

**AI Interface for Automation in Clusters Automation
At Visteon Technical and Service Center Pvt.Ltd**

Submitted in Partial fulfilment of the Requirements for the Award of
Bachelor of Engineering in Electronics and Communication.

By

Name: Maddu Bhavyan Sai Raj

Reg no:38130117



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

January 2020



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of M Bhavyan Sai Raj (38130117) who carried out the project entitled **AI Interface for Automation in Clusters Automation** under my supervision from 17th Jan 2022 to 17th June 2022

Mr.M.A.Muthiah

(Name in Capital letters
with Signature)

External Guide (if Applicable)

(Name in Capital letters
with signature)

Head of the Department

(Name in Capital letters with Seal & Signature)

Submitted for Viva voce Examination held on_

Internal Examiner

(Name in Capital letters with signature)

External Examiner

(Name in Capital letters with Signature)

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to Board of Management of **SATHYABAMA UNIVERSITY** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D. Dean, School of Electrical and Electronics Engineering** and **Dr. T. RAVI, M.E., Ph.D. Head of the Department, Department of Electronics and Communication Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Dr. M.A.Muthiah, Department of Electronics and Communication Engineering** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of the project work.

We wish to express our thanks to all teaching and non-teaching staff members of the **Department of Electronics and Communication Engineering** who were helpful in many ways for the completion of the project.

We express our gratitude to our parents for their constant encouragement and support for the completion of the project.

ABSTRACT

The Instrument Cluster in every car plays a key role for a driver whether it comes to warnings or tell-tales or any notifications regarding the performance of the car. This Cluster needs to be validated in a precised way, validating in a manual way takes a lot of time and we cannot validate all test cases in the time given, so here comes the role of Automation where number of test cases are validated at very less time and many different combinations can also be validated. We use Vision tool for Automation purpose. Here all the rules for checking and comparing the outputs are done by Image Processing. For that I use PROVE Tech as my script compiler and I used Win wrap as the scripting language

KEYWORDS

Instrument Cluster, PROVE Tech, CAN, Win-wrap, Vision, Artificial Intelligence, ECU (electronic Control unit), OCR (Optical Character Recognition), ROI (Region of Interest), Manual testing, Automation, Test bench, Unified Diagnostic Services (UDS), Test case, Bench Setup, Computer Vision, Machine Learning, Deep learning, Neural Networks, OCR, automating validation.

Literature survey

Author	Title Of The Project	Problem Addressed
Elumalai, Kannan	Hyundai AI3 AC3 Visteon	System Requirement for the project
Dr.B.Sharmila1, P.Venkadesan2	Automation for Instrument Cluster Panel Testing Using Machine Vision and NI-PXI Hardware-In-the-Loop	to develop automated instrument cluster panel testing and validation for shorten product development life cycle in automotive industry.
<u>M. Deepan Raj</u> ; <u>V. Sathiesh Kumar</u>	Vision based feature diagnosis for automobile instrument cluster using machine learning	utilizing the hardware-in-the-loop (HIL) with the vision based machine learning technique to make end to end automation in the feature diagnosis and validation of automotive instrument clusters
Assoc. Prof. Atanasov A.	FEATURES OF THE TESTING OF AUTOMOTIVE SOFTWARE	Usage of ECU and testing them in automotive world

Marius Alexandru Taut ; Mihai Daraban ; Dan Pitica	Software instrument for testing cluster devices using high-speed CAN protocol	This paper aims to present a new method for automated testing of industrial equipment, used primarily in the automotive field. The underlying concept of this work consists in developing an application to test the equipment (DUT - device under test) that automated tests need to be performed on, from the point of view of the hardware system, and in terms of the software loaded on such a device.
Sergiu Nimara ; Dorina Bianca Popa ; Razvan Bogdan	Automotive instrument cluster screen content validation	This paper describes a novel solution to provide this kind of screenshots to the car manufacturers, called Image Grabber application, using a DSLR digital camera, a CAN case

1.Introduction

1.1 Testing:

What is testing: In general, testing is finding out how well something works. In terms of human beings, testing tells what level of knowledge or skill has been acquired. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met.

The main benefit of testing is the identification and subsequent removal of the errors. However, testing also helps developers and testers to compare actual and expected results to improve quality. If the software production happens without testing it, it could be useless or sometimes dangerous for customers.

There are two types of testing's BLACK BOX TESTING and WHITE BOX TESTING

1.2 Black box testing

is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

Black box testing can be done in the following ways:

1.2.1. Syntax Driven Testing

This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers, language that can be represented by context free grammar. In this, the test cases are generated so that each grammar rule is used at least once.

1.2.2. Equivalence partitioning

It is often seen that many types of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into several equivalence classes such that each member of class works in an equivalent way, i.e., if a test case in one class results in some error, other members of class would also result into same error.

The technique involves two steps:

1. **Identification of equivalence class** – Partition any input domain into minimum two sets: **valid values** and **invalid values**. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.

2. Generating test cases –

- (i) To each valid and invalid class of input assign unique identification number.
- (ii) Write test cases covering all valid and invalid test cases considering that no two invalid inputs mask each other.

To calculate the square root of a number, the equivalence classes will be:

(a) Valid inputs:

- The whole number which is a perfect square- output will be an integer.
- The whole number which is not a perfect square- output will be decimal number.
- Positive decimals

(b) Invalid inputs:

- Negative numbers (integer or decimal).
- Characters other than numbers like “a”, “!”, “;”, etc.

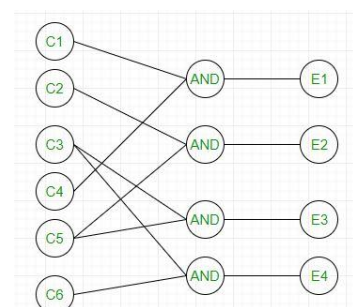
3. Boundary value analysis – Boundaries are particularly good places for errors to occur. Hence if test cases are designed for boundary values of input domain, then the efficiency of testing improves and probability of finding errors also increases. For example – If the valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

4. Cause effect Graphing – This technique establishes a relationship between logical input called causes with corresponding actions called effect. The causes and effects are represented using Boolean graphs. The following steps are followed:

1. Identify inputs (causes) and outputs (effects).
2. Develop cause effect graph.
3. Transform the graph into the decision table.
4. Convert decision table rules to test cases.

For example, in the following cause effect graph:

It can be converted into a decision table like:



		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	x	-	-	-
	E2	-	x	-	-
	E3	-	-	x	-
	E4	-	-	-	x

Each column corresponds to a rule which will become a test case for testing. So, there will be 4 test cases.

5. Requirement based testing – It includes validating the requirements given in SRS (Software Requirement Specification) of software system.

6. Compatibility testing – The test case result not only depends on product but also infrastructure for delivering functionality. When the infrastructure parameters are changed, it is still expected to work properly. Some parameters that generally affect compatibility of software are:

1. Processor (Pentium 3, Pentium 4) and number of processors.
2. Architecture and characteristics of machine (32 bit or 64 bit).
3. Back-end components such as database servers.
4. Operating System (Windows, Linux, etc.).

1.3 White Box Testing:

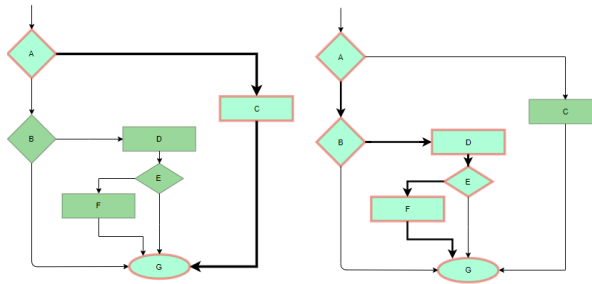
White box testing techniques analyze the internal structures of the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing, or structural testing.

1.3.1 Working process of white box testing:

- **Input:** Requirements, Functional specifications, design documents, source code.
- **Processing:** Performing risk analysis for guiding through the entire process.
- **Proper test planning:** Designing test cases to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- **Output:** Preparing final report of the entire testing process.

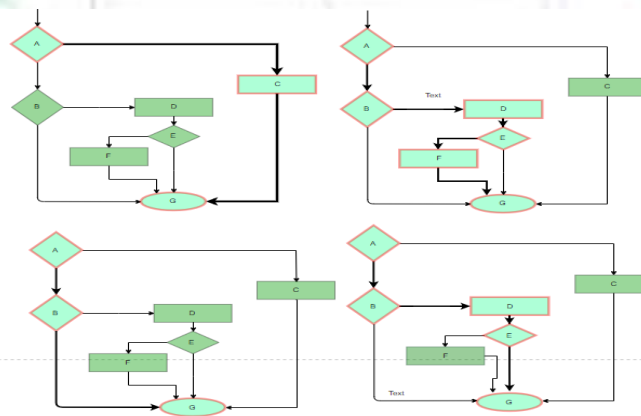
1.3.2 Testing techniques:

- **Statement coverage:** In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



Statement Coverage Example

- **Branch Converge:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



4 test cases required such that all branches of all decisions are covered, i.e., all edges of flowchart are covered

- **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:

1. READ X, Y
2. IF (X == 0 || Y == 0)
3. PRINT '0'

In this example, there are 2 conditions: X == 0 and Y == 0. Now, test these conditions to get TRUE and FALSE as their values. One possible example would be:

4. #TC1 – X = 0, Y = 55
5. #TC2 – X = 5, Y = 0

- **Multiple Condition Coverage:** In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

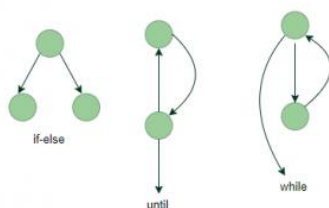
1. READ X, Y
2. IF (X == 0 || Y == 0)
3. PRINT '0'
4. #TC1: X = 0, Y = 0
5. #TC2: X = 0, Y = 5
6. #TC3: X = 55, Y = 0
7. #TC4: X = 55, Y = 5

Hence, four test cases are required for two individual conditions. Similarly, if there are n conditions then 2^n test cases would be required.

- **Basis Path Testing:** In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

Steps:

1. Make the corresponding control flow graph
 2. Calculate the cyclomatic complexity
 3. Find independent paths
 4. Design test cases corresponding to each independent path
- **Flow graph notation:** It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.

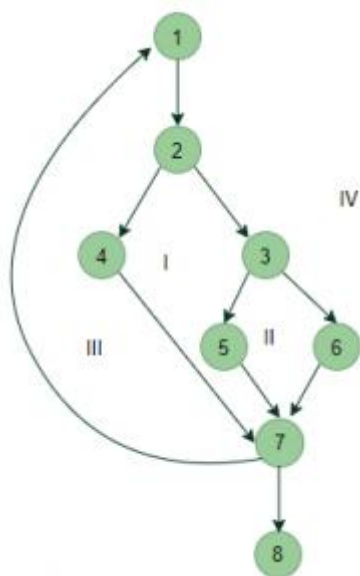


1.3.4 Cyclomatic Complexity:

It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G , $V(G)$ is its cyclomatic complexity. Calculating $V(G)$:

5. $V(G) = P + 1$, where P is the number of predicate nodes in the flow graph
6. $V(G) = E - N + 2$, where E is the number of edges and N is the total number of nodes
7. $V(G) = \text{Number of non-overlapping regions in the graph}$

Example:



$V(G) = 4$ (Using any of the above formulae)

No of independent paths = 4

8. #P1: 1 – 2 – 4 – 7 – 8
9. #P2: 1 – 2 – 3 – 5 – 7 – 8
10. #P3: 1 – 2 – 3 – 6 – 7 – 8
11. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

- **Loop Testing:** Loops are widely used, and these are fundamental to many algorithms hence, their testing is particularly important. Errors often occur at the beginnings and ends of loops.

1. **Simple loops:** For simple loops of size n , test cases are designed that:

- Skip the loop entirely
 - Only one passes through the loop
 - 2 passes
 - m passes, where $m < n$
 - $n-1$ Ans $n+1$ passes
2. **Nested loops:** For nested loops, all the loops are set to their minimum count, and we start from the innermost loop. Simple loop tests are conducted for the innermost loop, and this is worked outwards till all the loops have been tested.
 3. **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each. If they're not independent, treat them like nesting.

Advantages:

1. White box testing is very thorough as the entire code and structures are tested.
2. It results in the optimization of code removing error and helps in removing extra lines of code.
3. It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
4. Easy to automate.

Disadvantages:

1. Main disadvantage is that it is expensive.
2. Redesign of code and rewriting code needs test cases to be written again.
3. Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.
4. Missing functionalities cannot be detected as the code that exists is tested.
5. Overly complex and at times not realistic.

2 Levels of Testing in Software Testing

Tests are grouped together based on where they are added in SDLC or by the level of detailing they contain. In general, there are four levels of testing: unit testing, integration testing, system testing, and acceptance testing. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

There are many different testing levels which help to check behavior and performance for software testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. In SDLC models there are characterized phases such as requirement gathering, analysis, design, coding or execution, testing, and deployment. All these phases go through the process of software testing levels.

2.1 Levels of Testing

There are mainly four **Levels of Testing** in software testing:

1. **Unit Testing:** checks if software components are fulfilling functionalities or not.
2. **Integration Testing:** checks the data flow from one module to other modules.
3. **System Testing:** evaluates both functional and non-functional needs for the testing.
4. **Acceptance Testing:** checks the requirements of a specification or contract are met as per its delivery.

Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

2.1.1 Unit testing:

A Unit is the smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

Levels of Testing	
Unit Test	Test Individual Component
Integration Test	Test IntegratedComponent
System Test	Test the entire System
Acceptance Test	Test the final System

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

2.1.2 Integration testing:

Integration means combining. For Example, in this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

2.1.3 System testing:

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

2.1.4 Acceptance testing:

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

2.2 Fundamentals of Software Testing

2.2.1 Testing Activities:

Testing is commonly perceived to be only about test execution. The typical activities performed to achieve test objectives are test planning, test specifications definition, test execution, test recording, and test reporting.

Let us discuss these activities separately.

2.2.2 Test Planning

Test planning is the initial stage of testing. All testing activities are planned, which include resource requirements regarding human resource, training, software, tools, timelines, and risk and mitigations.

2.2.3 Test Specification

In the test specification stage, the test scenarios, test conditions, and test cases are derived from business requirements documents. All these artifacts are combined using traceability, which helps in determining the requirements coverage achieved during the testing effort. Traceability also helps in the maintainability of the test

deliverable in case of changes to a requirement. Prioritization of test case execution is also done at this stage, considering business risks associated with each requirement as the main factor.

2.2.4 Test Execution

During test execution, the tester executes all planned test cases and verifies whether the expected results match with the actual results. This can be performed manually or automatically using appropriate tools based on the type of testing.

2.2.5 Test Recording

Test recording is the process where, as a proof of execution, the tester documents all test results as test log.

2.2.6 Test Reporting

Test reports are retrieved either manually or automatically.

Test reports help in understanding the progress in testing made till date or at a frequency defined in the test plan.

Test recording is used to evaluate the quality of the software.

Testing for a phase can be considered complete once the exit criteria defined in the test plan is met.

Test closure report is an important deliverable before the testing activity is considered complete.

This can be at the end of the testing phase or at the end of entire project testing in distinct phases.

2.2.7 Document Review

Document review is also considered as part of testing. It helps in identifying defects in the early phases of testing when the executable code is under development.

Document review can help find missing or incorrect requirements, and defects in the design and even in the architecture of the application.

In the next section, we will discuss the risks involved in testing and mitigation.

Risk Involved in Testing and Mitigation

All possible scenarios cannot be tested; just as how the end user uses the product cannot be predicted. The features impacted by the latest code changes are also unknown. Considering these facts, the decision to move a product to the next phase of the software development lifecycle is always accompanied by risk.

Let us see how these risks can be controlled.

- Risks can be mitigated with automated testing that has significant coverage.
- Acceptance level tests with commonly used features in the product can also help reduce the risk.
- Test reporting should provide sufficient information to stakeholders to make decisions for the next development step or release to customers.
- The effort used in quality assurance and testing activities needs to be tailored according to the risks and costs associated with the project.
- Due to the limit in the budget, time, and testing; plan on how to focus the testing based on the risks.

In the following section, we will find out when to stop testing.

2.3 Timeline to Stop Software Testing

In software testing, it is important to know when to stop the process. If the aim is zero software defects, the testing process may never get completed.

In the figure given below, the Y-axis depicts the value of testing, and the X-axis depicts the cost of testing.

At the onset of the testing process, the cost of testing is less; however, the value delivered is extremely high as there are many critical defects in the system.

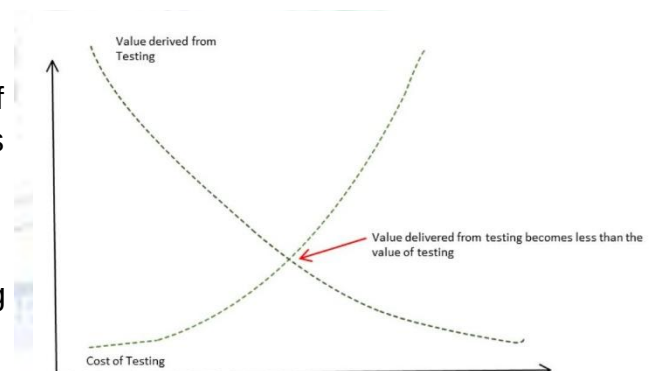
With time, the cost of testing keeps increasing due to the addition of resources.

