# DESIGN AND IMPLEMENTATION A FRAMEWORK FOR AUTOMATED TESTED OF EMBEDDED SOFTWARE

A Thesis

Submitted in partial fulfilment of the requirements for the award of the degree of

**DOCTOR OF PHILOSOPHY**

**in**

**Electronics and Communication Engineering**

**By**

**Srinivas Perala**

**41800017**

**Supervised By**

**Dr. Ajay Roy**

**LOVELY PROFESSIONAL UNIVERSITY**
**PUNJAB**
**2022**

# DECLARATION

I hereby declare that the thesis entitled "Design and Implementation A Framework For Automated Tested Of Embedded Software" is submitted by me for a Doctor of Philosophy in Electronics and Communication Engineering. My original and independent research work was carried out under Dr Ajay Roy, Professor, Lovely Professional University, Punjab. It has not been submitted for the award of any degree, associateship, the fellowship of any University or Institution.

I further declare that there is no falsification or manipulation in terms of research materials, equipment or processes, experiments, methods, models, modelling, data, data analysis, results, or theoretical work.

I have checked the thesis using Turnitin for ensuring that there is plagiarising as per university norms in my thesis and wherever any copyrighted material has been included, the same has been duly acknowledged.

[illegible]

SRINIVAS PERALA
School of Electronics and Electrical Engineering
Lovely Faculty of Technology and Sciences
Lovely Professional University
Phagwara, -144411 (Punjab)

Date: 18-05-2022

We endorse the above declaration of Ph.D. student

Signature of the Guide

## ACKNOWLEDGEMENT

I am thankful to the almighty for making things possible at the right time. I owe my success to my supervisor and sincerely thank **Dr. Ajay Roy** for his guidance. I greatly appreciate his support, positive attitude, and his vast knowledge of a wide range of topics. His guidance in giving ideas and solving research problems and giving me the freedom to research my way has proved valuable and invaluable. I am deeply influenced by my supervisor's guidance and sincerely thankful for standing by my side in tough times.

I want to thank my parents, my wife Anusha and my daughter Srinisha, without whose encouragement this undertaking would not have been possible. I would like to thank my family, friends, and colleagues for their continuous love and support. Finally, special thanks to all the people who helped me, directly and indirectly, accomplish this work.

I like to special thank **The honourable chancellor Mr Ashok Mittal** for transforming India by transforming education.

[illegible]

Date: 18-05-2022 Srinivas Perala

## CERTIFICATE

This thesis entitled " Design and Implementation A Framework For Automated Tested Of Embedded Software", submitted by Srinivas Perala of Lovely Professional University, is a record of bonafide research work done by her, and it has not been submitted for the award of any degree, diploma, associateship, the fellowship of any University/Institution.

Place: Phagwara

Date: 18-05-2022

[illegible]

Signature of the Guide

Dr Ajay Roy
Professor
School of Electronics and Electrical Engineering
Lovely Faculty of Technology and Sciences
Lovely Professional University,
Phagwara, Kapurthala-144411 (Punjab)

# ABSTRACT

Every Single embedded product has defects and finding flaws in the development process and resolving them prior to the product launch is significant. Verification and validation find the bugs in the software and report to the software developer to resolve reported bugs. Most of the time, test engineers don’t have sufficient time to complete all test cases to meet the product release deadlines. Test automation is the solution to this big problem. In this research work, we concentrated on automotive and home appliances embedded software test automation. Test automation is the best solution to achieve the deadline of the product release. Currently, Industries are using test management tools like LabVIEW, Test stand and Automation desk. These tools require expertise and a lot of manual configurations which increase the test automation time. This research work provides a novel solution to automate those manual efforts. One of these objectives generates test scripts from test cases to reduce manual efforts, time, and cost. Test automation is one of the essential aspects of the verification and validation of embedded software. Test automation plays a crucial role to meet the deadlines to launch a product. In terms of embedded software, display (graphical) validation is a bit complex to automate than text or numeric testing, where references are available for comparisons. In the second objective work, we addressed comparing digital images to find the similarity to facilitate the embedded software test automation. Embedded testing automation is widely used to reduce development time and cost. However, we cannot achieve 100% test automation due to dependency on a few manual observation aspects like the display output. Most of the embedded products have output displays that produce text, symbols or images and their testing is conducted manually. The manual observation involved can be automated with the presented work to reduce significant testing time and reduce paradox errors. In product development, stakeholders and top management summarise the concept and document the requirements in natural language. These ideas and descriptions are documented as software requirements by the technical department. Developers develop software following this software requirement document. Testing this developed software derives test cases from natural language requirements and then does the testing method to locate the bugs. The Developers and Testers develop the test cases by understanding the requirements. Due to increasing the advanced features, deriving the test cases is monotonous and takes more time. This third research objective shows a method to automate this process, deriving test cases from NLP algorithms' requirements. This helpful approach to reduce the time and cost of software development.

**Keywords:** Test Automation, LabVIEW, Test stand, Automation Desk, Machine learning, NLP, TensorFlow, Hardware-in-the-loo (HIL)

## List of Publications

1. Perala, S., Roy, A. "A novel framework design for test case authoring and auto test scripts generation", Turkish Journal of Computer and Mathematics Education, 2021, 12(6), pp. 1479,1487, ISSN 1309-4653

2. Perala, S., Roy, A. "A review on test automation for test cases generation using NLP techniques", Turkish Journal of Computer and Mathematics Education, 2021, 12(6), pp. 1488.1491, ISSN 1309-4653.

3. Srinivas Perala, Dr Ajay Roy and Koushik "A Novel Method of Test Automation for Testing Embedded Software" Think India Journal, Vol-22-Issue-37-December -2019, ISSN 0971-1260.

4. Srinivas Perala, Dr Ajay Roy and Dr Sandeep Ranjan "Image Processing Algorithm to Compare Test image with Reference Image to Validate Embedded Software of Display Application" Conference world, Jan-2021, ISBN 97881-948668-6-2.

5. Perala, S., Roy, A "Embedded Vision Software Test Automation using LabVIEW OCR" ICSCN 2021 Springer, 3rd International Conference on
Sustainable Communication Networks and Application ISSN: 2367-4512.

6. Perala, S., Roy, A. "**Implementing Hardware-in-the-Loop Test automation on AWS**", ACM Journal on Emerging Technologies in Computing Systems, SCI Journal, ISSN:1550-4832. (Communicated)

# TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBRIVATIONS

| | |
|---|---|
| NI | National Instruments |
| LabVIEW | Laboratory Virtual instrumentation engineering workbench. |
| HIL | Hardware-in-the-loop |
| ML | Machine Learning |
| AI | Artificial intelligence |
| NLP | Natural Language processing |

# Chapter 1 INTRODUCTION

This thesis focuses on test automation procedures and a novel framework design to reduce manual efforts in test automation. It also provided a novel solution to automate embedded vision software.

## 1.1 THE TESTING BACKGROUNDS

This research aims to fulfil the research gap between existing techniques of test automation and advanced machine learning test automation. Embedded System Product OEMs are spending massive money on product verification and validation. Manual testing takes enormous time and money. Embedded software validation is to validate that a product meets initial requirements, specifications, and regulations.

Embedded systems-based devices are a combination of software and hardware. Software controls the components based on sensors inputs and functionality. This software, called embedded software, is downloaded into the microcontroller, which controls all device parts. Automotive, Medical, and Home Appliances related embedded devices directly affect a human if embedded software fails.

Consider one automotive example: We call microcontroller ECU (Electronic Control Unit) in the automobile. Modern cars can have up to 60 plus ECUs for various functions such as Engine management system, chassis control, Power train and comfort systems. if any ECU software fails will affect human life directly. Proper validation and verification are essential before any product release. The embedded software is tested for its performance, reliability and validated as per the defined standard software requirements.

Embedded Software testing inspect the software in terms of all requirements and ensure the quality that meets all the requirements. Embedded software testing is a standard method to ensure safety in critical products like medical, railways, aviation and automobile etc. More reliable testing is vital to award software safety accreditation.

## 1.2 TEST MANAGEMENT SOFTWARE'S

There are few tests management software available which minimise the manual testing efforts. For example, d Space automation desk and the National instruments test stand.

Automation Desk is an intelligent test automation tool for hardware-in-the-loop (HIL) testing ECUs Software. Automation Desk customers have to develop scripts using python programming.

The ASAM standards allow reusing of the automated test scripts across various development phases. Developing scripts for this tool required a deep understanding of system knowledge and programming skills. Recruiting highly knowledgeable people is a big challenge and cost-effective for the company.

Another test management tool, which is a National Instrument's Test Stand. Test Stand is powerful test management software that is developed to rapidly develop automated test sequences and validation systems. Test stand used to develop, execute, and deploy test system software. We can expand the functionality of the system by developing test sequences in Test Stand that combine code modules developed in any programming language. Here also we need expert test engineers which is cost-effective to the company. Also, it consumes more time to develop test sequences. My research gives a solution to these industry challenges.

### Powerful algorithms to validate embedded image software

In today world, most of the products have image display units. Based on software or hardware triggering, the display unit shows different images. Manual observation is the only way to test these scenarios on the current trend. However, this manual observation testing takes enormous time, and paradox errors may occur. In a current scenario, the validation of digital images is considered manual regression due to the lack of a reliable comparison algorithm. In many cases, this can be highly repetitive which leads to big challenge to test engineers. The chance for bug leakage is higher, as testers tend to check such features less often and with less specific test cases. Hence, it is imperative to try to include visual tests into automation regressions as much as feasible. This research focuses on this industry requirement to automate image testing.

