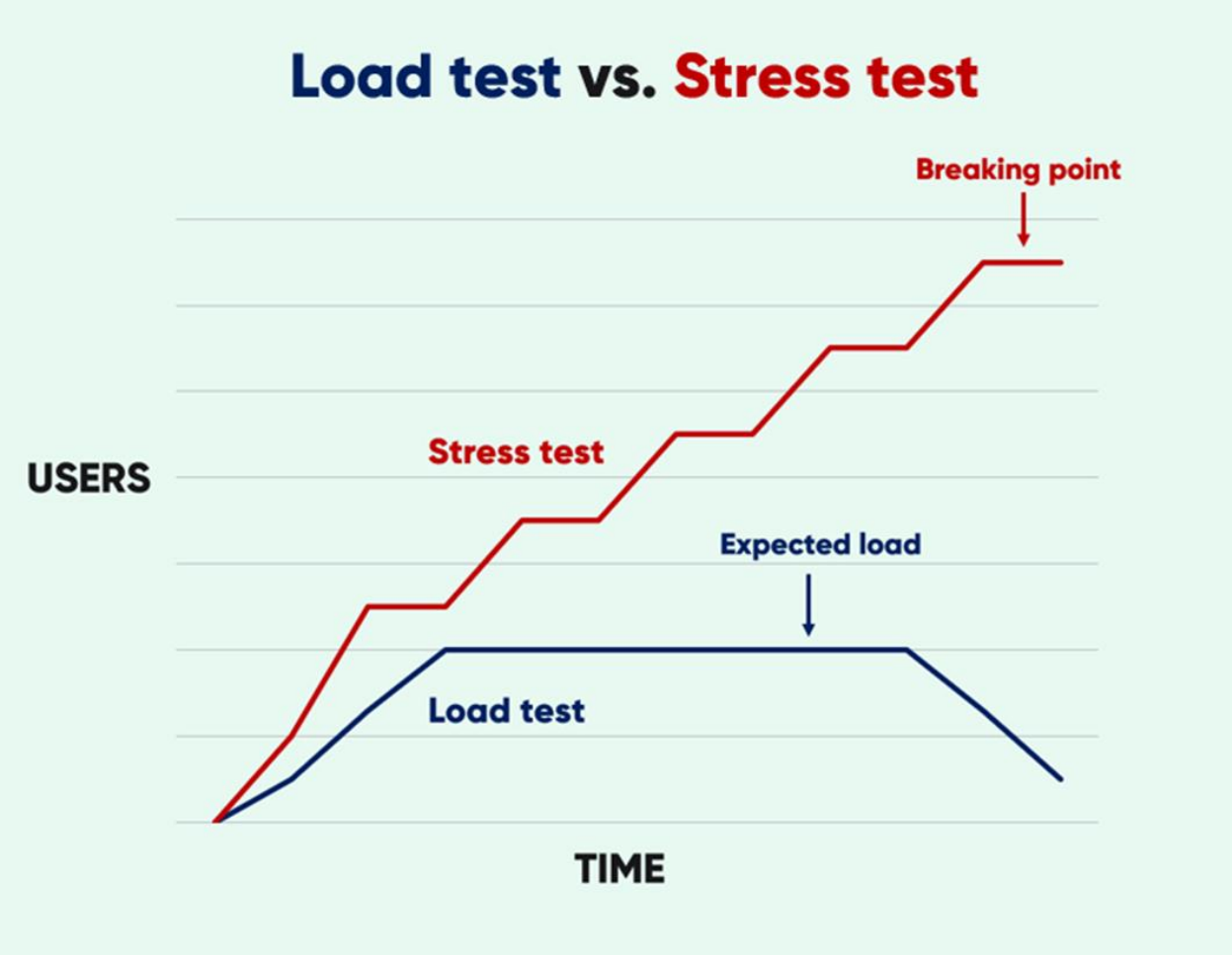
Metrics:

Maximum load the system can handle before failure.