However, the value of testing drops as most critical defects have already been addressed in the previous cycles.

Value can also drop as some identified defects may not be significant enough to cause software failure.

In addition to evaluating risks associated with testing, it is also wise to consider the cost-benefit analysis to decide when to stop testing.

When the “value delivered from testing becomes less than the cost incurred to run the tests,” the testing should be stopped. This has been depicted by the red arrow in the figure on the section.



We will look at an example to elaborate this concept in the following section.

2.4 Timeline to Stop Software Testing – Example

Three months into testing of a major release of the online railway reservation system, the Test Manager was still not confident of the quality of the release.

The holiday season was fast approaching, and the Manager was under pressure to release the product, to support the load of holiday bookings. How should the Test Manager decide whether to give permission for the release of the product?

The Test Manager needs to analyse the following information for decision-making:

- Whether all the critical functionalities have been thoroughly tested.
- If there are any open critical defects.
- Whether there are more critical defects being reported in the ongoing testing.
- The risk involved if the application is released without further testing; and
- If the application is ready to handle the load of the holiday season.

If all the above criteria are met, the Test Manager should give permission to the release of the product. The business opportunity owing to the holiday season calls for that the application is released if there is only a minimal risk involved.

In the following section, we will discuss the fundamental test process.

2.5 Fundamental Test Process

You have previously learned about the test activities performed at an elevated level to meet the defined testing objectives; let us now find out how these activities are mapped to the distinct phases of a test lifecycle.

As you can see below, these phases are:

1. Test Planning and Control
2. Test Analysis and Design
3. Test Implementation and Execution
4. Evaluating Exit Criteria and Reporting
5. Test Closure

It is important to note that while these phases are sequential, they are also iterative in nature.

For example, during test execution, there may be a need to go back to test design to introduce more test cases or test data before the test execution process is resumed.

Alternatively, during exit criteria evaluation, it can be decided to execute some more tests before the application is considered fit for release. Hence, all phases interact and might transition from one to the other based on the needs of the project.

We will discuss each phase separately in the next few sections.

Let us begin with the first phase of the test process, which is 'Test Planning and Control,' in the following section.

2.5.1 Phase 1 – Test Planning and Control

In the Test Planning and Control phase, you need to ensure the goals and objectives of the customers, stakeholders, and project are understood. Additionally, evaluate the risks of the system to be addressed by testing.

Based on this understanding and as a part of test planning, specify the objective of testing, and determine the scope and risk.

The next activity is to design test strategy, identify the resource requirements, schedule test analysis, and design tasks.

Then, plan for test implementation, execution, and evaluation, and determine the exit criteria for testing.

As a part of planning, you also need to plan for test controls, which will help in measuring the progress against the plan, and in taking corrective actions as and when required.

In the planning phase, we also identify the design of the test environment and identify the required infrastructure and tools. This includes testing and support tools such as spreadsheets, word processors, project planning tools, and non-IT tools and equipment.

In the next section, let us look at the next phase of the test process, which is 'Test Analysis and Design.'

2.5.2 Phase 2 – Test Analysis and Design

The Test Analysis and Design phase involves a review of the test basis and the identification of test conditions.

Let us take a closer look at both these activities.

Review of the test basis includes a review of product requirements, architecture, design specifications, and interfaces between the products. It also includes examining the specifications for the software being tested.

All these artifacts are called “test basis” as these are used as a basis for defining what and how you should test.

The designing of black-box tests can begin before the code is developed.

As the test basis is studied, gaps and ambiguities in the specifications are identified. These gaps and ambiguities occur when there is an attempt to identify incidents arising at every point in the system and helps us in preventing defects that may appear in the code.

After understanding the specifications, identify test conditions based on the analysis of test items, and their specifications and behaviour.

The test conditions provide a high-level list of area of interest in testing. In testing, use the test techniques to define the test conditions.

We will continue our discussion of the activities of this phase in the following section.

The requirements and system are also evaluated for testability in the Test Analysis and Design phase.

The requirements may be written in a way that allows a tester to design tests but may not be testable.

For example, if the performance of the software is important, it should be specified in a testable way.

If the requirement is specified as “the software needs to respond quickly,” it is not testable, as “quick” can be interpreted in more than one way. A more testable requirement could be, “the software needs to respond in 2 seconds with 10 people logged on”.

The testability of the system depends on various aspects of the feasibility of setting up the system in a surrounding that matches the operational environment. The other aspects of test analysis and design phase are comprehensibility and testability of all the possible configurations and uses of a system.

In the next section, let us look at the next phase of the test process, which is ‘Test Implementation and Execution.’

2.5.3 Phase 3 – Test Implementation and Execution

During the third phase, Test Implementation and Execution, the test conditions designed are taken and set up as tests. The test environment is also set up before executing the tests.

Implementation includes prioritizing the test cases, using techniques and test approach, and creating test suites from test cases for efficient test execution. You need to ensure the test environment has been set up correctly by running specific tests on it if possible.

Execution is running the test suites and individual test cases, following the pre-defined test procedures. This is done manually or by using test execution tools according to the planned sequence.

At the end of the execution of each test case, log the outcome and record the identities and versions of the software under test, test tools, and test ware.

You also need to know what tests are used against which version of the software, report defects against specific versions, and maintain the test log to provide an audit trail. Then, compare actual results that is what happened when the test was run, with expected or predicted results.

For differences between actual and expected results, report discrepancies as incidents.

Once the discrepancies have been fixed, repeat test activities to verify whether the fix has resolved them.

Re-execute previously failed tests to confirm a fix is working. This is also known as confirmation testing or re-testing.

Test that the fix did not introduce defects in unchanged areas of the software and that fixing a defect did not uncover other defects. This is called regression testing.

In the next section, we will discuss the fourth phase, which is 'Evaluating Exit Criteria and Reporting.'

2.5.4 Phase 4 – Evaluating Exit Criteria and Reporting

After the end of test execution, the evaluate phase begins.

In this phase, you need to measure the results of test execution against the test objectives and broadcast the test summary report to stakeholders.

Return to test execution if the test objectives have not been met and some more testing needs to be conducted.

Then, prepare the test summary report, which is a document summarizing testing activities and results. The test summary report also contains an evaluation of the corresponding test items against the exit criteria.

In the next section, we will discuss the fifth and final phase of the testing process, which is 'test closure.'

2.5.5 Phase 5 – Test Closure

When testing is considered complete, you can move into the Test Closure phase.

In this phase, you need to perform a check that all deliverables have been accepted and signed off, archive the test ware, close the environment, analyse lessons learned, and use the information to improve the test maturity.

Let us now move on to the next topic, 'Psychology and Ethics of Software Testing' in the following section.

3 Psychology and Ethics of Software Testing

In the next few sections, we will understand the psychology and ethics of software testing.

Let us begin with the psychology of testing in the next section.

3.1 Psychology of Testing

After understanding the complete process of testing, let us now look at the psychology for effective testing.

The degree of independence is vital in performing effective testing.

As seen in the figure on the section, the quality, and effectiveness of testing increase with an increase in the degree of independence.

Let us understand what independence in the context of testing means.

If tests are designed by the person who wrote the software, it will provide a shallow level of independence. This is because the person who wrote the software tends to overlook the defects or may repeat the mistakes made while coding.

Also, Developers may not want to see any defects in their code, so they execute the codes with an intention to get correct results rather than "breaking the system."

Hence, the quality of testing is relatively low.

When tests are designed by another person but from the same development team, they share a similar mindset. However, since the person is not the owner of the software, there is an increased chance to identify the mistakes of the Developer.

In this case, the independence of testing is comparatively more, thereby increasing the effectiveness of testing.

If tests are designed by a person from a different organizational group, for example, by an independent test team, then the degree of independence will be higher.

In this case, they will not be biased as they are not the creators of the code while at the same time, they will have a mindset to “break the system.” As they evaluate software against the requirement, they will be able to catch many more defects including requirements, design, and code-related defects.

If tests are designed and carried out by a person from a different organization or by a company that is a certified external body, they would in addition to a mindset of identifying the defects, also bring in experience in other similar domain, technology, and type of testing.

Hence the quality of testing will be at its best in this scenario.

In the following section, we will compare the mindset of a Developer with a Tester.

3.2 The Mindset of Developer vs. Tester

The Developer always thinks that there are no defects in the code as it was carefully developed. Developers may misinterpret the business requirement, hence assume that the code is functioning the way it is supposed to.

Testers must think like a destroyer. They always think that there are defects in code, waiting to be uncovered. Tester's view requirements from the end-user's point of view and hence can identify user-related defects.

After discussing the mindset of Developers and Testers, let us find out how this gap can be bridged in the following section.

3.2.1 Bridging the Gap

To make any software testing successful, there should not be any difference between the teams involved in building the software. The following considerations can help bridge the gap between Developers and Testers:

- Both teams share a common goal, which is the betterment of the system under test.
- Testers should ensure they are raising the defect against the system and not individuals.
- Both should try and reach a mutual understanding of defects.

- Any incorrect understanding of requirements should be discussed and resolved. Business Analysts can be approached to assist in such situations.

In the next section, through an example, let us find out why a team spirit between a tester and a developer is important.

4 Manual testing

The Manual Test Scenario represents a series of associated actions or functions and helps set up a condition to test or even multiple conditions as they relate to a specific functional area. Where the test case tells you how to test something, the test scenario tells you what to test. Generally, a single test scenario is comprised of several test cases. The associated test cases try to test all possible inputs and outputs using that functional path or area. Each test case associated with a scenario has input values and expected results. They are step by step procedures to test the condition put forth by the overall test scenario.

A test scenario may also be a series of independent tests that, when completed, ensure the business process flow is functioning correctly from beginning to end. Some of these test cases may rely on the output of the previous test and as such, must be executed sequentially. They always represent both typical, day-to-day operations as well as the unusual outlying situations.

An example scenario could be generalized to 'Validate the Login Page' and include multiple test cases to verify the behavior of invalid username and password combinations, 'remember me' functionality, or password resets. Or the scenario could be more specific such as, "User Logs in with invalid credentials". The test cases in the second scenario would be specific to invalid login situations. How generalized or specific a Test Scenario is depending on the QA Professional writing scenario, the time allotted for scenario and test case creation, and the nature of the application under test.

QA Mentor utilizes the use cases, business requirements, and functional specifications to create each test scenario. We then map each one to its associated business requirements and associate them with functional specifications. This helps ensure and prove complete test coverage for the highest quality results.

5 Automation testing

Automation Testing is the process of testing software and other tech products to ensure it meets strict requirements. Essentially, it's a test to double-check that the equipment or software does exactly what it was designed to do. It tests for bugs, defects, and any other issues that can arise with product development.

Although some types of testing, such as regression or functional testing can be done manually, there are greater benefits of doing it automatically. Automation testing can be run at any time of the day. It uses scripted sequences to examine the software. It then reports on what's been found, and this information can be compared with earlier test runs

6 Controller Area Network (CAN)

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed real-time control with a remarkably elevated level of security.

Its domain of application ranges from high-speed networks to low-cost multiplex wiring. In automotive electronics, engine control units, sensors, anti-skid-systems, etc. are connected using CAN with bitrates up to 1 Mbit/s. At the same time, it is cost effective to build into vehicle body electronics, e.g., lamp clusters, electric windows etc. to replace the wiring harness otherwise required.

The intention of this specification is to achieve compatibility between any two CAN implementations. Compatibility, however, has various aspects regarding e.g., electrical features and the interpretation of data to be transferred. To achieve design transparency and implementation flexibility CAN has been subdivided into different layers.

- the (CAN-) object layer
- the (CAN-) transfer layer
- the physical layer

The object layer and the transfer layer comprise all services and functions of the data link layer defined by the ISO/OSI model. The scope of the object layer includes

- finding which messages are to be transmitted
- deciding which messages received by the transfer layer are to be used,
- providing an interface to the application layer related hardware.

There is much freedom in defining object handling. The scope of the transfer layer mainly is the transfer protocol, i.e., controlling the framing, performing arbitration, error checking, error signaling and fault confinement. Within the transfer layer it is decided whether the bus is free for starting a new transmission or whether a

reception is just starting. Also, some general features of the bit timing are regarded as part of the transfer layer. It is in the nature of the transfer layer that there is no freedom for modifications.

The scope of the physical layer is the actual transfer of the bits between the different nodes with respect to all electrical properties. Within one network the physical layer, of course, must be the same for all nodes. There may be, however, much freedom in selecting a physical layer.

The scope of this specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers.

CAN has the following properties

- prioritization of messages
- guarantee of latency times
- configuration flexibility
- multicast reception with time synchronization
- system wide data consistency
- multimaster
- error detection and signaling
- automatic retransmission of corrupted messages as soon as the bus is idle again
- distinction between temporary errors and permanent failures of nodes and autonomous switching off defect nodes

7 Layered Structure of a CAN Node:

Application Layer
Object Layer - Message Filtering - Message and Status Handling

Transfer Layer

- Fault Confinement
- Error Detection and Signaling
- Message Validation
- Acknowledgment
- Arbitration
- Message Framing
- Transfer Rate and Timing

Physical Layer

- Signal Level and Bit Representation
- Transmission Medium

- The Physical Layer defines how signals are transmitted. Within this specification the physical layer is not defined to allow transmission medium and signal level implementations to be optimized for their application.
- The Transfer Layer represents the kernel of the CAN protocol. It presents messages received to the object layer and accepts messages to be transmitted from the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgment, error detection and signaling, and fault confinement.
- The Object Layer is concerned with message filtering as well as status and message handling.

The scope of this specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers

7.1 Messages

Information on the bus is sent in fixed format messages of different but limited length (see section 3: Message Transfer). When the bus is free any connected unit may start to transmit a new message.

7.2 Information Routing

In CAN systems a CAN node does not make use of any information about the

system configuration (e.g., station addresses). This has several important consequences.

- a. **System Flexibility:** Nodes can be added to the CAN network without requiring any change in the software or hardware of any node and application layer.
- b. **Message Routing:** The content of a message is named by an IDENTIFIER. The IDENTIFIER does not indicate the destination of the message, but describes the meaning of the data, so that all nodes in the network are able to decide by MESSAGE FILTERING whether the data is to be acted upon by them or not.
- c. **Multicast:** Because of the concept of MESSAGE FILTERING any number of nodes can receive and simultaneously act upon the same message.

7.3 Data Consistency: Within a CAN network it is guaranteed that a message is simultaneously accepted either by all nodes or by no node. Thus, data consistency of a system is achieved by the concepts of multicast and by error handling.

Bit rate

The speed of CAN may be different in different systems. However, in each system the bitrate is uniform and fixed.

7.4 Priorities

The IDENTIFIER defines a static message priority during bus access.

Remote Data Request

By sending a REMOTE FRAME a node requiring data may request another node to send the corresponding DATA FRAME. The DATA FRAME and the corresponding REMOTE FRAME are named by the same IDENTIFIER.

Multimaster

When the bus is free any unit may start to transmit a message. The unit with the message of higher priority to be transmitted gains bus access.

Arbitration

Whenever the bus is free, any unit may start to transmit a message. If 2 or more units start transmitting messages at the same time, the bus access conflict is resolved by bitwise arbitration using the IDENTIFIER. The mechanism of arbitration guarantees that neither information nor time is lost. If a DATA FRAME and a REMOTE FRAME with the same IDENTIFIER are initiated at the same

time, the DATA FRAME prevails over the REMOTE FRAME. During arbitration, every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal the unit may continue to send. When a 'recessive' level is sent and a 'dominant' level is monitored (see Bus Values), the unit has lost arbitration and must withdraw without sending one more bit.

Safety

To achieve the utmost safety of data transfer, powerful measures for error detection, signaling and self-checking are implemented in every CAN node.

Error Detection

For detecting errors, the following measures have been taken:

- Monitoring (transmitters compare the bit levels to be transmitted with the bit levels detected on the bus)
- Cyclic Redundancy Check
- Bit Stuffing

Message Frame Check

Performance of Error Detection

The error detection mechanisms have the following properties:

- all global errors are detected.
- all local errors at transmitters are detected.
- up to 5 randomly distributed errors in a message are detected.
- burst errors of length less than 15 in a message are detected.
- errors of any odd number in a message are detected.

Total residual error probability for undetected
corrupted messages: less than message error
rate * 4.7
* 10⁻¹¹.

8 MESSAGE TRANSFER

8.1 Frame Types

Message transfer is manifested and controlled by four different frame types:

A DATA FRAME carries data from a transmitter to the receivers.

A REMOTE FRAME is transmitted by a bus unit to request the transmission of the DATA FRAME with the same IDENTIFIER.

An ERROR FRAME is transmitted by any unit on detecting a bus error.

An OVERLOAD FRAME is used to provide for an extra delay between the preceding and the succeeding DATA or REMOTE FRAMES.