**The Hardware-in-the-loop (HIL) :**

Hardware-in-the-loop simulation is a method for validating software logic algorithms, running on ECU, by creating a real-time virtual system that represents your physical system. HIL supports testing the behaviour of software without a physical unit. HIL tests support validating embedded software on ECUs using simulation and modelling methods to reduce the test times and improve the requirements coverage, especially for test cases that are hard to replicate in physical lab/track/field testing reliably. HIL testing is necessary to ensure the reliability of fast-developing Electric vehicles and Advanced driving assistance systems (ADAS) Safety systems. HIL is vital for the increasing connectivity and interconnection between systems and automotive domains as a test methodology as they jointly contribute to crucial vehicle attributes.

HIL model is a real-time simulation. HIL model using to test controller design. HIL simulation demonstrates how the controller responds in real-time to the realistic computer-generated stimulator. Also, use HIL to find out if the physical system model is valid.

In the HIL model, use a real-time computer to represent a plant model and a real controller version. Figure 2 illustrates a HIL simulation setup. The real-time computer comprises the real-time capable model of the controller and plant. The enhancement hardware also comprises an interface with which to control the simulated input to the plant. The software flashed in the controller generates from the device model. The real-time computer contains information for the physical system generated from the plant model.

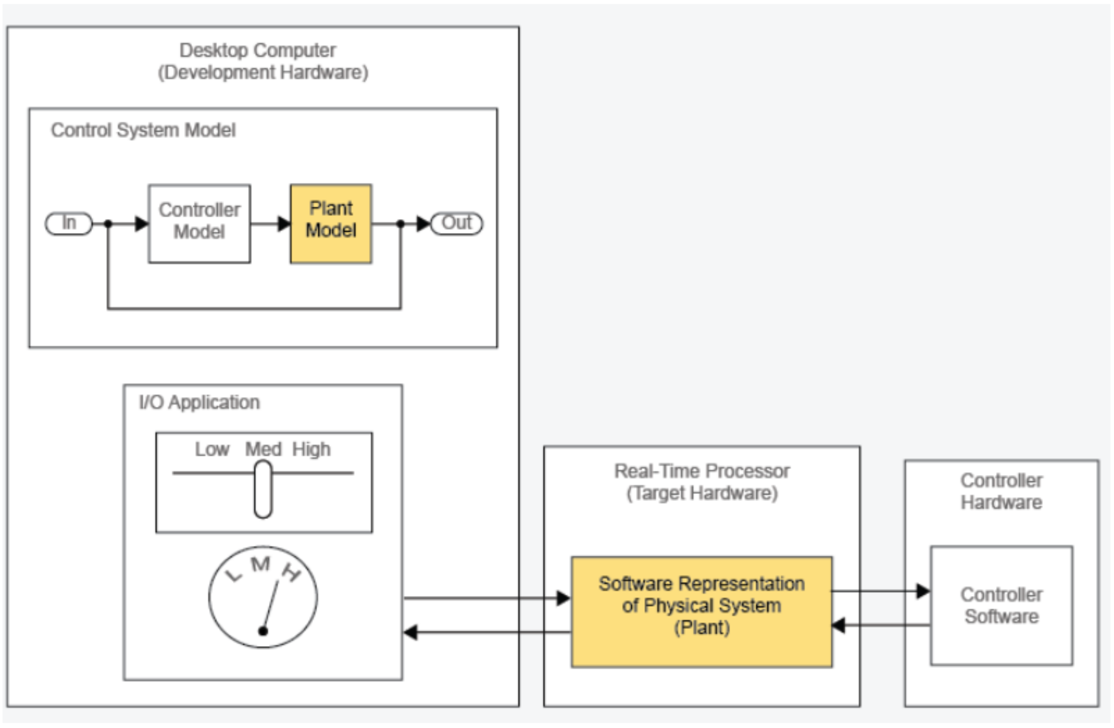


Figure 1. HIL block diagram

**HIL simulation work process:**

1. Build and replicate a computer-generated real-time application of physical elements such as plants, sensors, and actuators on a real-time target computer.
2. Compile the software control algorithms on an embedded controller and operate the plant or model in real-time on a target computer connected to the controller. The embedded controller communicates with the plant model is simulated through various input/output channels.
3. Enhance software depictions of modules and progressively substituted parts of the system environment with the real hardware components.

With this method, HIL simulation can reduce the expensive duplications in hardware manufacturing.

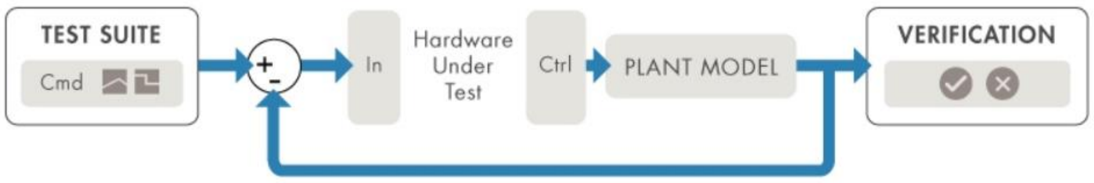


Figure 2. Block diagram of a HIL simulation

HIL simulation is helpful when you are testing control algorithms on a straightforward physical system that is expensive or hazardous. HIL simulation is used extensively in the home appliances, automotive, medical, aerospace, defence, industrial automation and machinery industries to test drive embedded designs.

Hardware-in-Loop testing provides the automotive and other industries to test their solutions before the actual target platform is equipped. Not only does this ensure reduced time-to-market but also leads to a better and more technically matured product.

The probability of call-backs is reduced significantly when you have a thoroughly tested product in hand. Thanks to some great HIL Test Systems, automated HIL Testing is redefining the automotive industry and the future only looks brighter.

## 1.3 AI, ML, AND NLP

Artificial intelligence (AI) enables the computer to think and process. Machine Learning (ML) is a statistical tool to explore the data. Deep Learning (DL) is multi neural network architecture to mimic the actual human brain. ML and NLP are the sun parts of AI. Natural language processing (NLP) is a test analysis process of deriving meaningful information from natural language text.

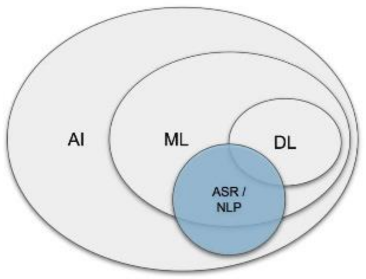


Figure 3. AI, ML, and NLP

Natural language processing is how computers process English or any language to carry out instructions. Natural language processing (NLP) is a subsidiary of artificial intelligence (AI) that makes machines break down and understand human language. We can use NLP techniques to interpret text data for analysis. Writing test cases from software requirements is a big job in testing. Software developers take Most of the time to develop test cases.

Moreover, test engineers have to wait for these test cases. So this is also one major industry challenge on how to minimise human efforts for writing test cases. My research also targets this challenge.

Natural Language Processing serves a lot of use cases when you're dealing with text or unstructured text data. Imagine you work for Google News and you want to group News articles by topic or maybe you work for a legal firm and you need to sift through thousands of pages of legal documents to find the relevant ones. This is where natural language processing can help. We will want to compile the documents in some fashion get features from them so we'll have to featuritis those documents and then compare their features.

Let's go through a simple example.

Imagine you have two very simple documents. In this case, the document is just a "blue house" and then the second document is "Red House".That means it's just a document of basically a single sentence. The simple way to featuritis text documents to feature is based on a word count. So we transform the blue house into a vectorized word counts.

"Blue house"  = (red,blue,house) = (0, 1, 1)

"Red house"  = (red,blue,house) = (1, 0, 1)

We create a vector count of all the possible words throw all the documents in this case they're blue and a house and then we just count how many times those words occur in each document. That means in this case for document one blue house we get 0 1 1 since Red occurs 0 times blue occurs once house occurs once. Similarly, in a red house, we get 1 0 or 1 because red occurs once blue 0 times in house one time a document. we can perform mathematical operations such as the Cosine similarity taking their dot products and then dividing it by the multiplication of their magnitudes or other similarity metrics to figure out how similar two text documents are to each other.

$$Similarity = cos(\theta) = \frac{A\ .\ B}{||A||\ ||B||}$$

We can improve on the bag of words by using word counts based on their frequency in the corpus and the corpus just stands for the group of all the documents because is something called T.F. IDF (term frequency - inversed document frequency)

Term frequency is the importance of the term within that document.

T.F. term frequency TF(d,t) = the number of occurrences of term t in document d.

The inverse document frequency which is the importance of the term in the corpus itself

IDF(t) = log(D/t) where D is the total number of documents and t is equal to the number of documents with the term.

Mathematically, TF-IDF is then expressed

$$Wx, y = tf_{x,y} * \log\left(\frac{N}{df_x}\right)$$

$tf_{x,y}$ = frequency of x in y

Dfx = number of documents contaning x

N = total number of documents

The reason we do this is so that we can get not just a word count but also some sort of notation on how important a word is not just relevant to the document but to the entire corpus of all the documents.

Most developers use machine learning frameworks to streamline the implementation of ML models. Machine learning is data-centric, so developers need tools to simplify the maths and statistics involved. This decreases the time-to-market and cost for implementing embedded ML. Some of the most popular ML frameworks include TensorFlow, Pytorch, and Caffe. These frameworks provide everything developers need to build, train, and deploy machine learning models through a simple programming interface.