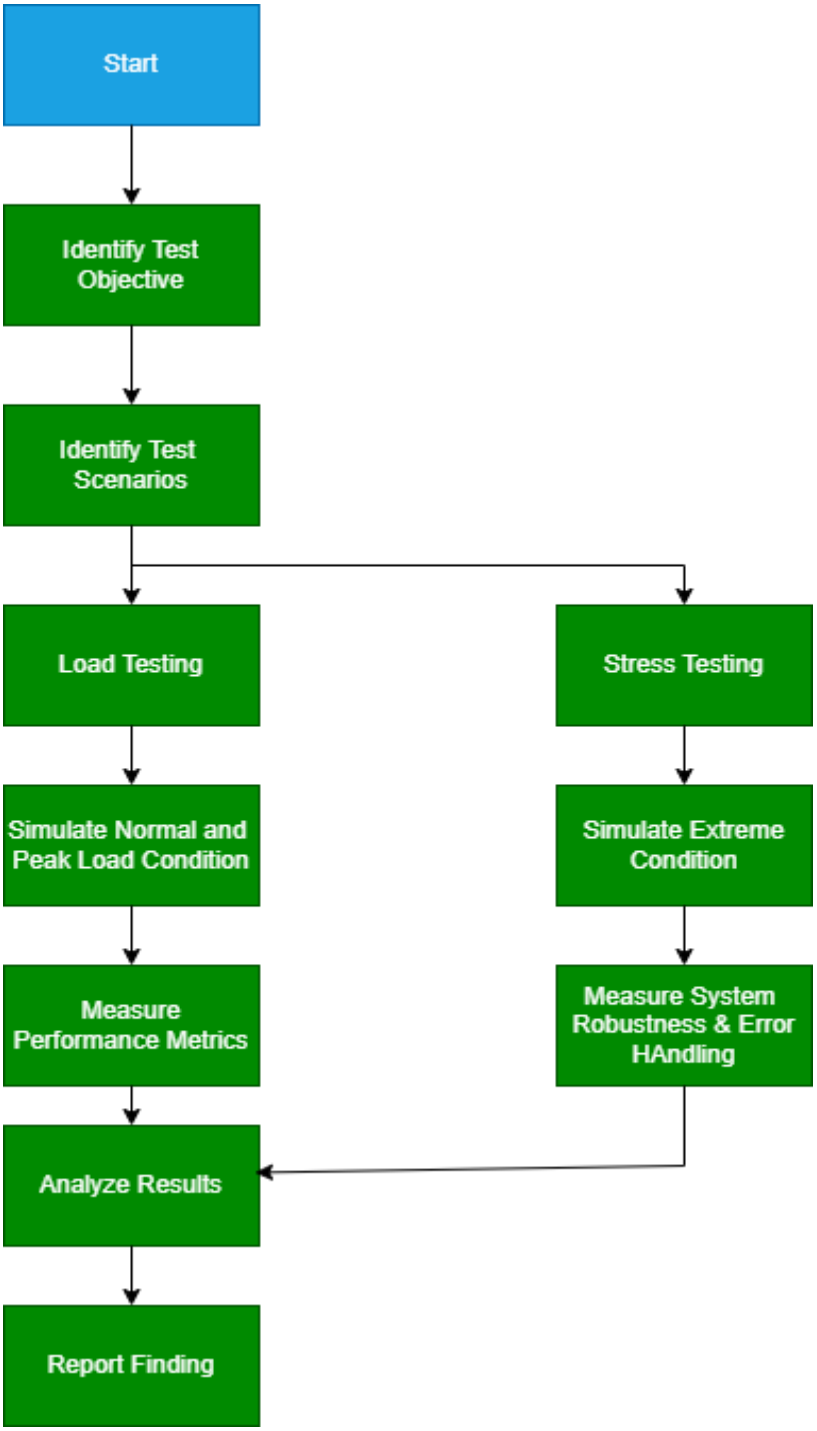
System recovery time after a failure.

Error handling and logging effectiveness.

Impact on system performance under extreme conditions.


Load test vs. Stress test
Breaking point
Stress test
USERS
Expected load
Load test
TIME

Flowchart Of Both Load & Stress Testing:



Explanation of the Flowchart Steps:

1. Start: Initiate the process.

2. Identify Test Objectives: Define the goals of the testing (performance for load testing, robustness for stress testing).

3. Identify Test Scenarios: Determine the specific scenarios for testing both normal/peak loads and extreme conditions.

4. Load Testing:

Simulate Normal and Peak Load Conditions: Create scenarios that mimic typical and peak usage of the cruise control system.

Measure Performance Metrics: Collect data on response time, stability, accuracy, and resource utilization.