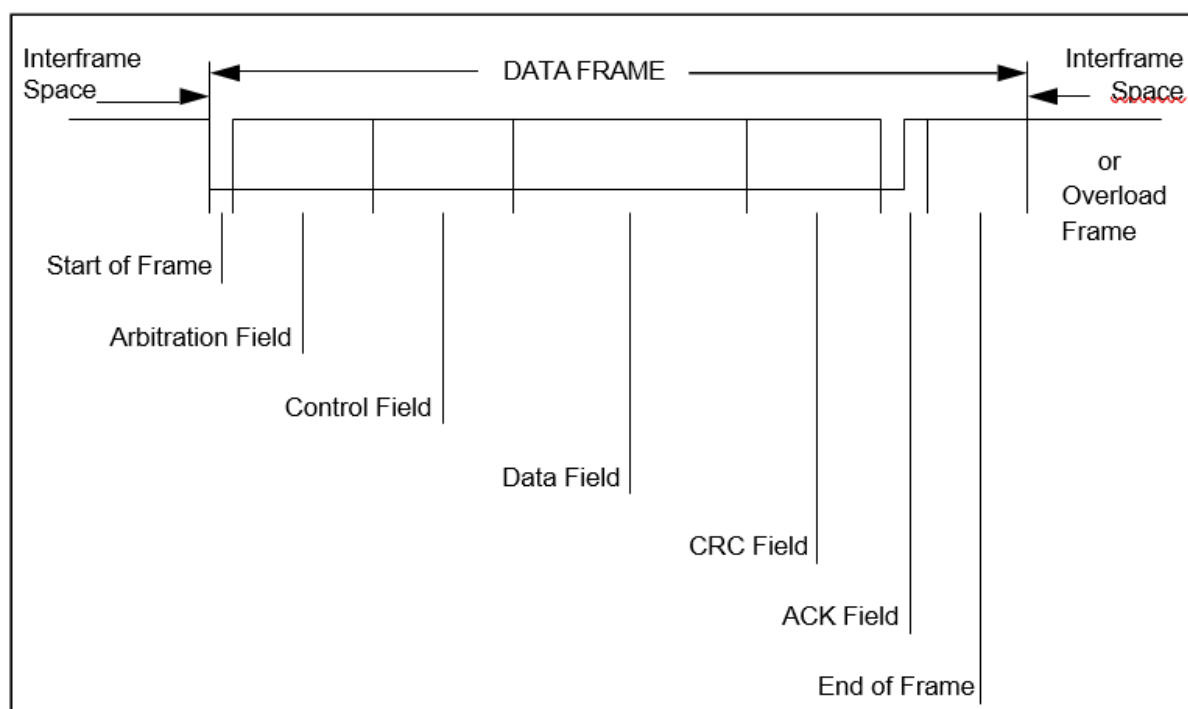
DATA FRAMES and REMOTE FRAMES are separated from preceding frames by an INTERFRAME SPACE.

8.2 DATA FRAME

A DATA FRAME is composed of seven different bit fields:

START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD, CRC

FIELD, ACK FIELD, END OF FRAME. The DATA FIELD can be of zero length.



8.3 REMOTE FRAME

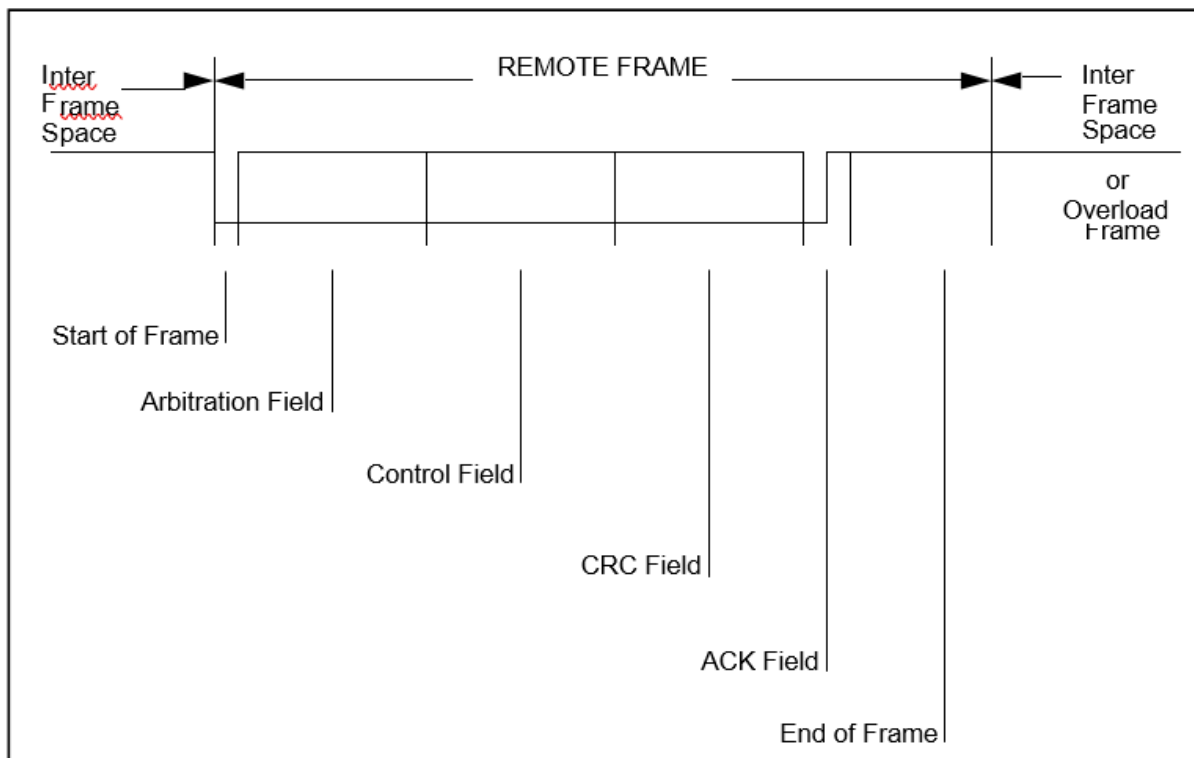
A station acting as a RECEIVER for certain data can initiate the transmission of the respective data by its source node by sending a REMOTE FRAME.

A REMOTE FRAME is composed of six different bit fields:

START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, CRC FIELD, ACK FIELD, END OF FRAME.

Contrary to DATA FRAMES, the RTR (Ready to Receive) bit of REMOTE FRAMES is 'recessive'. There is no DATA FIELD, independent of the values of the DATA LENGTH CODE which may be signed any value within the admissible range 0. 8. The value is the DATA LENGTH

CODE of the corresponding DATA FRAME.



Definition of TRANSMITTER / RECEIVER

TRANSMITTER

transmitter is **an electronic device used in telecommunications to produce radio waves to transmit or send data with the aid of an antenna**. The transmitter can generate a radio frequency alternating current that is then applied to the antenna, which, in turn, radiates this as radio waves.

RECEIVER

a device or apparatus for [receiving](#) signals and convert electronic signals into physical signal

8.4 MESSAGE FILTERING

Message filtering is based upon the whole Identifier. Optional mask registers that allow any Identifier bit to be set 'don't care' for message filtering, may be used to select groups of Identifiers to be mapped into the attached receive buffers.

If mask registers are implemented every bit of the mask registers must be programmable, i.e., they can be enabled or disabled for message filtering. The length of the mask register can comprise the whole IDENTIFIER or only part of it.

9 Test environment Canoe:

Real system software testing of instrumental clusters (IPC (Instrument Panel Cluster)) is related to the usage of special software environment which emulates the work of all or most of the car ECUs. This environment is Vector Canoe. It can be used for software development for any car ECU or for any controller using CAN real-time network communications. Canoe is a Windows oriented tool based on the event driven Communication Access Programming Language CAPL. The simulations of Canoe are based on software nodes (each node corresponds to one car ECU). The nodes are connected via virtual or real CAN network. For each node CAN messages and signals (events) can be defined, and the intercommunication of virtual CAN network can be simulated

10 Unified Diagnostic Services (UDS) :

A diagnostic communication protocol used in electronic control units (ECUs) within automotive electronics, which is specified in the ISO 14229-1.^[1] It is derived from ISO 14230-3 (KWP2000) And the now obsolete ISO 15765-3 (Diagnostic Communication over Controller Area Network (Do CAN)). 'Unified' in this context means that it is an international and not a company-specific standard. By now this communication protocol is used in all new ECUs (Electronic Control Unit) made by Tier 1 suppliers of Original Equipment Manufacturer (OEM), and is incorporated into other standards, such as AUTOSAR. The ECUs in modern vehicles control nearly all functions, including electronic fuel injection (EFI), engine control, the transmission, anti-lock braking system, door locks, braking, window operation, and more. Diagnostic tools can contact all ECUs installed in a vehicle, which has UDS services enabled. In contrast to the CAN bus protocol, which only uses the first and second layers of the OSI model, UDS utilizes the fifth and seventh layers of the OSI model. The Service ID (SID) and the parameters associated with the services are contained in the payload of a message frame.

Modern vehicles have a diagnostic interface for off-board diagnostics, which makes it possible to connect a computer (client) or diagnostics tool, which is referred to as tester, to the communication system of the vehicle. Thus, UDS requests can be sent

to the controllers which must provide a response (this may be positive or negative). This makes it possible to interrogate the fault memory of the individual control units, to update them with new firmware, have low-level interaction with their hardware (e.g., to turn a specific output on or off), or to make use of special functions (referred to as routines) to attempt to understand the environment and operating conditions of an ECU to be able to diagnose faulty or otherwise undesirable behaviour.

11 SID (Service Identifier)

Function group	Request SID	Response SID	Service	Description
Diagnostic and Communications Management	0x10	0x50	Diagnostic Session Control	UDS uses different operating sessions, which can be changed using the "Diagnostic Session Control". Depending on which session is active, different services are available. On start, the control unit is by default in the "Default Session". Other sessions are defined, but do not have to be implemented depending on the type of device: • "Programming Session" used to upload software. • "Extended Diagnostic Session" used to unlock additional diagnostic functions, such as the adjustment of

Function group	Request SID	Response SID	Service	Description
				sensors. "Safety system diagnostic session" used to test all safety-critical diagnostic functions, such as airbag tests. In addition, there are reserved session identifiers that can be defined for vehicle manufacturers and vehicle supplier's specific use.
	0x11	0x51	ECU Reset	The service "ECU reset" is used to restart the control unit (ECU). Depending on the control unit hardware and implementation, different forms of reset can be used: "Hard Reset" simulates a shutdown of the power supply. "Key off on Reset" simulates the drain and turn on the ignition with the key. "Soft Reset" allows the initialization of certain program

Function group	Request SID	Response SID	Service	Description
				units and their storage structures. Again, there are reserved values that can be defined for vehicle manufacturers and vehicle supplier's specific use.
	0x27	0x67	Security Access	Security check is available to enable the most security-critical services. For this purpose, a "Seed" is generated and sent to the client by the control unit. From this "Seed" the client must compute a "Key" and send it back to the control unit to unlock the security-critical services.
	0x28	0x68	Communication Control	With this service, both the sending and receiving of messages can be turned off in the control unit.
	0x29	0x69	Authentication	An update (2020) of the standard added this service to provide a standardized approach to more modern methods of authentication than are permitted by the Security Access (0x27) service, including bidirectional authentication with PKI-based Certificate

Function group	Request SID	Response SID	Service	Description
				Exchange.
	0x3E	0x7E	Tester Present	If no communication is exchanged with the client for a long time, the control unit automatically exits the current session and returns to the "Default Session" back and might go to sleep mode. Therefore, there is an extra service which purpose is to signal to the device that the client is still present.
	0x83	0xC3	Access Timing Parameters	In the communication between the controllers and the client, certain times must be observed. If these are exceeded, without a message being sent, it must be assumed that the connection was interrupted. These times can be called up and changed.
	0x84	0xC4	Secured Data Transmission	
	0x85	0xC5	Control DTC (Diagnostic Trouble Codes)	Enable or disable the detection of any or all errors. This is important when diagnostic work is performed in the car,

Function group	Request SID	Response SID	Service	Description
			Settings	which can cause an anomalous behaviour of individual devices.
	0x86	0xC6	Response On Event	
	0x87	0xC7	Link Control	The Service Link Control is used to set the baud rate of the diagnostic access. It is usually implemented only at the central gateway.
Data Transmission	0x22	0x62	Read Data by Identifier	With this service, it is possible to retrieve one or more values of a control unit. This can be information of all kinds and of different lengths such as Part number or the software version. Dynamic values such as the current state of the sensor can be queried. Each value is associated to a Data Identifier (DID) between 0 and 65535. Normal CAN signals are meant for information that some ECU uses in its functionality. DID data is sent on request only, and is for information that no ECU uses, but a service tool or a software tester can benefit from.

Function group	Request SID	Response SID	Service	Description
	0x23	0x63	Read Memory by Address	Read data from the physical memory at the provided address. This function can be used by a testing tool, to read the internal behaviour of the software.
	0x24	0x64	Read Scaling Data by Identifier	
	0x2A	0x6A	Read Data by Identifier Periodic	With this service, values are sent periodically by a control unit. The values to be sent must be defined to only using the "Dynamically Define Data Identifier".
	0x2C	0x6C	Dynamically Define Data Identifier	This service offers the possibility of a fix for a device specified Data Identifier (DID) pool to configure another Data Identifier. This is usually a combination of parts of different DIDs (Data Identifier) or simply a concatenation of complete DIDs. The requested data may be configured or grouped in the following manner: Source DID, position, length

Function group	Request SID	Response SID	Service	Description
				(in bytes), Sub-Function Byte: defineByIdentifier - Memory address length (in bytes), Sub-Function Byte: defineByMemoryAddress - Combinations of the two above methods through multiple requests.
	0x2E	0x6E	Write Data by Identifier	With the same Data Identifier (DID), values can also be changed. In addition to the identifier, the new value is sent along.
	0x3D	0x7D	Write Memory by Address	The "Write Memory by Address" service allows the external diagnostic tool to write information into the ECU at one or more contiguous memory locations.
Stored Data Transmission	0x14	0x54	Clear Diagnostic Information	Delete all stored DTC
	0x19	0x59	Read DTC Information	DTC stands for " Diagnostic Trouble Codes ". Each DTC handled by the control unit

Function group	Request SID	Response SID	Service	Description
				fault is stored with its own code in the error memory and can be read at any time. In addition to the error, additional information will be stored, which can also be read.
Input / Output Control	0x2F	0x6F	Input Output Control by Identifier	This service allows an external system intervention on internal / external signals via the diagnostic interface. By specifying a so-called possibility byte added conditions for a request can be specified, the following values are specified: <i>ReturnControlToECU</i>: The device must get back controls of the mentioned signals. <i>ResetToDefault</i>: The tester prompts to reset signals to the system wide default value. <i>Freeze Current State</i>: The device shall freeze the current signal value. <i>ShortTermAdjustment</i>: The device shall use the provided value for the signal
Remote	0x31	0x71	Routine	Control routine services of

Function group	Request SID	Response SID	Service	Description
Activation of Routine			Control	all kinds can be performed. There are three different message types: • With the start-message, a service can be started. It can be defined to confirm the beginning of the execution or to notify when the service is completed. • With the Stop message, a running service can be interrupted at any time. • The third possibility is a message to query the results of the service. The start and stop message parameters can be specified. This makes it possible to implement every possible project-specific service.
Upload / Download	0x34	0x74	Request Download	Downloading new software or other data into the control unit is

Function group	Request SID	Response SID	Service	Description
				introduced using the "Request Download". Here, the location and size of the data is specified. In turn, the controller specifies how large the data packets can be.
	0x35	0x75	Request Upload	The service "request upload" is almost identical to the service "Request Download". With this service, the software from the control unit is transferred to the tester. The location and size must be specified. Again, the size of the data blocks is specified by the tester.
	0x36	0x76	Transfer Data	For the actual transmission of data, the service "Transfer Data" is used. This service is used for both uploading and downloading data. The transfer direction is notified in advance by the service "Request Download" or "Upload Request". This service should try to send packets at maximum length, as specified in earlier services. If the data set is larger than the maximum, the "Transfer Data" service must be used

Function group	Request SID	Response SID	Service	Description
				several times in succession until all data has arrived.
	0x37	0x77	Request Transfer Exit	A data transmission can be 'completed' when using the "Transfer Exit" service. This service is used for comparison between the control unit and the tester. When it is running, a control unit can answer negatively on this request to stop a data transfer request. This will be used when the amount of data (set in "Request Download" or "Upload Request") has not been transferred.
	0x38	0x78	Request File Transfer	This service is used to start a file download from the client to the server or upload from the server to the client. Additionally, information about the file system is available by this service.
		0x7F	Negative Response	This response is given when a service request could not be performed, for example having a not supported Data Identifier. A Negative Response Code will be included.

12 Models:

There are many accepted models in the development process. Such as the Spiral Model, the Waterfall-model, the V-Model, and the Agile Model, which are so popular today.

All these models define a systematic way to carry out an orderly way of working during the project.

Testing appears in each of these lifecycle models, but with quite different meanings and different extents.

Some methods work better for a specific type of project, but as a result, the most crucial factor for the project's success may be how closely a particular plan was followed.