The first step is for developers to choose a framework and algorithm. The embedded system and data requirements will determine this. Many frameworks include tools for creating data formats and performing data transformations. Developers should carefully consider the features essential for the ML models they're building. A model can then be trained using labelled datasets. This happens before deploying to the cloud or embedded systems. Developers will need a large dataset to split between training and testing sets. Missing or mislabelled data can negatively impact effectiveness. Once ML models are trained, it is deployed to either run on-device or in the cloud as an inference system. An inference system makes predictions based on a trained ML algorithm. This works by feeding data into the algorithm to calculate outputs.

## 1.4 K-NEAREST NEIGHBOR(KNN) ALGORITHM

Supervised learning algorithms are skilled in using labelled samples, and that is a keyword label, such as an input or the desired output is known. That means within your data set, you are going to have some historical features with historical labels. So, you already have that information, such as a segment of text could have a category label. So, you take a bunch of previous emails, and someone has already gone by and classified them using the correct label. So, they read the email and classified it as spam versus legitimate. Alternatively, we have many movie reviews, and someone has already gone and labelled movie reviews either positive to the movie or negative to the movie. Moreover, the idea would be for future text information, such as a future email using the historical label data, the network or machine learning algorithm could learn of the historical data to predict for new Data whether it belongs in the spam category or legitimate category or the positive category or negative category for these movie reviews. So, the way this works is for neural networks, the networks, and receive a set of input data along with the corresponding correct outputs. Then the algorithm or network will learn by linking its real output with correct outputs to discover errors. And then, we will modify the model appropriately, such as adjusting the weights and biased values in the network. So supervised learning is used in applications where historical data predicts likely future events. Moreover, the machine learning process for supervised learning looks like this, so let us go ahead and go through this step by step. So, the first thing to do is get data, and it depends on what domain you are working in, where this data comes from. This can come from your customers or can come from collecting things into a database online, or maybe it is biological data, and it comes from sensors, et cetera. So, at some point, the data has to be acquired. Once we acquire the data, then we need to clean and format the data so that our neural network can process it, and often we will do this using a library called PANDAS. Then we split the data into training data and test data. However, what we do here is we take some portion of our data, maybe like 30%, to be test data, and then the larger majority, the data, like 70 %, to be our training data. And what we are going to do here is we are going to use that specific training set on our network or model to fit a model to that training data. Then we want to know how well our model performed, so we then run that test data through the model and compare the model's prediction to the actual correct label that the test data. we know the correct label for that test data. So, you can run that test data features through the model, get our models predictions, and compare them to the right answer and then we can evaluate the model. And then maybe you want to go back based on that performance and adjust the model parameters, add more layers or more neurons to try to get a better fit onto that test data. And once we are satisfied with this, we can then deploy the model to the real world. Now, something to note here is what we just showed was technically a simplified approach to supervised learning, and it does contain a key issue which we kind of touched upon during that test train split. And the question arises, is it fair to use that single split of the data into one test set and one training set to evaluate your model's performance? So, when you test your model on the test data, you will get some sort of performance metric for regression tasks. It could be something like a root mean squared error for a classification task. It could be something like accuracy. However, is it fair to use the accuracy you get after test data as your model's final performance metric, since technically, after all, you were given a chance to update the model parameters again and again after evaluating your results on that test set? So how do we fix this conundrum? Well, to fix this issue, the data, especially in neural networks and deep learning, is often split into three sets, and we have training data, validation data, and then test data. So, we kind of introduce this in-between step of this validation. And so, what we end up doing is we have these three sets, and we have the training data, just as we did before. And this is used to train the model parameters. So, the model gets to look at the features, look at the correct output, and then fit this training data. And then, the next step is our validation data, which was kind of our test data from before. And so, what we do with this validation data is after training on the training data, we check the performance on the validation data and may be based on that performance, we go back and adjust our models, maybe adding more neurons or adding more layers, changing the actual architecture of the network, et cetera. And then you kind of repeat that process repeatedly until you are satisfied with your model's performance on the validation data. And now it comes time to evaluate the true performance of your model. So, what do we do? Well, that is why we have that third split of test data that the model has never seen before. And what you use that final test data set is to get some final performance metric. Now, the key thing to note here is that once you run the model through the test data, that is going to be the performance metric that you expect your model to perform within the real world since you are not going to go back and adjust your models, weights or parameters or anything else. Once you go onto that final test data set, you are technically not allowed to go back and adjust the model to try to refine your performance on that final test data set. So, you can think of it this way. You train on the training data to fit your model. Then you use the validation data to see how your model performs unseen data and then go back and adjust your hyperparameters. But when it comes time to kind of report to your boss, how well will this model do in the real world? That is where your final test data set comes into play and the test data set. Once your pass through test data and you get that performance metric, that's it. You do not get to go back and adjust the parameters. Otherwise, you kind of cheating yourself again on understanding the model's real performance on truly unseen data. So, what this means is after we see the results on the final test, that we don't get to go back and adjust any of those model parameters. This final measure is what we labelled a true performance of the model to be on unseen data.

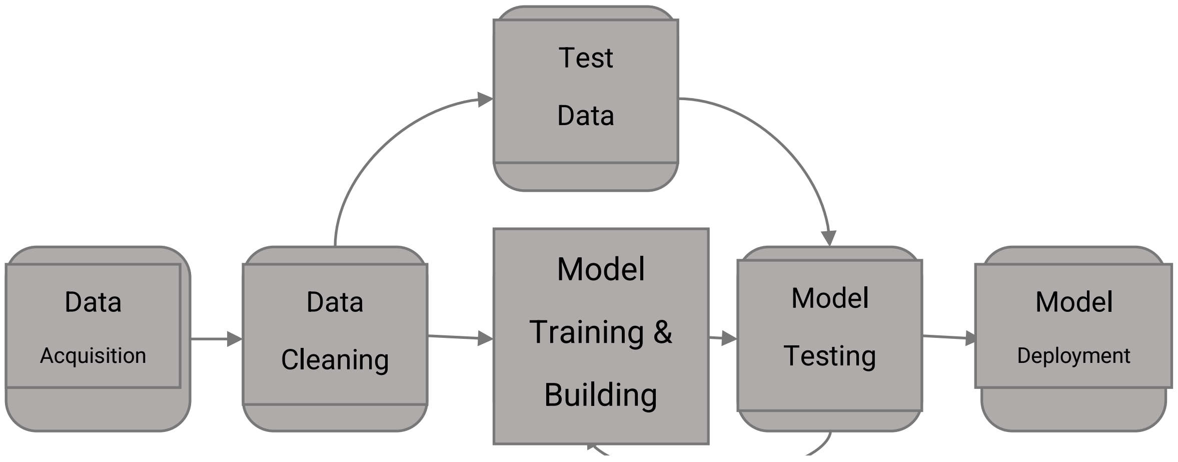


Figure 4. ML process block diagram

The k-nearest neighbors (KNN) algorithm is a straightforward, easy-to-execute supervised machine learning algorithm that is used to resolve classification and regression problems. Machine Learning is all about identifying patterns, creating its algorithm, and consistently evolving. The machine usually creates patterns, and it uses various techniques and algorithms for it. K nearest neighbour is one the most straightforward and yet effective algorithms for a classification problem in machine learning. It is used for classification and regression problems, i.e. it can classify any particular event/attribute and even predict its value. This method is simple to compare with the other complicated algorithms

. It works on measuring the distance between the data points. Measuring the distance makes a pattern to create a relationship between two points. This way, it decides to which group the new data point should belong.

If we need to calculate the distance between two points mentioned in the Figure 5 image, say A(Xp, Yp) and B(Xq, Yq), then it can be done by using a simple formula

$$Distance\ (f) = \sqrt{(Xq - Xp)^2 + (Yq - Yp)^2}$$

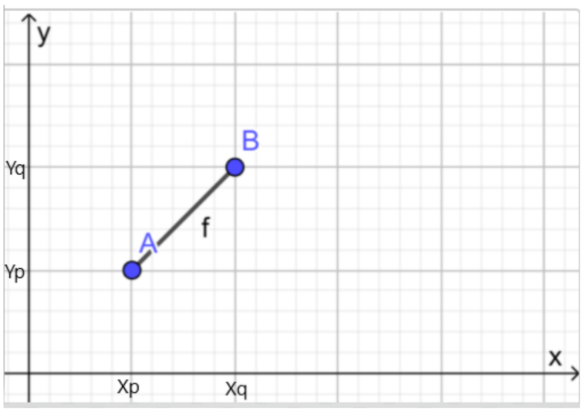


Figure 5. Distance between two points

This formula is the Euclidean way of measuring the distance between two points. There can be other ways too for this distance measurement. The method mentioned above was followed to calculate the distance between various data points. KNN has two distinct properties. It works great for a large volume of data. It is flexible in choosing the variables (no assumptions) & can fit a large number of functional forms. It can give better performance while predicting. In K nearest neighbours, the value of K denotes the number of nearest neighbours which have a minimum distance from our new data point. If we say k=3, we have to find out the three nearest neighbours for our new data point. If there are three neighbours and one is from the red dataset, and the other two are blue, then the new data point will be blue. More neighbours mean more clarity in classification, but it increases the complexity and timeframe for classification. There is one more critical aspect of the value of k, and it should be an *odd number* always for better classification.