5. Stress Testing:

Simulate Extreme Conditions: Create scenarios that push the system beyond its normal operational limits.

Measure System Robustness and Error Handling: Evaluate how the system handles overloads, component failures, and extreme environmental conditions.

6. Analyze Results: Review the data collected to understand the system's performance and robustness.

7. Report Findings: Document and communicate the results of the tests.

8. End: Conclude the testing process.

This flowchart outlines the steps involved in both load and stress testing for a cruise control system, ensuring comprehensive evaluation of the system's performance and resilience.

11. Diagnostics of the Defects

User Interface Diagnostics

Error Display

- Check Diagnostic Information:

- Test Case: Ensure the diagnostic information displayed on the user interface is clear and accurate.

- Expected Result: Users are informed of any system errors related to the cruise control, along with their severity.

- Potential Defect: Diagnostic information is unclear or inaccurate.

- Error Code: Varies based on specific user interface issue.

Severity Indication

- Test Case: The interface should differentiate between minor and critical errors, providing appropriate notifications.

- Expected Result: Users are appropriately notified based on error severity.

- Potential Defect: Errors are not correctly categorized.

- Error Code: Varies based on specific severity indication issue.

Troubleshooting Guidance

- Step-by-Step Solutions:

- Test Case: Validate that the user interface offers troubleshooting steps or suggestions for resolving cruise control errors.

- Expected Result: Users receive helpful troubleshooting guidance.

- Potential Defect: Troubleshooting steps are missing or unhelpful.

- Error Code: Varies based on specific troubleshooting issue.

- User-Friendly Design:

- Test Case: Ensure the diagnostic interface is intuitive, easy to navigate, and accessible to all users.

- Expected Result: Users find the interface easy to use.

- Potential Defect: The interface is difficult to navigate or understand.

- Error Code: Varies based on specific user-friendly design issue.

Diagnostic Logging and Reporting

Log Generation

- Detailed Logs:

- Test Case: Ensure the cruise control system generates detailed diagnostic logs for all detected errors and anomalies.

- Expected Result: Logs are detailed and comprehensive.

- Potential Defect: Logs are missing details or are incomplete.

- Error Code: Varies based on specific logging issue.

- Log Accuracy and Completeness:

- Test Case: Validate that these logs are accurate and comprehensive, covering all relevant data points and events.

- Expected Result: Logs accurately reflect system events.

- Potential Defect: Logs contain inaccuracies or omissions.

- Error Code: Varies based on specific logging issue.

Reporting

- External Reporting:

- Test Case: Test the system's ability to compile and report diagnostic information to external systems or maintenance personnel.

- Expected Result: Reports are generated correctly and comprehensively.

- Potential Defect: Reports are inaccurate or incomplete.

- Error Code: Varies based on specific reporting issue.

- Industry Standards:

- Test Case: Verify that the reporting format adheres to industry standards and can be easily interpreted by external diagnostic tools.

- Expected Result: Reports adhere to industry standards.

- Potential Defect: Reports do not comply with industry standards.

- Error Code: Varies based on specific compliance issue.

Periodic Self-Checks

Automated Diagnostics

- Self-Check Capabilities:

- Test Case: Validate the system's ability to perform periodic self-checks and diagnostics specific to cruise control functions.

- Expected Result: Self-checks are comprehensive and accurate.

- Potential Defect: Self-checks are incomplete or inaccurate.

- Error Code: Varies based on specific self-check issue.

Alert Generation

- Issue Detection Alerts:

- Test Case: Test the system's ability to generate alerts for any issues detected during self-checks.

- Expected Result: Alerts are generated accurately and promptly.

- Potential Defect: Alerts are delayed or inaccurate.

- Error Code: Varies based on specific alert issue.

- Timely Notifications:

- Test Case: Ensure timely notifications of potential problems to the user or maintenance system to prevent any lapse in cruise control functionality.

- Expected Result: Notifications are timely and accurate.

- Potential Defect: Notifications are delayed or inaccurate.

- Error Code: Varies based on specific notification issue.

Making Diagnostics Easier with AUTOSAR

Standardized Interfaces

- Diagnostic Services:

- Test Case: AUTOSAR provides standardized diagnostic interfaces and services, which simplifies the integration and interoperability of diagnostic tools within the cruise control system.

- Expected Result: Diagnostic tools integrate seamlessly.

- Potential Defect: Integration issues with diagnostic tools.