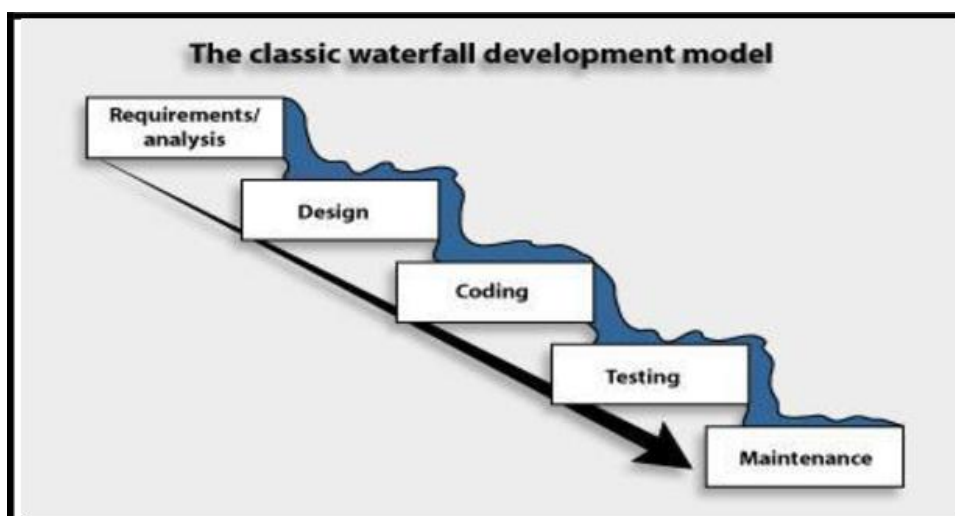
Most of the projects work according to a specific software development model chosen by development but in some cases some different models could be used to work in parallel.

For example, the V-Model or Waterfall model combined with the Agile model.

For Instance, a project using V-Model, but during the developing process, it becomes clear that the project should update some of the requirements cause of the change of design by the customer.

12.1 WATERFALL MODEL

The original SDLC model was the Waterfall model. This model is amazingly simple to understand and very well known in the development process.



The main concept of this model is that only when one development level is completed will the next one be started.

At the end of each phase, a review takes place to figure out if the project is on the right path and whether to continue or discard the project.

12.1.1 ADVANTAGES

- The Waterfall method is also well known by software developers
- The Waterfall Model works well for smaller projects where requirements are very well understood
- Cost-effectiveness: Time spent early in the software production cycle can lead to the greater economy at later stages

12.1.2 DISADVANTAGES

- The crucial disadvantage of this model is that testing is understood as a “one-time” action at the end of the project just before the release of the operation.

The test is seen as a “final inspection”, an analogy to a manufacturing inspection before handing over the product to the customer.

- High amounts of risk and uncertainty
- Inflexible
- Poor model for complex and object-oriented projects

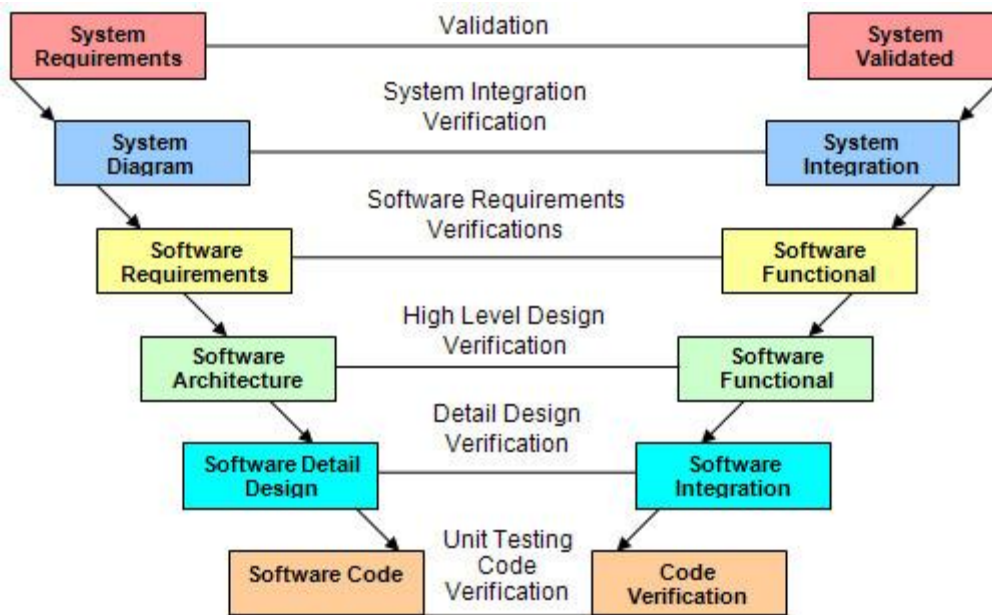
In conclusion, the **Waterfall Model** recommended being used in projects with a clear and deep understanding of project requirements, design, technical tools, and infrastructures.

12.2 V MODEL

The second model is the **V-model**.

This is the most familiar model in the development process. The model has the form of a “V”.

The main idea in the **V-Model** is that development tasks and testing tasks are corresponding activities of equal importance, which is symbolized by the two sides of the “V”.



The development process continues from the upper left point of the V toward the right, ending at the upper right point.

In the left-hand, downward-sloping branch of the V, development personnel define business requirements, application design parameters, and design processes.

At the base point of the V, the code is written. In the right-hand, upward-sloping branch of the V, testing and debugging is done.

The unit testing is carried out first, followed by bottom-up integration testing.

The extreme upper right point of the V stands for product release and ongoing support.

Like in Waterfall-Model each phase must be completed before the next phase begins, V-model is a modified version of a Waterfall-Model.

The **V-model** ordains that the code testing documentation is written in tandem with the development phases that means, for instance, the integration tests should be documented as and when the high-level design is finalized, and the unit tests should be ready as and when the detailed specifications are laid down.

In this model, testers should be involved in reviewing development implementation as soon as possible.

Like the Waterfall-Model testing process and fixing faults can be done at any stage in the life cycle, but the cost of finding and fixing faults increases dramatically as development progresses.

The number and intensity of the test levels may be modified according to the specific needs of the project and for every development stage, there is a corresponding test level.

12.2.1 ADVANTAGES

Due to the fact that in the **V-Model** defects are being repaired a brief time after they have been detected, it is cheaper to fix them.

The model has a reputation for a particularly good base for the partitioning of testing.

All the participants in the development of a system have a responsibility for quality assurance and testing.

Testing activities like requirements, test designing happen well before coding.

This fact saves a lot of time and helps in developing a particularly good understanding of the project at the initial stage.

The objectives of testing are changing, and specific for each test level

12.2.2 DISADVANTAGES

- The **V-Model** is very rigid and the least flexible, which means that if one of the requirements is changing, the tester should update the test documentation.
- This model applicable mostly to big companies because the model needs a lot of resources.
- The amount and the intensity of the test levels should be tailored according to the specific needs of the project.

In summary: In the **V-Model** the place of testing in the development process is critical. The fault-finding occurs in the early stage of the development process which provides the cheaper alternative for fixing it.

12.3 AGILE

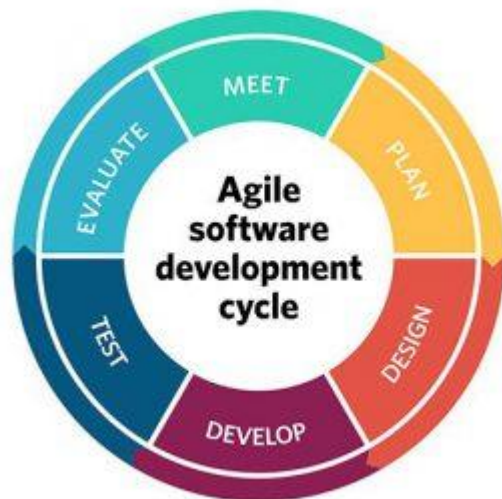
The **Agile Model** of software development is a conceptual framework for software engineering that promotes development iterations throughout the life cycle of the project.

The Agile approach meaning as the name refers implies something to do very quickly.

Hence, Agile Testing refers to validate client requirements as soon as possible and make them customer friendly.

Agile practices are specially tailored to eliminate all kinds of waste manifesting in a product lifecycle, but more especially, improving the quality of the developed product is a major goal threaded into most of that practice.

There are many agile development methods; most minimize risk by developing software in short amounts of time.



Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks.

Each iteration is an entire software project: including planning, requirements analysis, design, coding, testing, and documentation.

In Agile testing as soon as the build is out, testing is expected to get started and report the bugs quickly if any found.

There are some accepted properties to Agile Development, for example:

- Short release cycles- agile development can't be called agile unless it is done in short repetitive iterations.
- Customer involvement. The tester should supply his thoughts on the client's requirements rather than just being the audience at the other end.
- Responding to change – agile development is focused on quick responses to change and continuous development.
- Individuals and interactions over processes and tools

12.3.1 ADVANTAGES

Saving time and money.

Focus more on the application rather than documenting things.

Daily meetings and discussions for the project following an Agile model can help to figure out the issues well in advance and work on them accordingly.

Requirements changing even in the late stage of development

The result is high-quality software in the least possible time duration and satisfied customers.

12.3.2 DISADVANTAGES:

- The project can easily get taken off track if the customer representative is not clear what outcome that they want.
- There is a lack of emphasis on necessary designing and documentation.
- User involvement is often a potential problem especially in big and complex projects
- Agile processes are only applicable to products where reliability is not critical.

13 Tools required:

We use both software and hardware tools

13.1 Hardware tools:

Power Supply (DC (Direct Current)), Cluster Test Box, Programmable resistance box, Instrument Cluster (AI3PE variant), CAN tool, Channel cables (CAN-a (coupling loss network analyser), CAN-b), Cameras 2

13.2 Software Tools:

PROVE Tech, CANOE, Vision

13.1.1 Power Supply (DC):

A DC power supply, also known as a bench power supply, is a type of power supply that gives direct current (DC) voltage to power a device. A DC power supply management subsystem can use AC, DC, battery, or ultralow voltage as inputs.

- Step down transformation, rectification, DC filtration, and regulation are the four major steps in a regulated DC power supply block diagram.

- DC power supply designs are categorized into two types: unregulated power supply and regulated power supply.
- The regulated power supply can be linearly regulated or switched.
- Primary switch mode and secondary switch mode are the two types of switch mode regulated power supplies.
- A DC-DC converter is a type of DC power supply that utilizes DC voltage as an input.
- Two types of DC-DC converters are non-isolating and isolating converters.
- Operating car radio, CB transceiver, or mobile phone, running a personal CD player, energizing one of the latest CPU chips, engaging a PC power supply application, operating an electronic circuitry, supplying an insulation testing voltage, running a car HiFi (High Fidelity) amplifier 's circuitry, and engaging a DC-AC sine wave inverter application are some of the uses of DC-DC converters.
- DC power supplies have four basic outputs: constant voltage, constant current, voltage limit, and current limit.
- Battery eliminators, constant voltage power supply, constant voltage/constant current power supply, programmable supply, and multi-range power supply are the most common DC power supplies available on the market.
- When selecting a DC power supply, one should consider the following specifications: constant current and constant voltage mode, output, regulation, temperature, AC input, ripple and noise, tracking accuracy, and DC isolation.

13.1.2 Programmable resistance box:

Connect the 2090 via RS232 connection (USB to RS232 adapter available) to a computer to automate any task performed by a decade resistance box to automate calibration or ATE (Automated Test Equipment) tasks.

Providing a basic accuracy of up to 0.1% (0.01% accuracy option available), the 2090 is ideal for simulation of any resistance from 0 Ohm through to 99.9999 Kohm.

Control of 2090 is achieved using simple commands that can be sent to the unit from any program or programming language or using the included 2090 Virtual Front Panel software.



2090 provides passive resistance output and mimics the performance of traditional decade boxes exactly.

For use without a computer, the 2090 provides control via keys on the front panel, and a display to indicate the current output.01:18 PM

13.1.3 CAN tool:

The CAN BUS Analyzer Tool is a simple to use low-cost CAN bus monitor which can be used to develop and debug a high-speed CAN network. The tool supports CAN 2.0b and ISO11898-2 and a broad range of functions which allow it to be used across various market segments including automotive, industrial, medical and marine.

The toolkit comes with all the hardware and software required to connect a CAN network to a PC. The Graphical User Interface makes it easy to quickly observe and interpret bus traffic.

13.1.4 INSTRUMENT CLUSTER

Instrument Cluster is an ECU (electronic control unit). Every vehicle has an instrument cluster. It's right in front of the driver. An instrument cluster is a board with several different gauges and warning lights mounted in it. The instrument cluster is mounted in the dashboard, just behind the steering wheel. The driver depends on the gauges and indicators to track the vehicle's status. Some of the most common gauges and indicators include:

- Speedometer
- Tachometer
- Fuel gauge
- Oil pressure gauge
- Odometer
- Tell-Tales & indicators

Some instrument clusters are all electronic. Those instrument clusters have digital readouts instead of traditional Analog gauges.

The design of the instrument cluster has changed in one fundamental way over the last twenty or thirty years. Back then, gauges or instruments received their input from the part they were checking. For example, the fuel level sensor in the fuel tank was directly connected to the fuel gauge in the instrument cluster. In modern cars, all the sensors are connected to the ECU, and the ECU sends all the relevant information

to the instrument cluster. And all these signals are received by the cluster and shown as outputs.

In every industry before a sending a Product (ECU) to customer there it will be tested whether it is satisfying all its requirements given by the customers. Here we test this cluster in two ways Manual testing and Automation

Software Tools

13.2.1 PROVETECH:

PROVEtech: TA (Test Automation) is an efficient test automation tool for a variety of hard- and software platforms (HiL, MiL, SiL). Besides the database-based test development environment, PROVEtech: TA offers modules for vehicle diagnostics (ASAM MCD-3 D), electrical fault simulation, CCP, XCP, J1939, ISO-TP, AUTOSAR and real-time tests. Signals and bus messages (CAN, LIN (Local Interconnect Network), Flex Ray, MOST (Media Oriented Systems Transport), Automotive Ethernet) can be visualized, controlled, measured, and analysed interactively and automated.

13.2.2 WINWRAP

BASIC (Beginners' All-purpose Symbolic Instruction Code) is a family of general-purpose high-level programming languages

Win wrap is the basic scripting language used for controlling the COM Object. There are two kinds of Win wrap scripting.

Win wrap – COM

Win wrap -.NET

- #Language "WWB-COM" for Visual Basic for Applications (TM) compatibility
- #Language "WWB.NET" for Visual Basic.NET (TM) compatibility with the host application .NET environment

13.2.2.1 VARIABLE:

A variable is nothing, but a name given to a storage area that our programs can manipulate.

- The **Dim** statement is used for variable declaration and storage allocation for one or more variables.
- The Dim statement is used at module, class, structure, procedure or block level.

13.2.2.2 Looping Statement:

- We can write a Visual Basic procedure that allows the program to run repeatedly until a condition or a set of conditions is met.
- This procedure is known as looping. Looping is an especially useful feature of Visual Basic because it makes repetitive work easier.
- There are three kinds of loops in Visual Basic
Do...Loop
For.....Next loop
While.....Wend Loop

13.2.2.3 Variant List:

- The class Variant List is a list that can be iterated in both directions. It can store all kinds of values and objects in an exact order. This order can be reproduced.
- Single elements can be referenced with variables of the type of Position. Internally these variables are pointers to the elements. An improper use of position values causes the program to crash. Because of that you must use the position values returned by the methods of Variant List. To iterate a complete list, you can use For Each

13.2.2.4 Procedure:

Win wrap Basic offers diverse types of procedures to execute small sections of coding in applications. Win wrap Basic programs can be broken into smaller logical components called Procedures. Procedures are useful for condensing repeated operations such as the often-used calculations, text and control manipulation etc. The benefits of using procedures in programming are: It is easier to debug a program with procedures which break a program into discrete logical limits. A Procedure can be Sub, Function or Property Procedure

13.2.2.5 Sub Procedure:

A sub procedure is a series of Visual Basic statements enclosed by the sub and end sub statements that perform actions but doesn't return a value. A sub procedure can take arguments, such as constants, variables, or expressions that are passed by a calling procedure. If a sub procedure has no arguments, the sub statement must include an empty set of parentheses.

14 Artificial Intelligence:

Artificial intelligence (AI) is the ability of a computer, or a robot controlled by a

computer to do tasks that are usually done by humans because they need human intelligence and discernment.

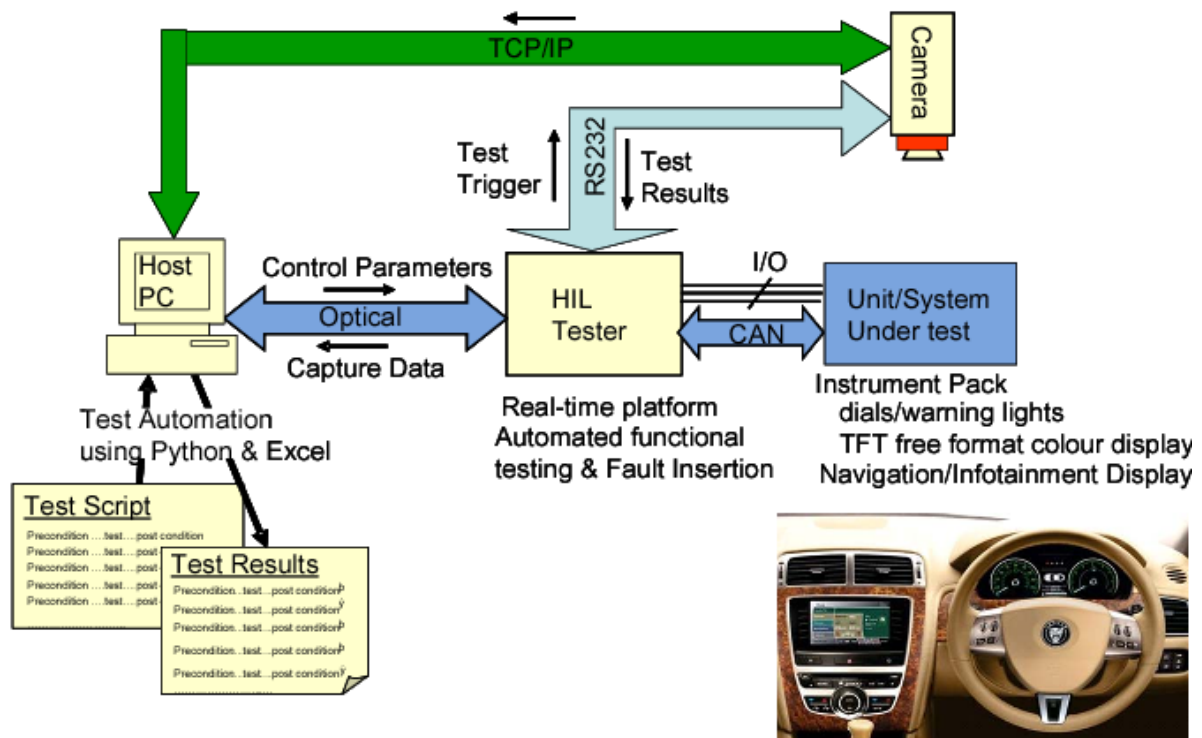
14.1 Implementation of AI in Validation:

Validation of HMI (Human to Man Interface) contents in Driver information system, In-Vehicle Infotainment, and Centre media Display is a big challenge as it's prone to human manual testing errors. Automating the text verification for multi languages and horizontal/vertical scrolling text is a difficult challenge as the intelligence needed to achieve is high. To mitigate this effect vision-based machine/deep learning algorithms will be implemented in such a way that three important applications have been created which serves to be the mother code for all the features that involves text as its main component of automated testing.