KNN is a classification algorithm, and classification is determining what group it matches. To function the KNN algorithm. It requires some examples, usually called reference data. The data record that needs to classify, it calculates the distance between the predicted data record and reference information. It looks at K the nearest data record at reference data. For example, if K = 5, then it will look at the five closest records. In that, the majority record is the predicted class of algorithm. The following example gives more clarity about the KNN algorithm.

Using the KNN algorithm with the above reference data, we can predict the sport based on age and gender. Let us check how The KNN algorithm can predict what class of sport for named Sitara female whose age is 5.

| Name | Age | Gender | Sport |
|---|---|---|---|
| Ajay | 32 | M | Foorball |
| Vasu | 40 | M | Neither |
| Sonali | 16 | F | Cricket |
| Trisha | 34 | F | Cricket |
| Ramu | 55 | M | Neither |
| Ramesh | 40 | M | Cricket |
| Reshmi | 20 | F | Neither |
| Vijay | 15 | M | Cricket |
| Rani | 55 | F | Football |
| Ashish | 15 | M | Football |

Table 1 Reference data

| | | | |
|---|---|---|---|
| Sitara | 5 | F | ? at K = 3 |

Lets consider K = 3.

First, we need to convert descriptive data to numeric. We identify males as "0" and females as "1".

KNN algorithm makes use of the Euclidean distance equation. And highlighted values are nearest to K value.

| Name | Age | Gender | Distance | Sport |
|---|---|---|---|---|
| Ajay | 32 | 0 | 27.02 | Foorball |
| Vasu | 40 | 0 | 35.01 | Neither |
| Sonali | 16 | 1 | 11.0 | Cricket |
| Trisha | 34 | 1 | 9.0 | Cricket |
| Ramu | 55 | 0 | 50.01 | Neither |
| Ramesh | 40 | 0 | 35.01 | Cricket |
| Reshmi | 20 | 1 | 15.0 | Neither |
| Vijay | 15 | 0 | 10 | Cricket |
| Rani | 55 | 1 | 50.0 | Football |
| Ashish | 15 | 0 | 10.05` | Football |

Table 2 Distance calculation

In this case, we considered K = 3. The above three closest records are 9.0, 10.0 and 10.05.

| Name | Distance | Sport |
|---|---|---|
| Trisha | 9 | Cricket |

|  |  |  |
|---|---|---|
| Vijay | 10 | Cricket |
| Ashish | 10.05 | Football |

Table 3 Closest records at K=3

The majority common and nearest neighbour sport for Sitara is cricket. The KNN algorithm is used to predict the library class for a combination of parameters in Objective 1.

## 1.5 THESIS OUTLINE

In this thesis, A novel framework was designed for test automation and implemented using LabVIEW and Python programming languages. Also implemented an algorithm to make the comparison test images and reference images to automate embedded vision software. This research work focused on three objectives.

**Objective1:**

Design framework to generate Test scripts by developing business logic with Machine learning techniques. It is the main object of this research work. Development of 100 test scripts takes a minimum of 3 months with 3 test engineers. As per my analysis, automation takes one week to generate 100 scripts, including verification.

Script generation has intelligent logic, which considered previous examples analysis. We do train the system using KNN supervised machine learning algorithm as per our requirements to generate scripts. For this input, which is the test case has to write in the custom format.

Test case examples:

Step1: Arrange temperature x above 40 degrees.

Step2: check the Fan is turned on or not.

Ste3: also check warning led indicator is on or off.

Customized test case;

Step1: Set temperature x = 41 degrees.

Step2: check fan = ON.

Ste3: check "warning led indicator" = ON

| Action | Parameters | Value | Remarks |
|---|---|---|---|
| Set | temperature x | 41 | |
| Check | Fan Status | ON | |
| Check | warning led indicator | ON | |

Table 4 Proposed UI Template to write test cases

.

**Objective2:**

"Improving Algorithm with advanced image processing techniques to the comparison Test image with a reference image to validate display application software."

Most of the embedded products have display units. These display units show different images based on embedded software development. The output of the display units is also part of testing. Currently, the output of the display unit is observed by the tester and recorded as pass or fail. This manual testing takes enormous time. It can automate by developing image processing code, which compares the test image with the reference image. There are a few algorithms available to compare images. However, the accuracy percentage is 60 to 70%. This objective goal is to increase the accuracy of comparing two images.

**Objective 3:** Test Cases generation from software requirements using Natural language processing (NLP).

Writing test cases from software requirements is an intelligent job. The only domain expert can write test cases by understanding software requirements. This process can automate using Natural language processing (NLP) techniques.

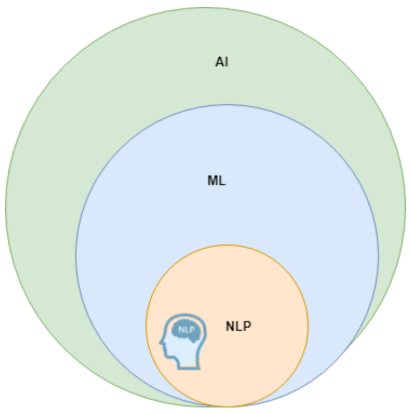


Figure 6 AI, ML, and NLP work together

AI educates systems to do intelligent things. ML educates systems to do intelligent things that can learn from the experience. NLP teaches systems to be intelligent to learn from experience and understand human language.

Example of NLP analysis:

Sentence: "If the temperature X is above 40 degrees, then Fan should turn on and warning LED indicator should turn on.

Expected output from NLP engine:

1. Set temperature X = 41 Degrees
2. Check Fan Status = On
3. Check "warning LED" status = ON

**REQUIRED TOOLS AND SOFTWARES:**

- PYTHON 3: Python Programming Language used here to build tool and operate all the features.
- LabVIEW: LabVIEW graphical programming tool
- sci-kit-learn it is a popular  machine learning library
- TEST STAND: Test Stand is test management software to develop scripts to automate manual test processes.
- Pandas: pandas are a Python package for data analysis and manipulation.
- TensorFlow: It is an end to end open-source platform for machine learning.

The above three objectives aim to fulfil the core issue of test automation of embedded software testing in the automotive and home appliances industry. Where evermore repeatable possible and critical to working on manual testing, those scenarios are identified and provided a cost-effective solution with these three objectives.

**Chapter 1** explains the background for test automation and research objectives.
**Chapter 2** discusses the relevant previous research in Test automation and the research gap
**Chapter 3** describes the objective -1 implementation and results analysis.
**Chapter 4** describes the objective -2 implementation using image processing algorithm and results from the analysis
**Chapter 5** describes the objective -2 implementation using the OCR method and results analysis.
**Chapter 6** describes the objective -3 implementation and NLP algorithms results analysis
**Chapter 7** presents the conclusion drawn from the present research work. Moreover, this chapter also depicts the future research agenda for practitioners, engineers, and potential researchers.

## 1.6 SUMMARY

This chapter explains Test automation fundamentals, Digital image processing, Machin learning basics and Natural language processing methods. It also describes the KNN supervised machine learning for example.

# Chapter 2 LITERATURE SURVEY

Test automation is crucial across all product lines to reduce cost and money. The relevant previous research performed on test automation is in this chapter.

## 2.1 RESEARCH GAPS

In software development procedures, testing is often considered for half of the total development work. Therefore, it is necessary to reduce the cost and enhance the efficiency of software testing with test automation. In the past times, significant research effort has been spent on developing and studying test cases generation, automatic and other semi-automated testing techniques. We require more stable test automation based on the theory and practical testing analysis. Hence, the automation of several testing events is now turning into an important part of the industrial exercise. The following are the significant research gap in the embedded software test automation.

1. Test Management tools are available, but still, test automation is a time-consuming task by providing inputs to tools.
2. There is a lack of high accuracy of the image processing algorithm, finding similarities between two images—major embedded industries looking for a solution.
3. Writing test cases from a software requirement document is a big task and a massive cost.

## 2.2 RELATED WORK

Model-based Integration and System Test Automation MISTA converts the test cases into executable test code by mapping model-level elements into implementation-level constructs. MISTA has implemented test generators for various test coverage criteria of test models, code generators for various programming and scripting languages, and test execution environments such as Java, C, C++, C#, HTML-Selenium IDE, and Robot Framework [24].

Now a day, using software has become an essential part of life. Software is integrated into everything useful in day-to-day life like washing machines, smartwatches, smart TV, microwave, pacemaker etc. Any type of fault in software may result in financial or life losses. So, proper testing of software is mandatory before its implementation. Testing is an essential module of the Software Development Life Cycle (SDLC). According to Beizer (2009), software testing is a costly process as it consumes nearly half of the expense of a software development process. It is most expensive in terms of time as it must be done in every phase of the software development process. Testing consumes a lot of supplies and does not add anything to the products in terms of functionalities [3]. Therefore, software costs must be reduced. S thamer (1995) explained that for cost reduction, more focus is required on automated software testing.

Software testing is the major methodology for finding errors and guaranteeing the quality of software, according to Zang and Kou (2010). Therefore, much effort and time must be spent on test cases generation to test the software for reliability. Test cased generation by manual method uses plenty of periods and it also varies depending on the ability of an individual. Consequently, the possibilities of mistakes made at the time of planning of test instances are enormous which leads to them being included in the list of errors in the system after tests are carried out too. Going Instead, some test cases are clearer than others following the conditions of the discovery bugs. Therefore, a systematic approach for testing is required to differentiate good (suitable) test data from bad (unsuitable) test data. The tool must be appropriate to detect good test data. To find this, it is extremely important to automate generating test data. Software testing tools need to ensure that the test cases generated by it is dropping under the appropriate testing standards and are of great quality. Testing tools should generate test cases with an expanded environment, and they should not ambush local targets. The tool should be robust, reliable, general, and flexible. Test data that is generated for a single program may or may not be automatically good for a different program. Therefore, tools should be adaptive in nature for creating test cases for the software following the test [4].