- Error Code: Varies based on specific integration issue.

Modularity

- Layered Architecture:

- Test Case: The layered architecture of AUTOSAR supports modular diagnostics, making it easier to isolate and diagnose specific components or functions within the cruise control system.

- Expected Result: Diagnostics can isolate issues effectively.

- Potential Defect: Difficulty in isolating issues.

- Error Code: Varies based on specific modularity issue.

Scalability

- Flexible Framework:

- Test Case: AUTOSAR's scalable framework supports a wide range of diagnostic capabilities, from basic error reporting to advanced remote diagnostics, ensuring the cruise control system can be effectively monitored and maintained regardless of complexity.

- Expected Result: The system is scalable and maintains effective diagnostics.

- Potential Defect: Scalability issues affecting diagnostics.

- Error Code: Varies based on specific scalability issue.

12. Diagnostic Error Code Analysis

Error Code Documentation

- Comprehensive Documentation:

- Test Case: Ensure that all diagnostic error codes are documented comprehensively.

- Expected Result: Documentation is complete and detailed.

- Potential Defect: Incomplete or unclear documentation.

- Error Code: N/A (Documentation issue without specific DTC).

- Detailed Descriptions:

- Test Case: Provide detailed descriptions for each error code, including possible causes and troubleshooting steps.

- Expected Result: Descriptions are detailed and helpful.

- Potential Defect: Descriptions are vague or unhelpful.

- Error Code: N/A (Documentation issue without specific DTC).

Code Validation

- Correctness of Error Codes:

- Test Case: Validate the correctness of error codes generated by the system.

- Expected Result: Error codes are accurate and correctly identify issues.

- Potential Defect: Incorrect or misleading error codes.

- Error Code: Varies based on specific validation issue.

- Specific Fault Identification:

- Test Case: Ensure that each error code corresponds to a specific fault or issue within the cruise control system.

- Expected Result: Error codes accurately identify faults.

- Potential Defect: Error codes are too general or do not correspond to specific faults.

- Error Code: Varies based on specific fault identification issue.

System Response

- Response to Diagnostic Error Codes:

- Test Case: Test the system's response to various diagnostic error codes.

- Expected Result: The system takes appropriate actions based on the severity and type of error detected.

- Potential Defect: Inappropriate or delayed response to error codes.

- Error Code: Varies based on specific response issue.

Code Mapping

- Mapping to System Components:

- Test Case: Map error codes to specific system components and functions.

- Expected Result: Mapping is accurate and helps in pinpointing the root cause of issues.

- Potential Defect: Inaccurate or incomplete mapping.

- Error Code: Varies based on specific mapping issue.

Historical Analysis

- Recurring Issues Identification:

- Test Case: Perform historical analysis of error codes to identify recurring issues and trends.

- Expected Result: Recurring issues are identified and addressed.

- Potential Defect: Recurring issues are not identified.

- Error Code: Varies based on specific historical analysis issue.

- System Improvement:

- Test Case: Use historical analysis to improve system reliability and performance over time.

- Expected Result: System reliability and performance improve.

- Potential Defect: No noticeable improvement in reliability or performance.

- Error Code: Varies based on specific system improvement issue.

Automated Error Resolution

- Automated Resolution Capabilities:

- Test Case: Test the system's ability to automatically resolve certain types of errors.

- Expected Result: Automated resolution processes work correctly and do not compromise system integrity.

- Potential Defect: Automated resolutions are ineffective or cause additional issues.

- Error Code: Varies based on specific automated resolution issue.

13. Diagnostic Error Codes for Cruise Control System

A. Unit Test Cases

1. Application Layer

Set Speed Function:

- Test Case: Verify that the set speed function correctly maintains the desired speed.

- Expected Result: The vehicle maintains the set speed accurately without fluctuations.

- Potential Defect: The vehicle speed fluctuates instead of maintaining a constant speed.

- Error Code: P0500 (Vehicle Speed Sensor Malfunction).

2. Runtime Environment (RTE)

Service Call Execution:

- Test Case: Validate that service calls from the Cruise Control application to other software components execute correctly.

- Expected Result: All service calls execute without errors and within the required time frame.

- Potential Defect: Service call delays leading to delayed throttle response.

- Error Code: N/A (Performance issue without specific DTC).