They are

1. Multilingual text verification at various illuminations, grade out conditions and various background gradients using OpenCV and Tesseract-OCR.
2. Horizontal & Vertical scrolling text using image key point stitching.
3. Automatic region of interest generation for text regions in the message center display using start v3 deep neural network architecture.

With the power of artificial intelligence in validating text makes it easier to automate with greater accuracy and can also be implemented in other areas like User Setting Menu traversal verification, Clock verification and much can be automated with less time and higher accuracy.



14.2 SYSTEM REQUIREMENTS:

For this application we have used the power of python libraries and Tesseract OCR to achieve end to end automation in various forms for text verification.

14.2.1 Tesseract-OCR:

Hewlett-Packard was the founder of Tesseract between 1985 and 1994 and this tool was made publicly available in the year 2005, and from 2006 Google observed the tool since then google is actively maintaining it, and recently Tesseract has released an alpha version 4.0.0 where it utilizes the power of Recurrent Neural Network/ Long Short-Term Memory. Because of the inclusion of Neural networks, the system started frame words that give meaning. And with this the error rate for all the languages is limited to under 10 %. But Tesseract needs a perfect high-quality image without noise, since the images gathered are from real time noise will be high and the quality will be par. By using image processing techniques, the noise and image quality can be somehow increased in such a way that Tesseract gets a neat image for extracting text.

14.2.2 Open CV:

Image processing techniques are done using this python library . This is an open source and easy to use library where a lot of matrix computation is handled internally making it quite simple and efficient to code. Using an open CV (Capacitive Voltage), we will be eliminating noise, enhancing the graded-out text, stitching of images for scrolling text verification and feature extractions for image can be done using this library.

14.2.3 TensorFlow:

TensorFlow is an open-source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs (Graphics Processing Unit) in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

By using Keras as front-end and by keeping TensorFlow as backend coding complexity can be increased with smallest sacrifice of speed which is not a concern as the processing time is already in the limits. For segmenting texts (to find the portions of text in each image) we need a powerful feature extraction technique that is achieved using CNN by coding through Keras-TensorFlow combination.

14.2.4 SciKit Learn:

SciKit Learn is another open-source Python library which is extensively used for training a machine learning model, the feature extracted data and labels will be trained using this framework, and it has almost all the important machine learning models. Using this we can train models (with the feature vectors and labels which are being extracted using OpenCV and TensorFlow) with different classifiers to know which classification model suits the best in terms of speed and accuracy.

Provetech:

In order to execute test procedures to the Instrument cluster and infotainment system, CAN signals and other hardware emulation can be commanded using this tool, and all the execution can be automated and the Provetech will call Python for text verification and further result will be generated in Provetech by getting feedback of the result from Python.

With this combination any forms of text verification can be done, and it is not limited to text verification and can be extended to verify other features to namely gauges, odometer, telltales etc.

14.3 METHODOLOGY OF AI Implementation:

The report explains three techniques namely

1. Multilingual text verification at various illuminations, grade out conditions and various background gradients using OpenCV and Tesseract-OCR.
2. Multilingual Horizontal & Vertical scrolling text using image key point stitching.
3. Automatic region of interest generation for text regions in the warning pages using deep neural network architecture.

14.3.1 Multilingual Text verification (OCR):

OCR is nothing but Optical Character Recognition (i.e.) it can recognize character which is in the format of image or pdf. The OCR will achieve great accuracy if the characters in the words are well organized like shown in the below fig.

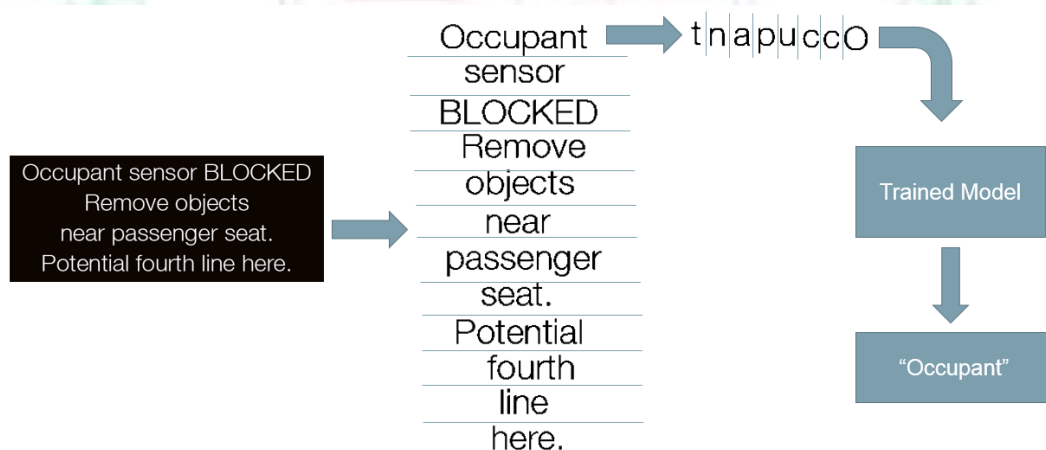


Fig. Normal OCR working

The biggest problem occurs in the character chopping mainly due to some properties like Kerning, Connected Character and interspace in single word as shown in fig.

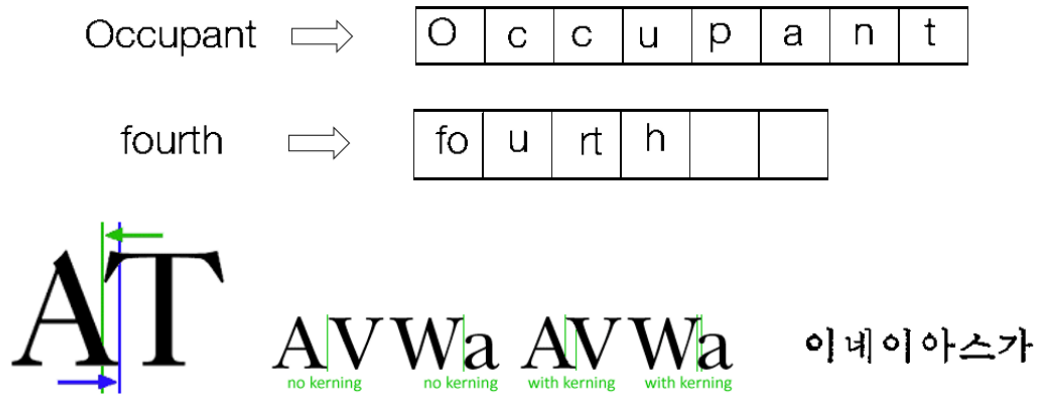
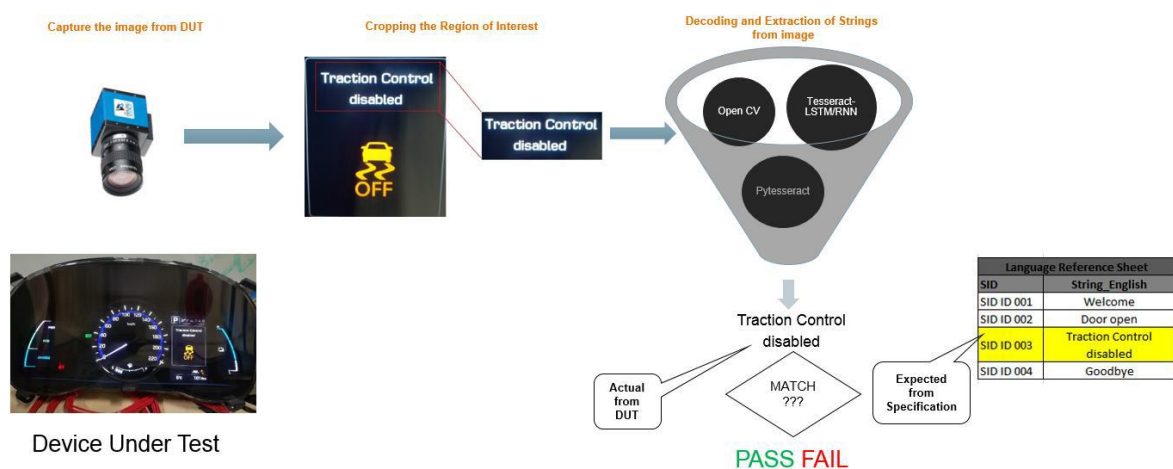


Fig. problems in character chopping – connect character, kerning, interspace in single word

Thus, by using the power of RNN/LSTM the previous learnt memory will be helpful in chopping the characters precisely to predict the words.

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture that remembers values over arbitrary intervals. Stored values are not modified as learning proceeds. An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between notable events.

With the power of LSTM Tesseract has leaped the market with error less than 10 % for all the languages. Using this ideology an application that involves processing of images and applying it to Tesseract done using python as shown in the fig.



AUTOMATED MULTILINGUAL TEXT VERIFICATION

The above Figure depicts the working method. Initially the camera will be placed on top of the cluster in an isolated environment where no external disturbance will occur. Using prove Tech the test case is driven, and respective image is captured by the camera, with fixed region of interest the text part alone will be cropped and preprocessed and further sent to Tesseract to extract string from image and gets the corresponding expected text from an excel sheet given by the customer, based on the matching the result will be stored in the excel sheet (pass/fail). In touch of a button the text verification for all the languages can be automated.

Also using OpenCV pre-processing can be done for Gray out images to test the string in it, with the correction in the first part of the process.

By using thresholding operation, the graded-out text can be converted into lighter and using Gaussian blur the pixelated portions can be smoothed up and finally fed to Tesseract and will follow the same process as above as in fig

14.3.2 Multi-Lingual Scrolling Text verification using Machine Learning algorithms:

This module utilizes the concept of panorama images that is nothing but Image key point stitching.

For this technique Features are extracted using SIFT (Scale invariant Feature Transform) algorithm, where in which unique key point in a particular image will be extracted as feature and the key points are invariant to scale as shown in the fig for vertical scrolling text and fig b for horizontal scrolling text.

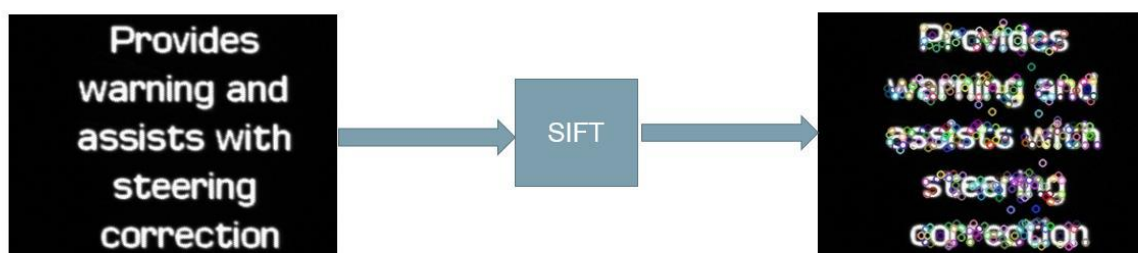


FIGURE A



FIGURE B

Once feature extraction is done, brute force matcher is used for identifying homogenous key point between two images as shown in the fig. c & d



FIGURE C



FIGURE D

And finally, when there is enough key point matched in two images it will be warped together to form a single image and likewise for all the successive images the same technique will be applied in such a way that the entire text can be addressed and a final image that has all the content in it. And the image will be fed to Tesseract OCR to extract string which is further validated with the expected string. The overall working of the concept is shown in the fig. E

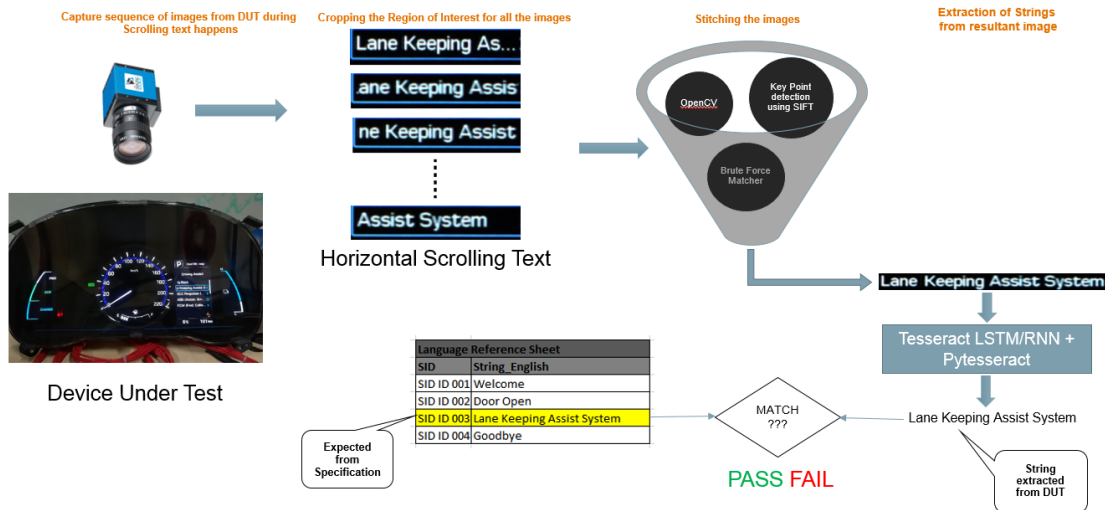


FIGURE E

Using this method all the scrolling text can be automated, as the size and speed of the scrolling is not a concern, as cameras take 24 images per second which makes it more than enough to stitch it.

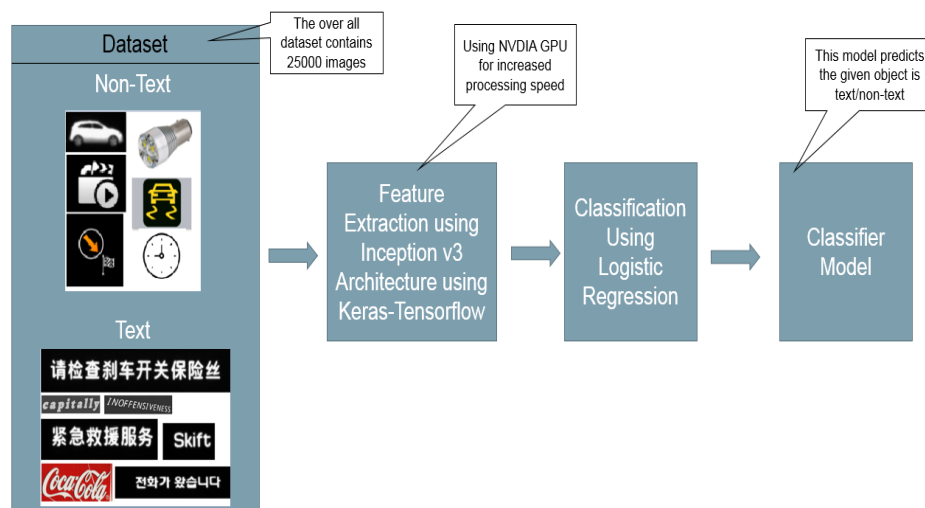
14.3.3 Auto text segmentation using Convolutional Neural Network:

Segmentation of text from real time images is an incredibly challenging part to address in the field of Artificial Intelligence. Text Segmentation is a technique used to extract text part alone in the real time images. The text segmentation is still under active research and is being implemented in self-driving cars.