Software testing plays a very important role in the development of the software life cycle. If testing is not done properly, errors may come at any time during software execution that may cause financial or life losses. So proper and careful testing is required before the implementation of software. The intention behind the testing is to discover defects. The system fails when a defect triggers during execution. Proving software code is error-free is not the purpose as no one can give a guarantee about it. Testing is done to check and identify the failure in software. Furthermore, identified failure may be corrected. S.T. identifies the product quality, generally called an investigational approach to collect information from the test for end-user according to Gill and Ritu (2007), Mitrabinda and Mohapatra (2013) and Srikanth and Laurie (2005).

History is a witness of financial losses and life losses that occurred due to improper testing before the launch of the final product. When identified defects are not handled properly, causes severe problems afterwards. Some real examples related due to defects are described as below: Majesty's Revenue and Customs (HMRC) Tax Blunder: The Company involved in tax collection in the UK and its software PAYE (Pay as You Earn) was hit by a bug that affected approximate 5.7 million people in the UK. The wrong text code was allotted to taxpayers due to which taxpayers paid more than the actual tax. Amazon Christmas Glitch: in 2014, Amazon platform, vendors are surprised to see their products are on sale for just 1 Penny. Many persons purchased expensive items like a mobile phone in just 1 Penny. Royal Bank of Scotland (R.B.S) in 2014 also came under the scanner. R.B.S. must hefty fine (£56 M) for no access to its users. It occurred due to an error in the software. It affected more than musers at least for 1 week as reported in local newspapers. The users were unable to access personal accounts or take the cash out. The payments for all purposes were ceased. This claimed „small issue‟ stopped the salary payment to employees of organizations that chose R.B.S. as a medium to transfer salaries. Border and Immigration system case, UK: the estimated loss was approximate £1 billion. Due to software errors, systems were unable to deal with backlog cases and leads to a backlog of 2900 applications. London Airspace 2014 case: It's common hearsay that bad weather was a reason for the cancellation of airflight. It's very uncommon in the case of software. But in 2014, London airspace was forcibly closed. The air traffic control centre identified a malfunction due to software [6] . The software was unable to manage arrival/departure. Even after repairing work, the airport system took time to come back on the routine track. More than 50 flights were cancelled. The authorities of Heathrow were bound to send the flights back to their starting points. Bizarre Incident 2017: American Airline closely avoided a great software glitch. Due to a scheduling error in the system, the airline announced the week-off near Christmas. During Christmas, usually, all airlines don't approve even emergency leaves. This became a great surprise to all pilots. If no one has reported it or it's not been caught, more than 15000 flights would have been affected. The airline finally resolved the problem by correcting the software. Law students, US 2014: In Aug. 2014, the law students (great in number) across the U.S., submitted their completed exam files using the software. They need to press a "Submit-Button" after uploading the files. After following the procedural steps, they found that test papers were not accepted by Bar Exam software. What happened afterwards is a surprise to none. The Bar council faced multiple lawsuits from upcoming lawyers. Bitcoin 2017: In March 2017, the most serious glitch happened with Bitcoin. They suffered from two failures one after another. A software bug affected more than 100 Bitcoin nodes. These were close to 70% of the total unlimited nodes. One could understand what happens if 70% node disappears from the network. The goal of testing is to ensure that product meets the requirements of the customer. Software products are checked in a pre-planned way to find software's outputs are the same as demanded by the customer. Software testing is done while keeping in mind that software under test has defects and the tester must detect maximum defects. The objective of the software test is to find bugs in software products but not to prove that software is defect-free. It is impossible to make 100% defects free software as some errors appear during run time in rare conditions. To detect the defects, test cases are designed. Basic Terminology - The basic terminologies used in software testing are mistake, fault, and failure, which are defined as Mistake – Occurrence of incorrect result due to human involvement. Fault (or Defect) - An incorrect input in a program. This may be a step or a process or a data definition. Failure - The incompetence to accomplish the specified function. If a system or component is unable to perform its purpose can be termed as failure. Error - Commonly referred to as dissimilarity in computed value and theoretically calculated value. The reason behind the occurrence of error may be due to a faulty program. Incident –It’s a symptom that alerts a user regarding the occurrence of a failure. The occurrence of failure is generally not deceptive to a user. Test – Identifying the error, fault, failure is the purpose of a test. It is an act by which software undergoes an exercise using pre-specified test cases.

**Test Case** – A test case is a way to check the program behaviour. It has prespecified inputs and outputs. These are checked during a test.

**Validation vs. Verification** – S.T. is a process involved in the validation and verification of software. A program or application is expected to meet its business and technical requirements and S.T. is required to validate the claims.

**Validation -** Validation deals with the evaluation of products meeting the requirements. This is a confirmation of the making of the correct product or system.

**Verification -** It is the process of confirming the judgement for a system/product. Does it confirm that the developed system/product has met every condition that was assumed in the initial phase i.e., correctly making the product?

**Testing Documents** – A written document of judgment about a system or product meets the standard or not. Now a day, it’s compulsory to provide test documents. This written material is also required for verifying and future enhancement in test quality. IEEE 829-2008 is a well-defined standard used for software and system test documentation. The standard involves a set of documents for a use covered in 8-stages. In every stage, a separate document is prepared. These standards specify a particular format. Preparation of all the documents is not made mandatory by IEEE but preparing all could be beneficial. The details associated with these documents are:

**Test Plan -** Most essential planning sheet which covers:

- What to test?
- Who will do testing?
- How it will be done?
- How much time will be consumed during testing?
- Details related to testing coverage.
- And above all, what is the quality of software testing?

**Test Design Specification -** It covers detailed information regarding test conditions, criteria for passing the test. The expected results are also documented.

**Test Case Specification -** It deals with test data requirements that are required to run a test under-identified test specification.

**Test Procedure Specification-** It covers the details related to testing steps to be followed. It also looks after the preconditions of the test.

**Test Item Transmittal Report** – it is This document deals with information related to the progression of test-item from one stage to the last stage.

**Test Log -** It records information about tests cases involved, who performed, what the order is and the results.

**Test Incident Report** – it is a complete report sheet about failed test cases. It covers the actual output versus expected output. It also gives a view about the reasons for the failure of a test. The name "Incident report" has been given deliberately; it's not a fault report. Because the difference in actual output and expected output may have occurred due to many reasons other than any fault in the system including the wrong expected results. The other reasons may be running the test the wrong way or differentiation in output requirements which mean that more than 1 interpretation can be made. The report also consists of incident details related to failure besides the actual output and expected output. It also gives the supporting evidence that is required in the removal of the found errors. This is an assessment report of the impact of a particular incident.

**Test Summary Report** – It is commonly known as TSR. It is a managerial level report. It contains all the important details collected by the execution of test cases. This report can be used as a referral of test quality as well as it also quantifies the testing efforts. It contains all details in form of statistical data obtained from Incident Reports (I.R.). This report can be considered as a record-book of what is tested‟ and „time consumption in performing the test‟, and it can also be used as a referral for future test planning and future improvements. This final document confirms whether the software system under test is fit or not for the purpose mentioned in Software Requirement Specification (SRS) specified by project stakeholders. In software testing, the software tester tries to determine a set of test inputs that are effective in testing the code of the program/software. During the design of test cases, some test cases can detect defects, and some may fail to discover the faults. The test cases which can discover are called potential test cases. With a set of potential test cases, the tester tries to uncover as many faults as possible. Therefore, a test sequence that can find many defects is better than one that can only uncover a few defects. Some test cases may be good, or some maybe not be so good. A test case can be called good if it is finding bugs. Creating good test cases is a complicated task. It is complicated because – (i) the various types of test cases are needed for various types of information. (ii) It is not possible that all test cases surely be good but can be good in other ways. (iii) Human involvement is there in creating test cases that may be based on confirmed testing styles like domain testing or risk-based testing. The testing fundamentals principles are –the testing process must find defects detected at customer ends. Testing must be done with the mindset to find a defect, only then tester can find errors otherwise not. She/he never feels that software is free from defects. Persons who can work in a team and have deep analytic knowledge and talent are the main factors for successful testing. Automation is a key to good testing as it helps in meeting the goal in a very short time than manual testing. Throughout the software, life cycle testing must be performed so that defects can be detected at earlier stages [12]. Defect Classifications – There are numerous ways available to classify the defects. But it is advisable that one organization must follow a particular classification scheme for all its projects. It is worth noting that its does' matter much that which categorization system is selected as many defects are considered into many categories. This is one of the reasons that all developers, testing personals and people involved in Software Quality Assurance (SQA) are advised to be very careful while entering the defect data in record books. The identified defects and their repeatability could be used as a guideline for test planning and test design. One is advised to follow or use the execution-based testing strategy so that it surely detects the types of defects which are expected. For new software as well as for any modified software, a test must be designed in such a way that it must detect the defects which occur recurrently.