3. Basic Software (BSW) Modules

Communication Service Validation:

- Test Case: Test the data exchange between the Cruise Control system and other ECUs via the CAN network.

- Expected Result: Data packets are transmitted and received correctly without loss.

- Potential Defect: Data packets are lost or corrupted during transmission.

- Error Code: U0100 (Lost Communication with ECM/PCM).

B. Integration Test Cases

1. User Interface Integration

Touchscreen Functionality:

- Test Case: Verify the touchscreen interface responds correctly to inputs for setting and adjusting the cruise control speed.

- Expected Result: The touchscreen responds accurately and swiftly to all inputs.

- Potential Defect: Touchscreen unresponsive or inputs are not correctly processed.

- Error Code: B1503 (Touch Screen Circuit Malfunction).

2. Sensor Integration

Brake Switch Input:

- Test Case: Test the system's response to brake switch input by pressing the brake pedal while cruise control is active.

- Expected Result: Cruise Control disengages immediately upon brake application.

- Potential Defect: Cruise Control does not disengage or delays in disengaging.

- Error Code: P0571 (Brake Switch "A" Circuit Malfunction).

C. System Test Cases

1. System Boot and Shutdown

Startup Sequence:

- Test Case: Verify that the Cruise Control system initializes correctly during vehicle startup.

- Expected Result: The system initializes without errors and is ready for use.

- Potential Defect: System fails to initialize or shows error messages.

- Error Code: P0650 (Malfunction Indicator Lamp (MIL) Control Circuit).

2. Functional Scenarios

Maintaining Speed Uphill:

- Test Case: Test the vehicle's ability to maintain the set speed while driving uphill.

- Expected Result: The vehicle maintains the set speed with appropriate throttle adjustments.

- Potential Defect: Vehicle slows down or speeds up unexpectedly.

- Error Code: P0500 (Vehicle Speed Sensor Malfunction).

D. Regression Test Cases

1. Re-run Previous Test Cases

Existing Feature Validation:

- Test Case: Re-run all previous test cases to ensure no existing functionalities are broken by new updates.

- Expected Result: All previously passed test cases should pass again.

- Potential Defect: Previously fixed issues reappear.

- Error Code: Varies based on specific defect reoccurrence.

E. Performance Test Cases

1. Latency and Throughput

Response Time:

- Test Case: Measure the response time for the system to engage or disengage Cruise Control.

- Expected Result: The system should engage or disengage within the specified time limits.

- Potential Defect: Delayed engagement or disengagement.

- Error Code: N/A (Performance issue without specific DTC).

F. Compliance Test Cases

1. AUTOSAR Standards

Module Configuration:

- Test Case: Validate that all software components conform to AUTOSAR standards.

- Expected Result: All components are compliant and properly configured.

- Potential Defect: Non-compliance with AUTOSAR standards.

- Error Code: N/A (Compliance issue without specific DTC).

10.1 Potential Defects and Error Codes

1. P0500: Vehicle Speed Sensor Malfunction

- Description: This code indicates a malfunction in the vehicle speed sensor, which can affect the ability to maintain the set speed.

2. P0564: Cruise Control Multi-Function Input "A" Circuit

- Description: This code indicates an issue with the multi-function input circuit for cruise control, affecting input recognition.

3. P0571: Brake Switch "A" Circuit Malfunction

- Description: This code points to a malfunction in the brake switch circuit, which is crucial for disengaging the cruise control.

4. P0581: Cruise Control Multi-Function Input "A" Circuit High

- Description: This code indicates a high voltage issue in the multi-function input circuit for cruise control, affecting control inputs.

5. U0100: Lost Communication with ECM/PCM

- Description: This code signifies a loss of communication with the engine control module or powertrain control module, which can disrupt cruise control operation.

6. B1503: Touch Screen Circuit Malfunction

- Description: This code indicates a malfunction in the touch screen circuit, affecting user interface functionality.

7. P0650: Malfunction Indicator Lamp (MIL) Control Circuit

- Description: This code points to an issue with the MIL control circuit, which can indicate broader system failures.

Conclusion

In conclusion, this test plan serves as a comprehensive guide to ensure that the Cruise Control System not only meets but exceeds the required standards of functionality, integration, performance, and security. By following this plan, project managers, developers, testers, and stakeholders can work together effectively to deliver a high-quality, reliable cruise control system that enhances vehicle safety and user satisfaction.

THANK YOU