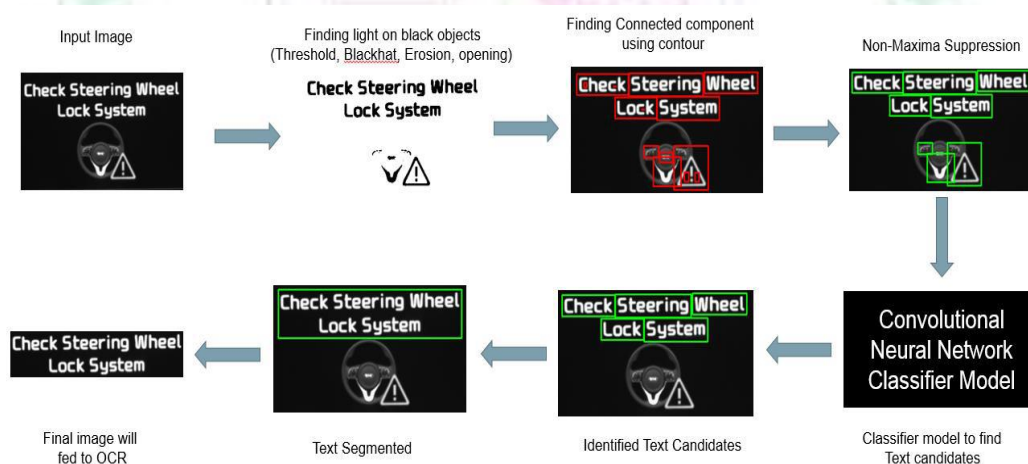
In the OCR method, we are manually measuring the co-ordinates and given as a coordinates to the image such that only texts are cropped and sent to Tesseract for OCR, this will become more time consuming if there are more number of languages as the size of text will vary for each language, in order to mitigate this effect a deep learning model is created to identify the text regions in the image such that the region of interest can be found and cropped automatically and further validated using Tesseract OCR

In this technique we are applying series of image processing technique to minimize the character candidates present in the image such that the model will be used very less for classifying the candidates belong to text or icon, the classification part is done using Inception V3 deep neural Network Architecture

So, dataset is collected that particularly defines text and icons, later the features are extracted using inception v3 using karas as frontend TensorFlow as backend and logistic regression is used as the classification technique using scikit learn from which the final classifier model is generated in the form of cPickel format, using the model the given image can be classified as text/non-text



And now the model is ready to classify the text/non-text regions, candidates of the images are cropped and send to the model, the classifier model will classify given images are text. For that series of steps are done to achieve it



14.4 VISION

The manual testing difficulties are overcome by closed loop automated testing and validation scheme using [Hardware-In-the-Loop] system integrated with vision system. The past few years several HIL techniques have been implemented

to automate instrument cluster panel testing and validation for flawless product delivery in the automotive industry. functional testing of a vehicle instrument cluster and *d-space HIL simulator* is used for rapid prototyping and model-based testing approach. Amanjot Dhaliwal et al [2] proposed HIL system for ECU unit tests and function tests on a single ECU using open-loop models and interaction between other ECUs can be simulated by generating CAN messages from the HIL system.

The machine vision system consists of a camera, lighting, optics and image processing software. A Cognex in-sight smart camera was selected for image acquisition and standalone processing which offers a resolution of 800*600 and

64MB programmable flash memory for standalone operation. The acquisition rate of the smart camera is 102 fps. The image processing software [In-Sight-Explorer] supplies a wide library vision tools for feature identification, verification, measurement and testing application. The primary source of illumination is LED (LIGHT EMITTING DIODES) ring lights with directional front lighting, which supplies a sharp contrast between the object and background. The choice of optical lens depends on the field of view and working distance. The various functionalities of ECU result have been seen by vision system such as speed gauge, Fuel level, tell-tale checking and text message read from ECU LCD. The closed loop automated machine vision system is shown in fig.

14.4.1 Device under Test:

DUT (Device Under Test) or ECU is an instrument cluster panel of AMW tractor consists of following major components, speed, Fuel, pressure gauge, telltale, warning indicator and LCD display.

14.4.2 NI-LabVIEW System:

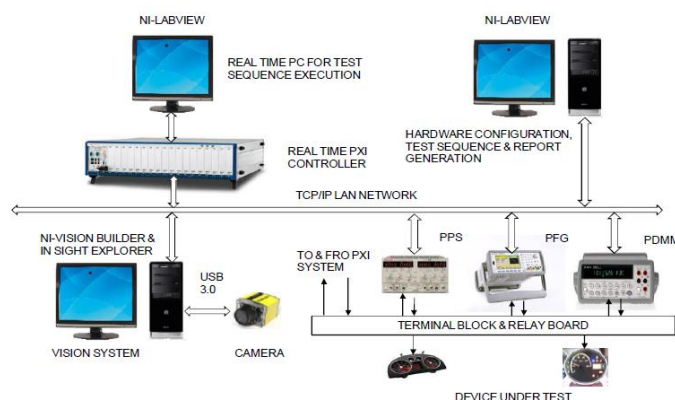
The LabVIEW system is used to configure HIL test equipment, test case execution for functionality and report generation for each variant of cluster.

14.4.3 Machine Vision System:

The vision system consists of Cognex camera and in-sight-explorer software tool. Insight- explorer has many vision tool libraries like blob and edge tool which is used for image processing algorithm implementation in closed loop ECU testing.

14.4.4 Communication Interfaces:

The TCP/IP protocol is used for communication between NI (National Instruments) LabVIEW and vision system. Also, the peripheral instruments are communicated through TCP/IP protocol with LabVIEW system.



It is vision-based hardware in the loop ECU testing. The Fig is Show the system architecture with hardware in loop test equipment with vision system.

It consists of five vital elements including machine vision system, NI-PXI HIL tester, NI-LabVIEW system, device under test and communication interfaces.

14.5 HIL SIMULATOR IMPLEMENTATION

In the HIL simulator, the peripheral modules and system controller are communicated through PCI/PCI Express bus which is available in PXI chassis. The HIL equipment such as programmable power supply and function generator is used for simulating the ECU functionality in Real time. NI-CAN card generates the CAN message for device under test which has been distributed to other ECUs in the CAN network. The DMM is used for measuring the sleep mode current passed through ECU. The machine vision system is integrated with NI-PXI system to achieve the closed loop observation for complete automated ECU testing. The PXI controller and HIL equipment are communicated through TCP/IP and USB protocol. The ECUs are interfaced through terminal block and relay board. The two types of NI-LABVIEW application tool have been developed for real time test sequence execution, hardware configuration and report generation for each variant of device under test. The image processing algorithm has been developed using insight-explorer software tool.

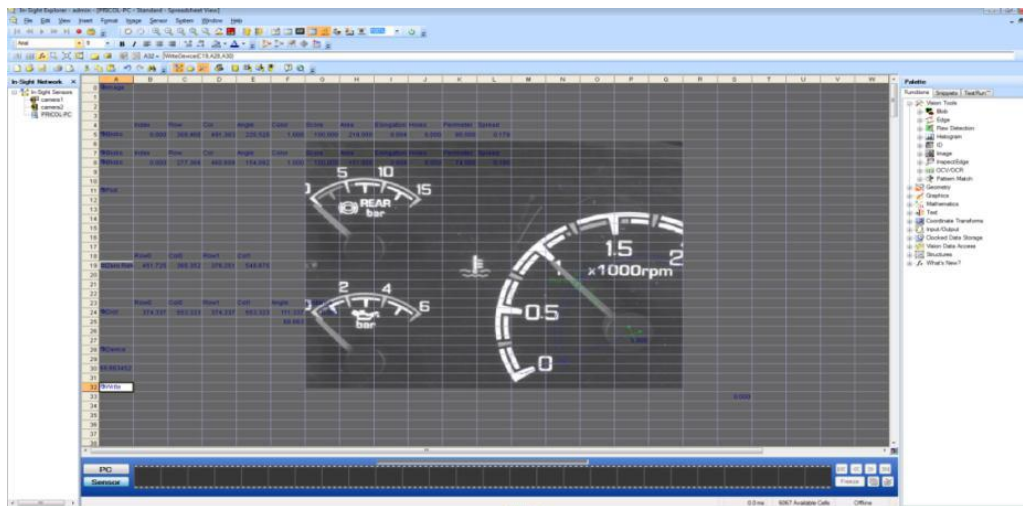
14.5.1 SPEED GAUGE DETECTION

The observation process for the speed as indicated by the needle of the speedometer is discussed in this section. When

The message for the test is received by the ECU and image of ECU is taken by the USB Camera continuously. In

captured image, two annular of region is drawn with the ends of the region coinciding with the extremes of the

speedometer and then the blob of the needle is detected using blob detection function which is available in the Insight ,Explorer as shown in fig.

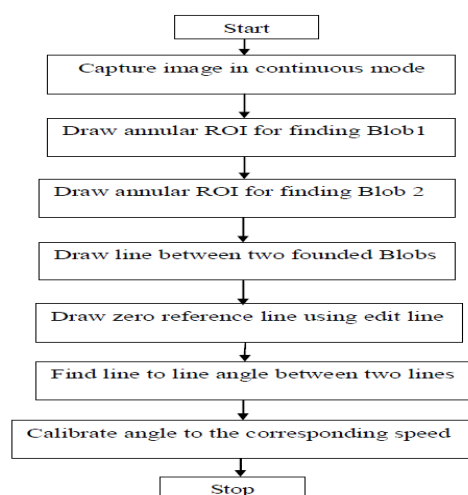


A blob is a set of connected pixels with a gray scale value above or below a specified threshold. Once two blobs have

been found in the ROI then draw a line between two blobs. Also draw zero reference line between zero point and speed

gauge pivot center. These two lines are used to find the angle for corresponding speed. This angle helps to figure out

the original speed indicating in the meter. The speed gauge detection algorithm flowchart is shown in fig



14.5.2 FUEL LEVEL DETECTION

This test decides the fuel level status from the bar graph displayed in the fuel gauge. The “*Find multi-Line*” edge tools are positioned to detect positions of 0%, 100% and actual fuel level. In machine vision terminology, an edge is defined as the boundary (a line, arc or circle) between two adjacent pixel groups with contrasting grayscale values.

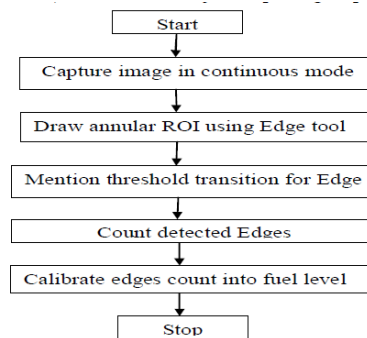


Fig. 5 Algorithm for fuel bar graph detection

The In-Sight Edge Tools are used to detect and process statistics about the found edges. Count the total edges detected by edge detection algorithm and calibrate the edge count into fuel level. The speed gauge detection algorithm flowchart

is shown in fig.

14.5.3 TELL TALE DETECTION

There are various tell-tale indicators on the ECU such as power steering, seatbelt warning, low fuel indicator etc. These light up or glow depending upon the state of hardware of the vehicle. While the state of on or off can be detected using the algorithm to detect the difference between the telltale indicators, pattern matching is needed. To run this test, a template of the indicator is loaded from the memory and learnt by the pattern max tool. The real-time input image of the cluster is then captured and loaded onto the In-Sight Explorer. The image is then scanned for the learnt pattern and an ROI is created around the pattern once it is detected. Initially, each standard symbol displayed in the cluster is trained using PatMaxTM tools and stored as expert patterns. The image captured from the unit under test is compared with the trained patterns. The symbol is recognized based on the pattern matching score returned from the PatMaxTM tools.

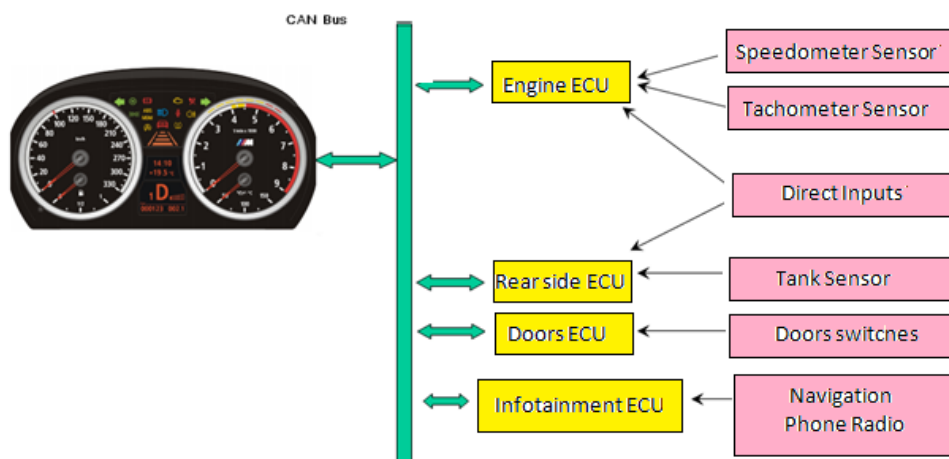
14.5.4 TEXT MESSAGE DETECTION

The warning messages displayed in the message Centre is in the need to be checked for their correctness. This test aims at recognizing and reporting these warning texts in the image processing library, OCR / OCV vision tools are designed to recognize a text string. However, this tool can only be used for identifying a text string with a known number of characters. In OCR/OCV, text fonts are trained before recognizing or verifying the text. These trained fonts are stored in a database with different instances. The ROI should be defined in the captured image area for recognizing and verification of text. The OCR/OCV compares the ROI text with database fonts, once the text is matched with database the found text has been sent to the API (Application Programming Interface) based on request.

15 Features:

15.1 IPC Functions that should be tested

As given in Fig. below the IPC collects and presents to the driver information provided by most car ECUs and sensors. Only the information that is valuable and that can be correctly interpreted by the driver is presented to him/her. The diagnostic or specific malfunction information related to major systems is stored in error memory of IPC or memory of the problem unit and can be read in service via diagnostic program.



ECUs and sensor information collected by the IPC

The IPC visual information is presented in several ways: gauges (stepper motors) with pointers, like a speedometer, a tachometer, oil, coolant and temperature meters, etc. tell tales that indicate the state or usage of different car's units and systems like the usage of lights (High/Low Beam, Turn Left/Right, etc.), state of battery, brakes, airbags, stability, etc. displays presenting main and trip odometers, board computer messages (average speed or fuel consumption) or warnings, gear box info or some errors reported by some major car ECUs, etc. The main functionality of the IPC units is discussed in the following paragraphs.

15.2 Speedometer

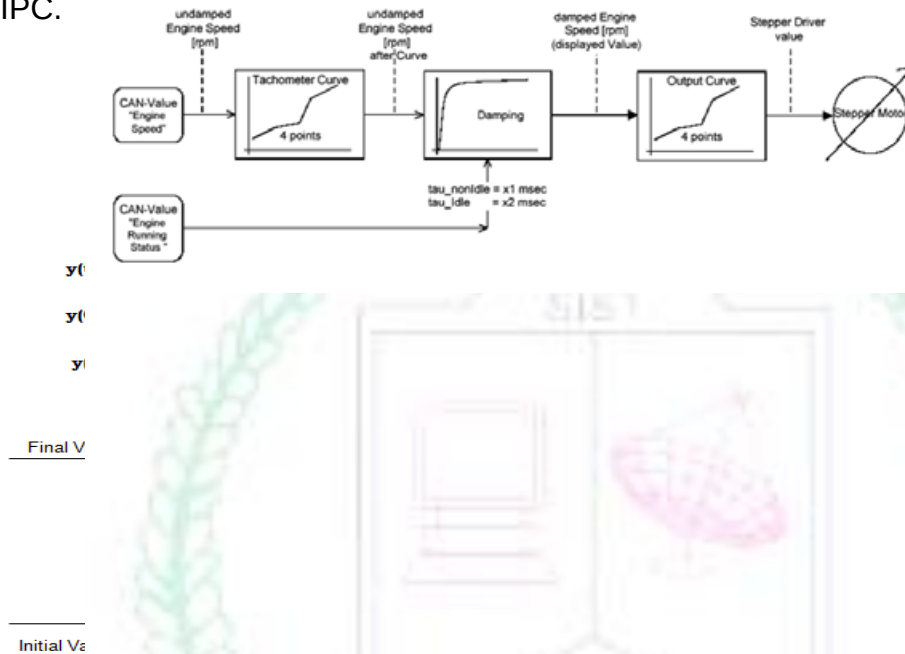
The speedometer is used to indicate the vehicle speed in km/h or mph. The instrument cluster performs the transformation from the input value (the number of impulses or revolutions per second or the value from the bus) to an angular indication reflected by several stepper motor steps (1 step = 0,05625 degrees). When the speed is received via the bus (Fig. 4), in general it is accompanied by a validity signal (quality factor). The conversion is done using characteristic curves that

have some points (4 - 6) with the correspondence between input value and output angle, between these points linear interpolation is used. The characteristic

curve is stored in EEPROM (Electrically Erasable Programmable Read Only

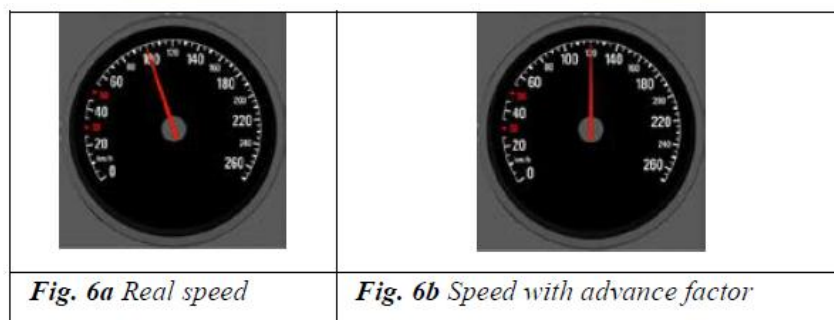
IPC.

Memory) of the To avoid pointer flickering or jumping, damping is used.