**Requirement and Specification of Defects**– If a high-quality product or software is required, the defects must be identified during starting of the software life cycle. Then only the high quality can be ensured. The initially found defects usually stay for a longer period and it’s commonly not possible to rectify them in later phases. It is also a common occurrence that some contradictory or redundant requirements are mentioned in the report because all the records are maintained using common language representation. This is also one of the reasons that sometimes, unclear, and imprecise requirements are found in the documents.

- Functional Description Defects – It describes the purpose or function of software along with details related to inputs and outputs and what is incorrect or incomplete.
- Feature Defects – A feature of a software component is the distinguishing characteristic of the component. This characteristic describes the software system.
- Feature Interaction Defects – As mentioned above, one component may have many features, and these may interact with each other. These types of defects may be seen due to incorrect narration that how these features should interact with each other.
- Interface Description Defects – Developed software must be easily connected with hardware and operating system. If this interfacing is not done properly, these kinds of defects generally occur. So, it is quite necessary to make the proper interface between developed software and external software and hardware.
- Design Defects –As the name suggests, an improper interaction among system components, improper interaction among the components and operating system software or hardware, are not designed correctly. The defect in designing of algorithms, the control-systems, mentioned logic, data elements, the module interface description defects and the external interface defects are examples of these kinds of defects

(i) Algorithmic and Processing Defects – The main reason for the occurrence of these defects is that processing steps in the designed algorithm are incorrect. It means that the steps used as per pseudo code are not correct.
(ii) Control Logic Defects – These defects are related to logic flow.

(iii) Data Defects –Purely occurred due to incorrect data structure design.

(iv) Module Interface Description Defects –These defects mainly occur when module elements use the parameter types which are not correct and inconsistent, ordering of parameters is not correct or when parameters are not correct in number.

(v) Functional Description Defects –If the design element is unfit and/or absent and/or uncertain, these defects are known as Functional Description Defects.

**Coding Defects** - The defects occurred due to errors in implementing the code.

(i) Algorithmic and Processing Defects - The Ability to Add levels of programming detail to design, code-related algorithmic and processing defects including unchecked overflow and underflow conditions, comparing unsuitable data types, converting data type into another, wrong ordering of arithmetic operators.

(ii) Typographical Defects –Also referred to as principal syntax error, e.g., incorrect spelling. These defects are identified by a compiler, by self/peer-reviews.

(iv) Initialization Defects - These defects occur at the start mainly when starting statements are misplaced or wrong.

(iii) Data-Flow Defects –In general, data flows through a particular predefined set of operative sequences. For example, a variable is required to be initialized, before using it in any calculation. But it's twice initialization surely to be avoided before in intermediate use.

(iv) Data Defects –Represented by the wrong enactment of data structures

(v) Module Interface Defects –Similar to module design elements, the interface defects in coding mainly occurs due to the usage of wrong or varying parameter types. The other reasons are wrong no. of parameters or wrong order of parameters.

(vi) Code Documentation Defects –As the name suggests. The defect occurs when there is a considerable difference in code documentation and output of a program. This kind of defect also occurs due to inadequate code documentation.

(vii) External Hardware, Software Interfaces Defects –There is a long list of reasons for the occurrence of these defects: system calls, links to databases, I/O Sequences, memory usage, resource usage, interrupts and exceptions handling, data exchange using hardware. There must be a reason behind the design of the test case. Test cases must be used after proper verification. Defects may come in clusters and need more focus in such cases. It is the tester’s responsibility to write test cases in form of failure causes. But one should understand and accept that all faults could not be detected. Test cases are designed in such a way that will cause system failure, but it is expected that no test case repeats the same failure. According to Hambling (2010), conducted detailed studies to understand the money involved due to the presence of errors during each stage. It is almost impossible to put a statistical amount in the form of the cost involved in identifying the defects at each level in the software development life cycle.

Our day-by-day life is loaded with vulnerability, uncertainty, and difficulty. The vulnerability in the problem area, a vulnerability in the solution area and human involvement in the development of software has made and expanded the vulnerability in the software development process. The same problem also exists in software testing. Vulnerability in software testing is present because of vulnerability in the fitness function, unsure nature of fitness parameters, clashing nature of objective functions, vague testing conditions, vagueness in test cases choice, doubtful classification, uncertainty in test plans, no proper test planning, questionable rarities (SRS, SDD, Source Codes), unsure error handling and checking procedures and policies, estimation policies and their quality and other testing procedure. Kumar et al. (2012) and Kumar et al. (2013) described the vulnerability in the fitness of test cases, unclearness in fitness parameters, quality and accuracy of estimation, uncertainty in characterization as central issues in programming testing. Software testing is also involved human beings, so testing is a human escalated movement and, in this way, presents a vulnerability. Human participation in every phase of software unavoidably brings vulnerability and irregularity into the testing of software.

During software development phases, several human characteristics such as dedication, mood, experience, human psychology, knowledge have a great impact on the quality of software through testing. It has expanded the hard work, testing cost and compromised the testing quality. Existing conventional procedures for optimization of software test cases have not taken into consideration this ambiguity impression in software testing. As per Gill and Tomar (2007), Kumar et al. (2011), Causevic et al. (2012), Lee et al.

(2012), Mitrabinda and Mohapatra, (2013), software test case optimization techniques are obsolete and require computational insightful strategies.

## SOFTWARE TESTING ISSUES

Considering the absence of known procedures, decisions are made based on knowledge, instinctive appraisals, and heuristic policy. Adequacy criteria is a key issue associated with test cases optimization. According to Kumar et al. (2011), decision-makers need to answer the accompanying inquiries with economic criteria, what test cases might tester use to test the program? How test cases should be chosen to achieve maximum coverage ability? At the point when and how to decide if testing has been done sufficiently? At what point need to quit testing or proceed further with the testing? At what point need to quit from optimization or proceed further with optimization? How to establish that whether to produce optimized test case or optimize the arbitrarily created test cases? How to decide the quality of the program from test cases? At the point when the quality of the program ought to be assessed through testing where the program/software is time-variant? What will be the chances of system breakdown?

## IMPLICATION OF TESTING

Today software is used everywhere. Every item used today in daily life has an inbuilt software component. Software becomes an essential part of people. In hoses, offices, automobiles, banking, industries, telecom, communication, weather forecasting, health sector, defence sector, the software is used in one or another way. It becomes as common as electricity in the early days. Software is embedded in every commonly used hardware, gadget, and device. Mobile phones, washing machines, microwave ovens, induction cookers, wrist watches etc. all have embedded software. Software failures in critical projects such as the share market, health sector is not acceptable.

Software used in the pacemaker cannot be shut down even for a minute for rectification of error if it is exposed. So, software failures are not tolerable as it not only leads to financial losses but also involves human life. Almost every system used these days is controlled by software directly or indirectly, so these services cannot be afforded to fail due to software defects. So, proper testing is required before delivery of software as these systems must be run continuously with 100% reliability. The designed system must be so reliable that it should run every check time i.e., the software must undergo rigorous testing before delivering to the user.

## MOTIVATION

Software testing is the process of testing a program with the typical objective of searching errors before handing them to the user. It usually performs quality improvement, verification, validation, and reliability calculation. A software bug is defined as a mistake or error in software or program that gives a wrong or unwanted output. The consequences of bugs may be extremely dangerous. Software testing is a very costly process. Running of software without software testing may lead to monetary losses that are exponentially higher than that of software testing, in real-time systems such as Railway Reservation System, Robotic Surgery, ATM security system, Defence Management System and Share Market Software. So, the Software bugs in the real-time system will result in serious effects.

National Institute of Standards and Technology, the commerce department of the US, published the report and concluded that software bugs or failures are ubiquitous, and their annual cost is $59 billion, i.e., 0.6% of US GDP. It clearly shows that financial loss due to improper testing is very high. So, it is an important need of time to discover new adaptive and smart techniques or strategies to conduct adequate testing economically. The test cases generated by various combinations of input variables are very high. The pool of test cases also contains redundant, obsolete, unfit, vague test cases. Therefore, software testing is a much costly process of software development, all test cases cannot be executed due to budget and time constraints. Running the duplicate, vague, wrong, biased, and error-full test will boost up the cost factor unnecessary. It requires test case optimization. The most difficult issues for software testing are the hard work, cost associated with test case optimization and test data sufficiency. Test case optimization will unquestionably chop down endeavours, cost of software testing and enhance the sufficiency of test cases.

There are a few destinations of test case optimization like most extreme number of defects detecting capacity, least test plan endeavours/cost, least execution cost, the greatest scope of customer need and codes, most extreme mutant killing score etc. The test case optimization is an NP-hard problem. Existing test case optimization methods are outdated and not satisfying the goals of the software industry because the test case optimization is NP-Complete, information, knowledge, and search optimization issues. Optimization of test cases by evolutionary and natural computing was not appropriately investigated.

Above issues of testing have constrained, supported, and propelled the scientist to investigate evolutionary and natural computing techniques thoroughly for software testing. In this work, the focus on the generation of unique, distinct test cases by integrating genetic algorithm, particle swarm optimization and ant colony optimization computing approaches was investigated.

## TESTING CLASSIFICATION

There are various types of testing varies based on the test closer to code and user.