The Damping curve has a t (τ) parameter (time), and it assures that for a pointer variation, 64% of the variation range is reached after the time τ and 99% of the final value is reached after about 4-5 times

In some car clusters the advanced factor is applied. It stands for a percentage that is added to the real vehicle speed having as a result a bigger velocity value to be shown. It is done to ensure safety driving. On figures below an advance factor of 20% is added to the speed of 100 km (about 62.14 mi)/h (Fig. 6a) and for the driver it is 120 km (about 74.56 mi)/h as given on Fig. 6b



15.3 Testing the Speedometer common behavior

During the speedometer tests the following requirements related to battery connection and engine ignition must be checked:

When the battery is off the speedometer pointer stays in the same position. At reconnection, the pointer is synchronized and then it returns to the valid sign.

When the ignition is on the pointer returns to zero sign and then to the valid sign. For values in normal range, the pointer moves damped according to the characteristic curve. If the validity signal is false, then the pointer drops to zero (invalid speed signal).

After a power recovery (and in some cases after under/over voltage) the pointer returns to the zero position and then to the valid sign (synchronization).

In case the input signal is out of range, (error handling) the pointer indicates this by dropping to zero position (without damping) after a delay.

In case of under (less than 9 V)/over voltage (bigger than 16V) and power fail - the pointer stays in the same position and when the voltage is back to normal range, the pointer is synchronized (in some IPCs (Instrument Panel Cluster) it stays in the same position) and then indicates the valid value.

15.4 Tachometer

The tachometer testing is like the speedometer. Here two damps can be used. One when the engine is running and the vehicle stands still, then the damping is higher (large variations in brief time can be performed). Second damping is when the vehicle is moving - then the damping is smaller (the possible variations are

smaller). Similar common behavior tests for the speedometer must be done for tachometer.

15.5 Fuel level indication

If the signal is received over the CAN bus, the characteristic curve is like the speedometer. If the signal comes directly from the sensor, two curves are used: ADC (Analog to Digital Converter) (Analog to Digital Converter) value -> liters -> stepper motor steps. Two damping values are used (in some cases one is as in Fig. 5 and the other is linear) about vehicle moving status - when

The vehicle is stopped, the damping is lower. A bigger damping is needed for the moving vehicle to overcome the fuel movements in the tank due to accelerations and braking. When the fuel level is below a defined threshold, the fuel reserve telltale is activated.

15.6 Coolant temperature indication

The test for coolant temperature is like the fuel level indication. The behavior and the damping depend on the vehicle speed. When the vehicle is stopped (after a driving

session) the indication will not exceed the last valid value for an amount of time proportional to the driving duration because the temperature around

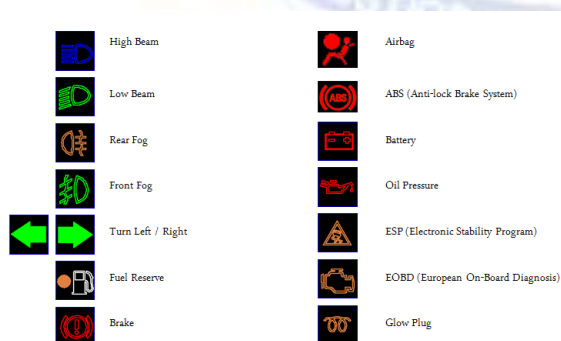
the sensor can increase (burst of heat). Some extra tests are related to normal coolant values (75-115 deg) - then the pointer stands still. In case of engine overheating (temperature > a threshold) - a warning is generated (can be a LED, an acoustic warning, a warning message on the display or a combination of them), so these warnings also must be checked.

15.7 Odometer

The main odometer is stored in IPC EEPROM area without the possibility of reset (after production); maximum displayed value is 999.999 km (about 621.37 mi). The storage is done using a multi-block storage technique - the value (km/miles) is stored in a location of more bytes (for ex. 32) and each byte is written according to an IPC specific algorithm (to avoid writing the same location each time). These EEPROM locations cannot be overwritten with a lower value. The trip odometer is stored in IPC RAM (Random Access Memory) (in some cases in EEPROM). After the maximum value is reached (999,0 or 1999,0 km (about 12421.21 mi)) a rollover to 0,0 km is performed. SCIENTIFIC PROCEEDINGS X INTERNATIONAL CONGRESS "MACHINES, TECHNOLOGIES, MATERIALS" 2013 ISSN 1310-3946

15.8 Tell Tales

The Tell-tales have specific activation/deactivation/reactivation conditions. Some of the most popular tell tales and their meaning are given in Fig. 7. In some clusters the lamps or the LEDs (LIGHT EMITTING DIODES) of Tell tales are situated in speedometer or tachometer areas and some tell tales are outside these areas. In some clusters some telltales (warnings) are presented as icons only in the main display.



15.9 Information and warning messages

More complex instrument clusters having a big dot matrix display (in some cases color) show more information that alerts the driver about the state of several car functionalities (tire pressure, engine status, ABS (Antilock Brake System), ASR (Anti

Spin Regulation), ESP, seat belt, gear information, ice warning, navigation, telephone, radio/cd etc.).

A text message is displayed, and it can be doubled by a symbol or / and an acoustic chime too. The messages can be presented in different languages

(Configurable from menu or EEPROM). When more warnings/messages should be displayed at the same time, they are presented in rolling mode: each message

(With a symbol if the case) is shown for an amount of time, then the next one.

The messages have priorities assigned. Project specific procedures are defined to treat the priorities - a message cannot/cannot be interrupted by a higher priority one, a message has a minimum display time. Next are given some added messages which can appear on display, and which are object of specific tests:

- fuel level
- coolant temperature
- outside temperature
- warning messages
- oil level
- service interval indication
clock, date
- gear box position and mode
- distronic / tempomat
- navigation
- telephone.

16 EEPROM

The EEPROM memory holds some configuration parameters which are the object of tests. It contains info about the cluster itself model, version, production day, month, year, similar data about the engine and most important ECUs in the car, odometer data, characteristic curves, damping parameter tau, type of engine (gasoline, diesel, hybrid), measurement units km, miles, error codes and lots of other data. Except the software tests related to the functionality of mentioned above IPC units and elements mentioned above, special attention must be given to IPC Power Management tests and Illumination tests described below. Power Management tests are related to normal voltage range is which is between 9V and 16 V, under voltage (below 9V), over voltage (above 16V) and power fail (below 6V). In normal voltage range the instrument cluster has normal functionality. The test voltage is around 13.5V. Outside this range, except in the case of power failure we can have the following situation:

Pointers are frozen to the actual position, the microprocessor-controlled LED's (lamps) are deactivated and the write function for EEPROM is deactivated. Data retention in RAM is assured. When the voltage resumes to normal range (with hysteresis) the normal functionality is resumed. The pointers are synchronized (return to zero position and then to the valid indication) to avoid step losses. During power failure, added to the under-voltage situation, the RAM is no longer kept. When the voltage resumes to normal range, the instrument cluster is reset.

The background illumination is related to the displays and scales and pointers. They can be illuminated separately. The illumination is different according to two situations: - daytime, when only the display is illuminated with small brightness and nighttime when display(s), scales and pointers are illuminated. During night

time illumination the brightness can be modified manually or automatically. The day/night transition is done when the front lights are activated (manual mode) or when the outside illumination level is below a defined threshold (auto mode). The illumination levels are stored in EEPROM, and the selected level is memorized as a

pointer in this illumination table. The display and the scales + pointers can have different illumination tables to have different illuminations. Manual dimming is performed using one cluster button or a function from the menu. Automatic dimming is performed using a sensor that decides the transition from day to night illumination level. Illumination dimming does not influence the telltales' brightness. Illumination can be used to signal a warning. For example, in some cars the tachometer scale illumination is activated at full brightness (while the other scales' illuminations are deactivated) when the engine speed exceeds a defined threshold - to signalize that the engine can be damaged.

17 Testcase:

A test case is exactly what it sounds like a test scenario measuring functionality across a set of actions or conditions to verify the expected result. They apply to any software application, can use manual testing or an automated test, and can make use of test case management tools.

A key thing to remember when it comes to writing test cases is that they are intended to test a basic variable or task such as whether a discount code applies to the right product on an e-commerce web page. This allows a software tester more flexibility in how to test code and features.

17.1 Test Script vs. Test Case:

The difference between test cases vs. test scripts should also be clarified. A test script is a short program meant to test certain functionality. A test case is a document with steps to be completed as planned out ahead of time.

17.2 Test cases have a few integral parts that should always be present in fields. However, every test case can be broken down into 8 basic steps:

Step 1: Test Case ID

Test cases should all bear unique IDs to represent them. In most cases, following a convention for this naming ID helps with organization, clarity, and understanding.

Step 2: Test Description

This description should detail what unit, feature, or function is being tested or what is being verified.

Step 3: Assumptions and Pre-Conditions

This entails any conditions to be met before test case execution. One example would be requiring a valid Outlook account for a login.

Step 4: Test Data

This relates to the variables and their values in the test case. In the example of an email login, it would be the username and password for the account.

Step 5: Steps to be Executed

These should be easily repeatable steps as executed from the end user's perspective. For instance, a test case for logging into an email server might include these steps:

1. Open email server web page.
2. Enter username.
3. Enter password.
4. Click "Enter" or "Login" button.

Step 6: Expected Result

This indicates the result expected after the test case step execution. Upon entering the right login information, the expected result would be a successful login.

Step 7: Actual Result and Post-Conditions

As compared to the expected result, we can figure out the status of the test case. In the case of email login, the user would either be successfully logged in or not. The

post-condition is what happens because of the step execution such as being redirected to the email inbox.

Step 8: Pass/Fail

Figuring out the pass/fail status depends on how the expected result and the actual result compare to each other.

Same result = Pass Different results = Fail

18 Difference Between Manual Testing and Automation Testing:

Parameter	Automation Testing	Manual Testing
Definition	Automation Testing uses automation tools to execute test cases.	In manual testing, test cases are executed by a human tester and software.
Processing time	Automated testing is significantly faster than a manual approach.	Manual testing is time-consuming and takes up human resources.
Exploratory Testing	Automation does not allow random testing	Exploratory testing is possible in Manual Testing
Initial investment	The initial investment in the automated testing is higher. Though the ROI is better in the long run.	The initial investment in the Manual testing is comparatively lower. ROI is lower compared to Automation testing in the long run.
Reliability	Automated testing is a reliable method, as it is performed by tools and scripts. There is no testing Fatigue.	Manual testing is not as accurate because of the possibility of the human errors.
Investment	Investment is required for testing tools as well as automation engineers	Investment is needed for human resources.
Cost-effective	Not cost effective for low volume regression	Not cost effective for high volume regression.
Test Report Visibility	With automation testing, all stakeholders can login into the automation system and check test execution results	Manual Tests are usually recorded in an Excel or Word, and test results are not readily/ readily available.
Human observation	Automated testing does not involve human consideration. So, it can never give assurance of user-	The manual testing method allows human observation, which may be useful to offer

Parameter	Automation Testing	Manual Testing
	friendliness and positive customer experience.	user-friendly system.
Performance Testing	Performance Tests like Load Testing, Stress Testing, Spike Testing, etc. must be tested by an automation tool compulsorily.	Performance Testing is not feasible manually
Parallel Execution	This testing can be executed on different operating platforms in parallel and reduce test execution time.	Manual tests can be executed in parallel but would need to increase your human resource which is expensive
Batch testing	You can Batch multiple Test Scripts for nightly execution.	Manual tests cannot be batched.
Programming knowledge	Programming knowledge is necessary in automation testing.	No need for programming in Manual Testing.
Set up	Automation test requires less complex test execution set up.	Manual testing needs have a more straightforward test execution setup
Engagement	Done by tools. Its accurate and never gets bored!	Repetitive Manual Test Execution can get boring and error prone.
Ideal approach	Automation testing is useful when often executing the same set of test cases	Manual testing proves useful when the test case only needs to run once or twice.
Build Verification Testing	Automation testing is useful for Build Verification Testing (BVT).	Executing the Build Verification Testing (BVT) is exceedingly difficult and time-consuming in manual testing.
Deadlines	Automated Tests have zero risks of missing out a pre-decided test.	Manual Testing has a higher risk of missing out the pre-decided test deadline.
Framework	Automation testing uses frameworks like Data Drive, Keyword, Hybrid to accelerate the automation process.	Manual Testing does not use frameworks but may use guidelines, checklists, stringent processes to draft certain test cases.
Documentation	Automated Tests acts as a document provides training value especially for automated unit test cases. A new developer can investigate a unit test	Manual Test cases provide no training value

Parameter	Automation Testing	Manual Testing
	case and understand the code base quickly.	
Test Design	Automated Unit Tests enforce/drive Test Driven Development Design.	Manual Unit Tests do not drive design into the coding process

19 Reasons for Choosing Automation in Validation:

Validation Task	Previous Method effort Hours	Current Method Effort Hours
Text verification in all the 23 languages	90	30
Scrolling text verification	24	5
Text Segmentation	5	1
OCR Verification	15	6

Thus, using the above-mentioned techniques, end to end automation is achieved for string verification. We have used this testing tool in Hyundai Ionic Instrument Cluster which has 23 languages with 20,000 strings in it. Using these techniques, the validation was done at ease for all the sprints which reduced the effort hours used and increase the accuracy that obviously decreased the slip through rate to the customer. The below table Compares the effort hours reduction with the current and previous tools.

The technique can be applied for USM functionality (User Setting Menu) as there will be no feedback of where the focus bar is after some combination of key presses.

Also, it can be applied for Clock time verification, ODO and other number verification too.

20 Testing High Beam (tell-tale):

Sl.no	Test case	condition	expected	result
-------	-----------	-----------	----------	--------

1	Ignition OFF	IGN= 0	Lamp off	Missing the telltale
2	Ignition ON	IGN = 1	Lamp on	Show telltale
3	Ignition on-off	Switch signal 0 to 1 and 1 to 0	Lamp on and off	Show and miss telltale in 2000ms (about 2 seconds) time gap
4	Miss the signal	Can = time out	Lamp on until 1500 ms	Show telltale for 1500ms (about 1 and a half seconds)
5	Missing signal	Can < timed out	Lamp should not go OFF	Show telltale
6	Missing signal	Can > time out	Lamp should behave as usual	Show telltale until Can time given
7	Under voltage	Voltage < 6.5	Cluster Off state	blank

21 Results and Conclusion:

21.1 Output for Manual Verification for High Beam:

The below Video shows how manually the Output is verified



video_2022-04-16_13
-36-07.mp4

21.2 Output for Automation testing of High Beam:

Here when in the above fig when the ignition is turned on there should be a change in cluster and the High beam should turn on, but it's still turned OFF this is considered as an issue and should be resolved later before handling to a customer

TT_HL01001_MAT_Off_Run - A13_PE_01.07.01 @ INDF47NH32; 12/25/2021 03:37:22 AM

Comment Test Protocol Runtime Errors Parameters History

<Main protocol>

ID / Time	Reac	Duration	ObjStatus / Frame	Object Name	Expectation	State
14	500	500		101:High_Beam//	High_Beam	ON
00.472			-1-04d_10h_45m_14.386-img-6026020-.jpg	High_Beam	[State]:ON	OFF
00.512			-1-04d_10h_45m_14.426-img-6026021-.jpg	High_Beam	[State]:ON	OFF
00.992			-1-04d_10h_45m_14.906-img-6026033-.jpg	High_Beam	[State]:ON	OFF
NCT = 04d_10h_45m_14.914 [NCT + 1000]						
Spent Time for Sound Analyze: 0.0 sec						
154.656 Test NOK						
154.656 MCO analysis failed						
18			154.672 [LAT:04d_10h_45m_51.841]			
154.672 Setting Telltale High_Beam to OFF						
19			154.672 Setting CAN Input pCF_Gway_HeadLampHigh = 0			
20			154.719 [LAT:04d_10h_45m_51.894]			
154.719 Store LAT: TRIG = 04d_10h_45m_51.894						
154.719 [LAT:04d_10h_45m_51.895]						
154.719 Store LAT: TRIGGER = 04d_10h_45m_51.895						
154.735 Verify High_Beam (High Beam) is OFF						
185.891						
NCT = 04d_10h_45m_51.895						
14	300	500		101:High_Beam//	High_Beam	OFF
00.011	11		-1-04d_10h_45m_51.906-img-6026958-.jpg	High_Beam	[State]:OFF	OFF
00.491	480		-1-04d_10h_45m_52.386-img-6026970-.jpg	High_Beam	[State]:OFF	OFF
NCT = 04d_10h_45m_52.386 [NCT + 11 + 480]						
Spent Time: 29.548 sec (Vision: 29.522 sec) Spent Time For Sound Analyze: 0.0 sec						
22			185.891 [LAT:04d_10h_46m_23.070]			
185.891 Enabling transmission of CAN Message CAN1_IC.CLU.CGW1.CF_Gway_HeadLampHigh						
23			185.891 Setting Telltale High_Beam to OFF			
24			185.906 Setting CAN Input pCF_Gway_HeadLampHigh = 0			
25			185.938 [LAT:04d_10h_46m_23.114]			
185.938 Store LAT: TRIG = 04d_10h_46m_23.114						
185.938 [LAT:04d_10h_46m_23.115]						
185.938 Store LAT: TRIGGER = 04d_10h_46m_23.115						