### Black Box Testing (B.B.T)

B.B.T. is quite popular with name dark box testing. B.B.T. works as a link between the outside world and test item – means a program or complete unit using a pre-defined interface. The interface could be an application interface, maybe an internal module interface or the I/O depiction of a batch process. B.B. Ts confirms whether interface definitions are adhering in all situations or not. The notable point is knowledge of the internal structure of the software is not required. B.B. Testing is performed as per the needs of the customer including other related considerations. It must be noted carefully that the tester has nothing to do with the coding of the program. The tester mainly focuses on the specifications as defined in S.R.S (Software Requirement Specification). In B.B.T., the tester is only informed about the sets of input and expected output and the tester have no idea about how program inputs are transformed into output by the software. So, while conducting B.B.T., the tester is only concerned with designated inputs and pre-known or expected outputs. Beizer (1990) and Ince (1987) mentioned that the knowledge of the functioning of the test program is essential while performing the testing under black-box testing. Testing based on the black box provides overall functionality-based verification of the system under consideration. Black box testing demands requirements that may be stated or implied requirements. Both valid and invalid inputs may be implied in black-box testing. Singh (2013) tried to find the errors involved in testing. The errors under consideration are - interface errors, performance errors, data structures errors, external database access errors, incorrect or missing functions related errors, initialization errors, termination errors etc. Product acceptance tests also fall under the category of black-box tests. These tests are done by customers. The test is performed to confirm that the product has met all the pre-requisites. Test cases are created based on the task descriptions.

According to Murnane (2007), black-box testing includes different techniques such as:

**Equivalence Class Partitioning** - Murnane (2007) described equivalence class testing as a type of black-box testing is assumed to divide a program's I/O domains incorporated in a finite no. of (valid/invalid) courses in such a way that test-scenarios in one parted class to experience the same function or undergo the identical behaviour. That means the partitioned program acts similarly for each input value that is part of an equivalence class. Test cases are planned in such a way that they could test the I/O domain partitions. Equivalence class depends upon input data range. It is defined by examination and analysis of the data entered range. As all the test cases are the same so only one partition is required for examination, so quite less few test cases are needed to achieve functional coverage. Shao (2007) further described that the result accuracy mainly depends upon the tester, the tester must be able to recognize the partitions of I/O spaces, because distinct sequences of program source code are to be implemented. The Steps for creating test cases for equivalence class partitioning are:

- Describe equivalence classes.
- Choose and compose the initial test case which must cover the maximum possible valid equivalence classes.
- Continue composing the test cases till the inclusion of all valid equivalence classes should have been confirmed.
- In the last stage, a single test case is composed for each invalid class.

As described by Murnane (2007), limitations of equivalence class partitioning are described below:

- The first limitation of equivalence class partitioning is: each set of data that falls under the same equivalence class must be processed similarly in the system as it was assumed that data is the same in each partition.

**Boundary Value Analysis (BVA)–** Nidhra (2012) and Murnane (2007) explained that before applying the boundary value analysis, the tester must identify the edge conditions of the I/O classes as per specification. The tester must create such tests which should be able to exercise the edges of the identified I/O classes. Test cases are dependent upon the „boundaries" of similarity classes. Usually programming mistakes occurring at the boundary of a uniformity class called "Boundary Value Analysis". It is quite common that programmers don't check special processing requirements particularly at boundaries of equivalence classes. Careless use of „<=" is a very common mistake of a programmer.
The choice of boundary values of a class can be divided into 3 parts:
(a) Above the boundary.
(b) Below the boundary
(c) On the boundary.
Common restrictions are of boundary value analysis are:

- It is not applicable for Boolean and logical variables.
- It is unable to calculate the boundary analysis for some cases e.g., countries.
- Not useful for strongly typed languages.

**Decision Tables** – As per Beizer (1999), decision tables are design experts' skill in a compact shape. It can be also called human-readable rules. Decision tables are useful where the outcome or the logic of a program depends upon pre-specified decisions/rules which are compulsory and to be followed. Sharma (2010) described that the decision table consists of condition, sub condition, entry, action, and action entry.
The steps in applying the decision table testing are:

- It starts with the analysis of test inputs. Followed by listing the several conditions in the table. • Calculating all expected no. of combinations or rules.

- Filling the data in available columns (in the decision table) with all expected no. of combinations.
- Finding those cases in which ideals accepted by the variable are found irrelevant for any given sequence. If found, represent it by the “Don’t care” icon.
- For every combination of values, action needs to be identified.
- Creating a test case for each rule (at least one). It is worth noting that for binary rules, a separate test for every combination is likely to be enough. Otherwise, if a situation has a certain range of values, testing must be performed at lower range values as well as higher range values

**State Transition Diagrams:** It is commonly known as State Graphs. Kansomkeat (2003) termed the state graph as an excellent tool that is used to capture the system requirements and verify the internal system design. State transition testing can be employed when a system remembers what has been occurred before or it remembers when valid/invalid orders of an operation have existed. State graphs come into action when a system transforms from one state to another. Scollo (2005) enlightened that state graphs are represented with symbols. For example, the circle represents the state, arrows represent the transition and events are represented by the label on the transition. Ran (2009) described that transitions and routes involved between the beginning state to the ending state are represented with the help of a change over diagram. Orthogonal Arrays (O.A.) - Nidhra (2012) described that Orthogonal Array Testing Strategy can be called a systematic cum arithmetical way of testing for pair-wise interactions obtained from getting the tiny stage set of test scenarios out of very large no. of scenarios. OATS is employed to reduce the no. of test combinations and provide the highest possible coverage with the least no. of test cases. OATS uses an array of values in form of pairing using all selected variable factors. The main advantage of using the OATS is it doesn’t consider all possible combinations of factors and levels.

**All Pairs Testing** – Nidhra (2012) described that all pair testing is a unique testing technique that is widely accepted. This technique is used for verifying a finite no. of values when no. of parameters is finite. It is mainly used for keeping the no. of test cases rational.

**Comparison of black-box testing techniques** – A no. of black-box testing methods has been discussed so far to distinguish each other in the requirement of no. of test cases as well as efforts involved in the development of these test cases. Fig. 7 and fig. 8 shows how three main black box testing techniques (Boundary Value Analysis, equivalence class partitioning and decision table) distinguish each other. Fig. 9 shows a comparison in several test cases required by different methods of black-box testing. It depicts that the decision table technique is more sophisticated as in this technique both data and logical dependencies are considered.

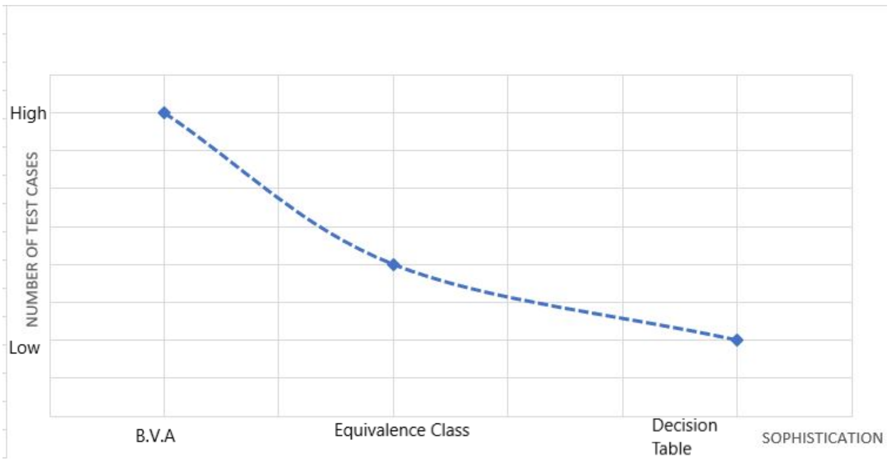


Figure 7. Test cases required as per the testing method

Fig. 8 presents a comparison of efforts required to identify test cases by different methods of black-box testing. It can be easily depicted that the effort involved in the case of boundary value analysis, (B.V.A.) is lower in comparison to the rest of the techniques.

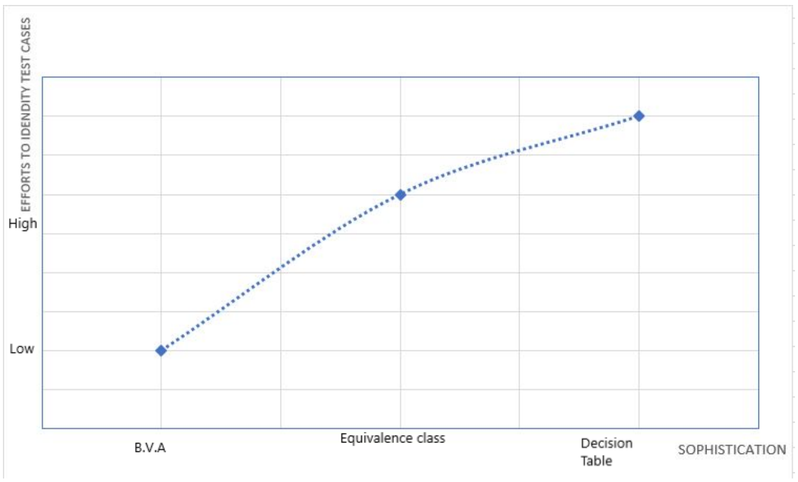


Figure8.Test cases identification efforts as per the testing method

**White Box Testing (W.B.T.)**

W.B.T. is a technique that relates to the outside functionality of the software under consideration by analysing and then test the program code. According to Roper (1994), other names of W.B.T. are clear box testing, open box testing, program-based testing, logic-driven testing or glass box testing. In this technique, the internal structure of the program or code is examined in depth. This technique uses test data to examine the program logic. As described by Desikan and Ramesh (2007), this technique takes into consideration various parameters like code structure, program code and internal design flow. White box can be applied by a programmer who has full knowledge of the program structure. In other words, only an experienced programmer is suitable to apply this technique because it can test every branch and decision in the program. Knowledge of internal structure analyses different coverage criteria quite interesting. The crucial one in this technique is decision coverage. Mitra(2011) also described that the selected test might be fruitful only if the tester knows, in prior, about the function of the program. The tester checks whether the program separates from its expected goal. W.B.T. is mainly used for spotting the logical errors in selected program code. It is used for debugging a code, finding random typographical errors and uncovering the wrong programming assumptions as described by Nidhra (2012). The classification of white-box testing is shown in fig. 8.