OK Cancel Apply

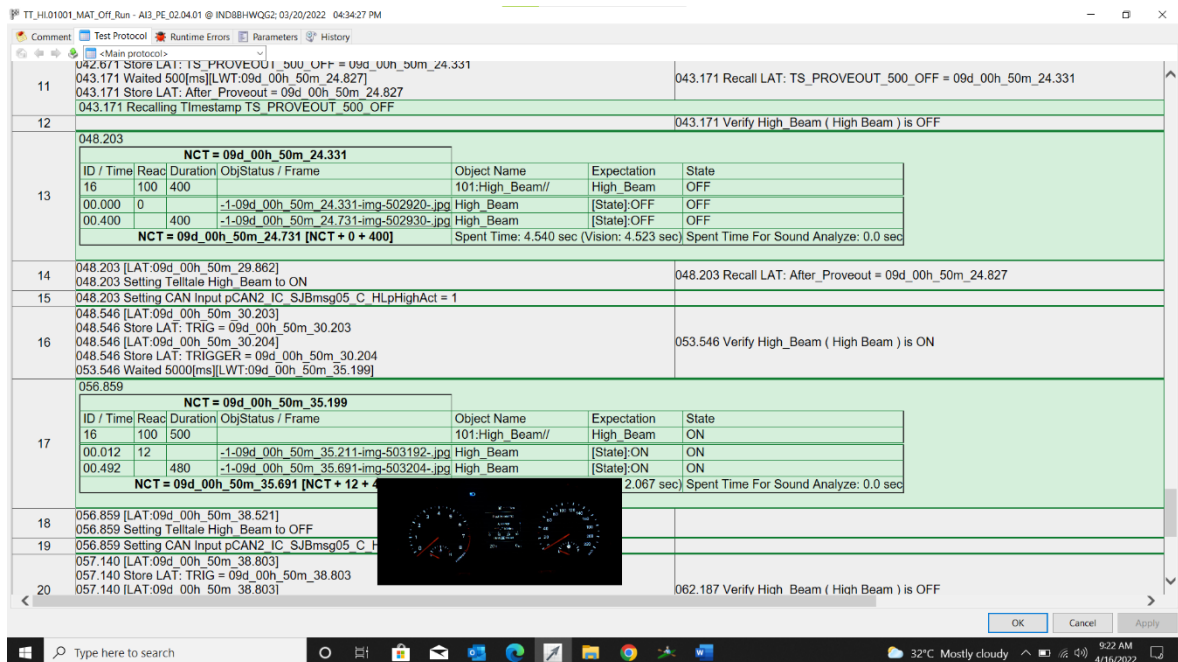
Type here to search

32°C Mostly cloudy 9:25 AM 4/16/2022

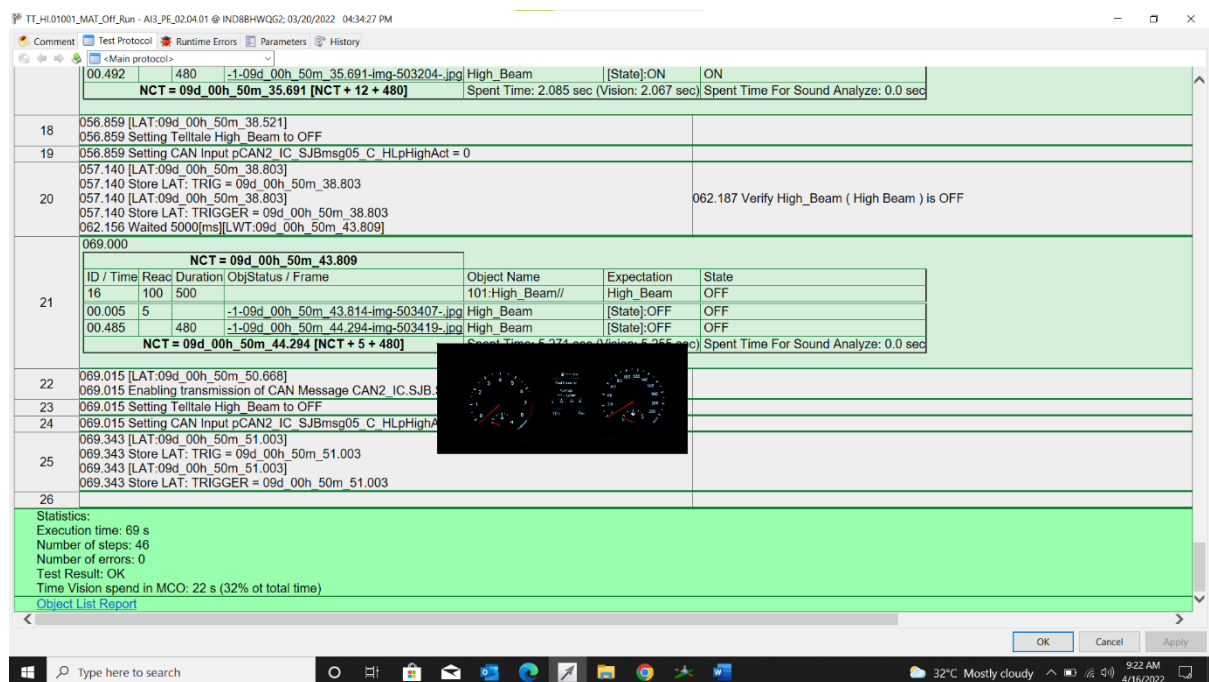
HIGH-BEAM TELL-TALE FAILED CONDITION



Now after resolving the issue/Defect output of the tell-tale will be shown as shown in below figures High-beam Tell-Tale passed condition figures 1&2



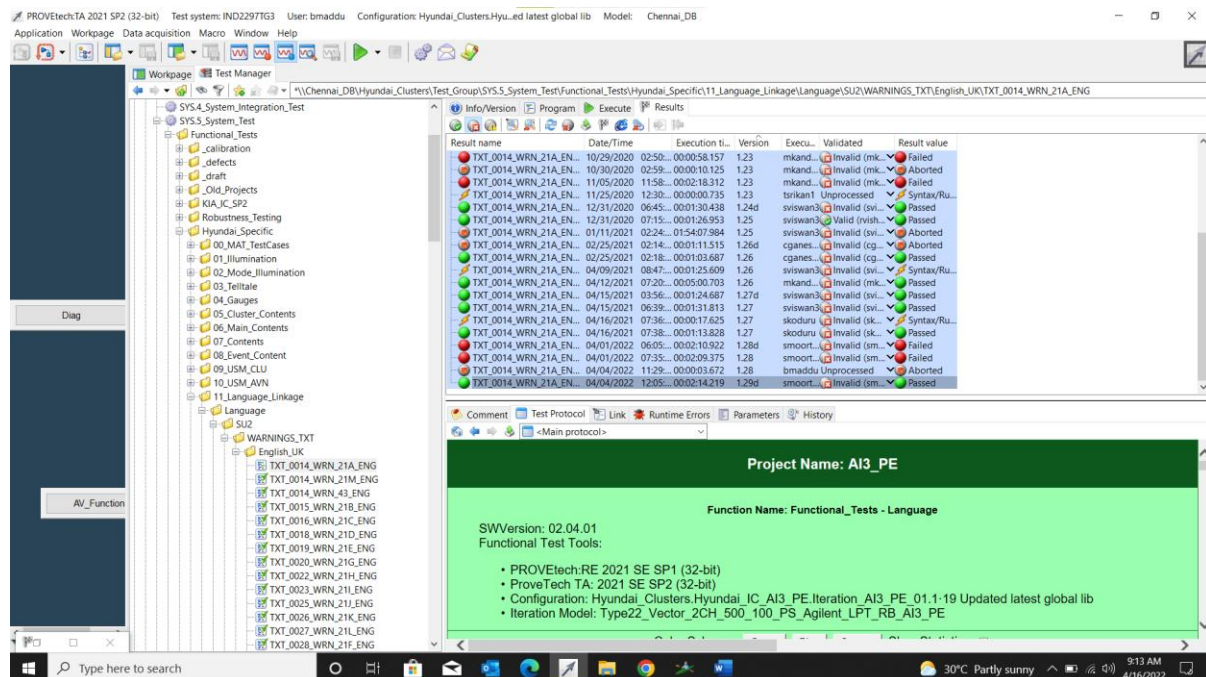
HIGH-BEAM TELL-TALE PASSED CONDITION FIGURE 1



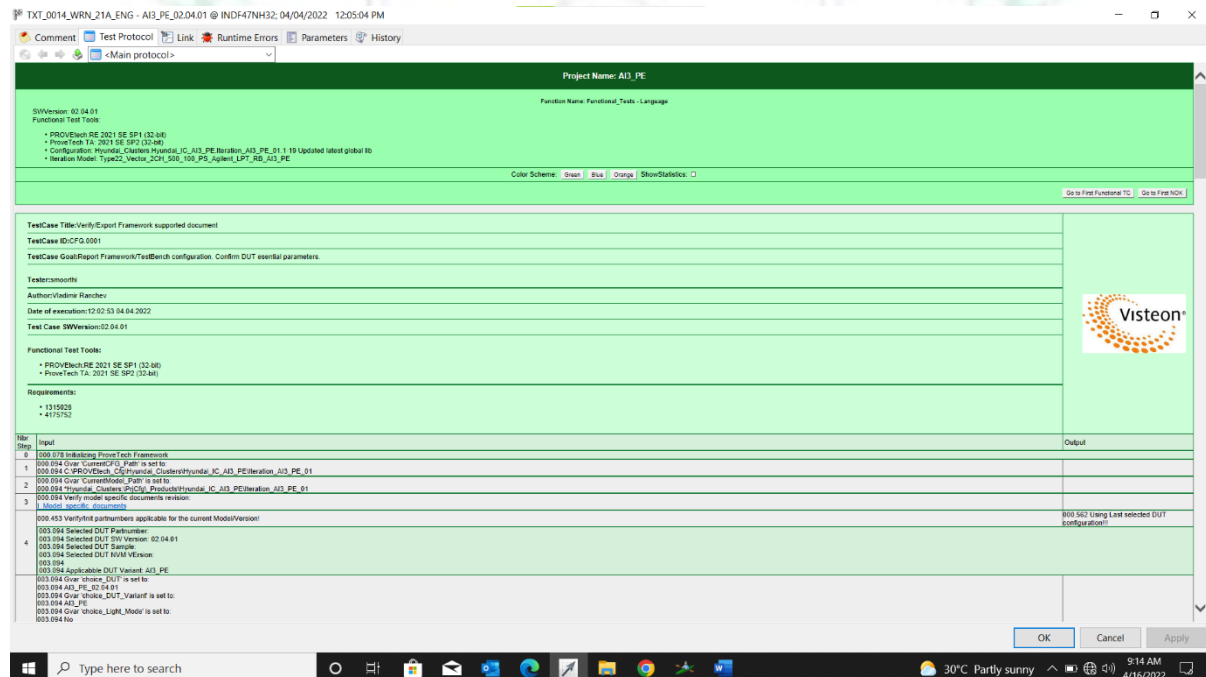
HIGH-BEAM TELL-TALE PASSED CONDITION FIGURE 2

Similarly, the outputs for checking texts through Automation:

Step 1: open results terminal

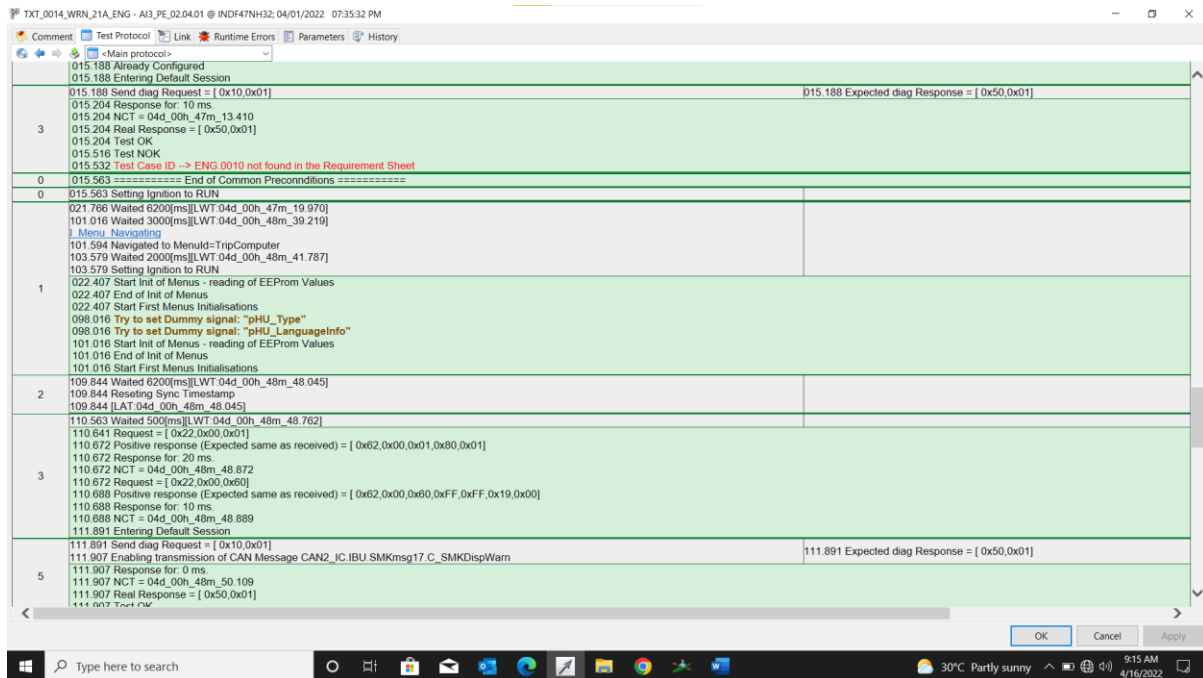


Step 2: if any failed test cases click on go to first NOK



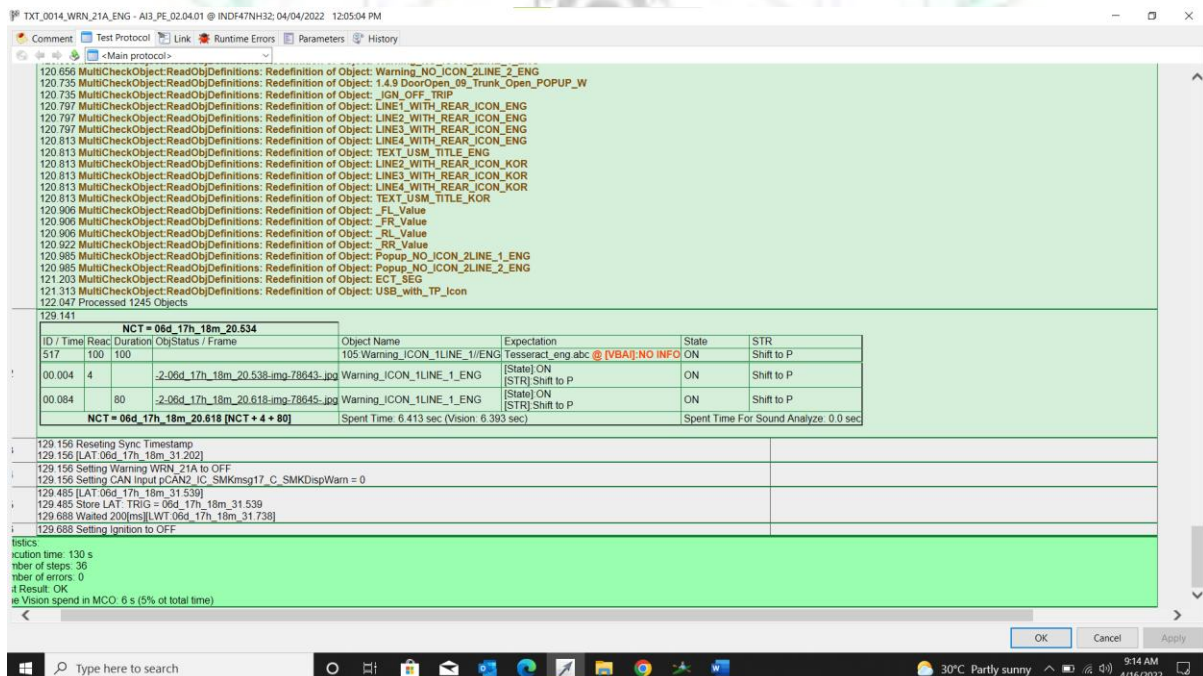
Step 3:

Verify the reasons for respective errors if not make it as an issue



Step 4:

after correcting the error run again and verify the expected and state O/P is matched



22 Reference :

- Instrumentation cluster : <https://www.mouser.in/applications/automotive-instrument/>
- Automation : <https://www.semanticscholar.org/paper/Automation-for-Instrument-Cluster-Panel-Testing-and-Sharmila-Venkadesan/c215a2f8694ae08fb2b8744628a402bad0a6671a>
- Vision : <https://www.ni.com/en-us/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-vision-builder-for-automated-inspection.html>
- Models : <https://productivehut.com/v-model-agile-waterfall-spiral/>
- Levels of testing : <https://www.guru99.com/levels-of-testing.html>
- UDS : https://en.wikipedia.org/wiki/Unified_Diagnostic_Services
- HMI screen automation : <https://ieeexplore.ieee.org/document/8249425/authors#authors>
- Automation using vision : <https://www.sae.org/publications/technical-papers/content/2016-36-0235/>
- Multilingual string verification : <https://visteon.com/wp-content/uploads/2019/01/multilingual-string-verification-for-automotive-instrument-cluster-using-artificial-intelligence.pdf>