W.B.T. is applied at low-level design and implementable code. It is applicable at all levels of system development, especially during unit, system and integration testing. Saglietti (2008) described that W.B.T. could also be applied for other development artefacts like requirements analysis, designing and test cases.

**Static Testing**

It demands the source code of the program to check the code of the program. It requires human intelligence to do the testing, or it can be done with the help of some special tools. For static W.B.T., only trained people can be employed who can find the defects in the code. Code execution is not required in static testing. But this technique required the full involvement of testers to go through and deeply analyses the code to find the code is appropriate as per the requirement. This testing determines whether the code is robust or not. Robustness relates to the coding efficiency to handle errors. The main purpose of static testing is to check the software code to find whether the software fulfils the functional requirement and meet coding standards. It all ensures that all functionalities are covered by software or not. Testing also checks and verify the error handling mechanism of the software.

**Desk Checking** - Nidhra (2012) described, also known as primary testing performed on the code. Static checking is required to be performed by programmers before compiling or execution. If any error is found, the author must check it. It’s the author duty to correct the code. The code must meet the required specifications. The designed code must adhere to the requirements of the client. Into this process, the writers who know the programming skills closely are required to participate in desk inspection tests to be carried out. This can be achieved very rapidly without too much dependence on other developers or testers. The main advantage is the observed defects can be easily located and can be corrected as quickly as possible

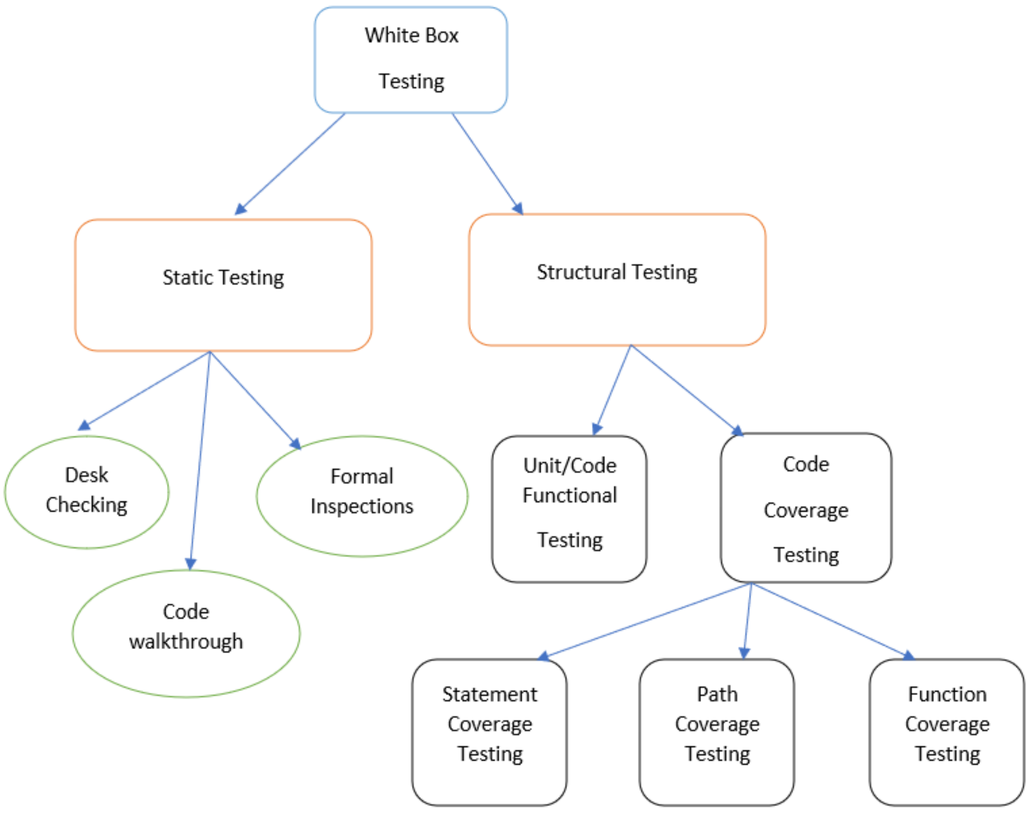


Figure 9 White Box Testing Classification

**Code walkthrough** – It is commonly called a technical code walkthrough, one kind of semi-formal review technique. Nidhra (2012) described that in this the test methodology a bunch of technological people are required. These selected must be such that they should go through the code. In the Code walkthrough process, senior employees such as technical leads, database administrators and peers must be involved.

## Structural Testing

Structural testing demands the execution of the test cases to test the software that must be tested. This is just the reverse of static testing that does not demand the execution of test cases to be run on the software. The following types of structural testing were explained by Desikan and Ramesh (2009).

**Statement Coverage Testing -** During this testing, at least once every statement of the program must be carried out. Lie (2008) called statement testing node coverage or basic block coverage. It is the common assumption that code coverage is directly and proportionally related to the testing of the functionality. An increment in code coverage will lead to better testing of the functionality. The statement coverage of the program, which is indicative of the percentage of statements executed in a set of tests, can be calculated as follow:

$$\text{Statement Coverage} = \frac{\text{Total Statements Exercised}}{\text{Total Number of Executable statements in Program}} * 100$$

Eventually, the advantage of statement reporting is its ability to recognize which blocks of code have not been executed. The issue with the statement coverage, though, is that it does not identify bugs that arise from the control stream constructs in the source code, such as compound conditions or consecutive switch labels. The major shortcoming in this technique is that it does not reflect that program is error-free even though achieving a high level in statement coverage testing. Branch Coverage Testing – Software applications are not meant to be written in a continuous mode of coding, sometimes, branching the code to perform a requisite functionality is also needed. Branch coverage testing helps to invalidate all the branches in the code. It also helps in confirming that no branching leads to abnormal behaviour of the application. In part testing, each edge is crossed at best once. Jin et al. (1995) describe that in this testing, at least once every branch and every entry point of the program or software under consideration must be executed which also involves all control transfer. Mitra (2011) described that decision or branch coverage can also be called edge coverage. A branch is the outcome of a decision. Thus, branch coverage helps in identifying which decision outcomes must be tested.

**Path Coverage Testing -** In this type of testing, a program is split into no. of distinct paths. Now, every possible path is executed. This technique has more chances to detect more errors as it attempts to find out the ratio of code coverage to great extent. A path means the flow of execution from the start of a method to the end. A method with n decisions has 2n no. of possible paths and if a loop also exists there, no. of paths increases to infinite. Fortunately, a metric known as cyclomatic complexity can be applied to reduce the no. of paths that require testing. The cyclomatic complexity can be calculated as one more than no. of unique decisions in the method. Cyclomatic complexity helps in defining the no. of linearly independent paths, called the basis set. Like branch coverage, testing the basis set of paths gives a surety that every decision outcome is tested. But unlike the branch coverage, basis path coverage takes care that all decision outcomes are tested independent of others. It also allows analysis i.e. how changing that one decision affects the method's behaviour. Path coverage testing can be called a better technique among the three techniques discussed till now. The formula used is:

$$\text{Path Coverage} = \frac{\text{Total Paths Exercised}}{\text{Total Number of Paths in Program}} * 100$$

The disadvantage associated with path testing is that it cannot be applied if there is a loop in the program.

**Condition Coverage**–Covering all paths never implies that the program is completely tested. Path testing is not enough as it does not exercise each part of the Boolean expression, relational expression and so on. This technique of condition coverage is responsible for monitoring whether every operand in complex logical expression has taken on every true and false value as described by Mitra (2011). Therefore, this means that no. of test cases rises exponentially with no. of conditions and Boolean expressions. The condition coverage for a program, which is indicative of the percentage of conditions covered by a set of test cases, is defined by the formula as:

$$\text{Condition Coverage} = \frac{\text{Total Decisions Path Exercised}}{\text{Total Number of Decisions in Program}} * 100$$

Thus, Condition coverage can be called as better criteria than path coverage criteria.

## MANUAL TESTING CHALLENGES

To test the software, test cases are written. But there is no proper mechanism to determine the validity of the test case. It is a total dependency on the understanding of the tester who may be able to understand the prescribed requirement or not. Due to this, defects might be incorporated into the system after testing also. So, testing depends a lot on the skill, understanding and efficiency of the tester.

One cannot rely on manual testing as there is no way to confirm that whether actual and expected results have been matched carefully or not. It depends on the tester. Testing is non-exhaustive. Full manual testing is impractical. Manual testing depends upon the feasibility. To resolve this issue, it is important to generate test data automatically. The process of automated test data generation helps in saving time, effort, and cost of testing of software. It reduces the manual work used in the design of test cases. The main goal of automatic test data generation is to minimize the manual work involved in testing i.e., test case design, test case execution with minimum time and maximum coverage.

## TEST AUTOMATION

Testing of programs using software is called automation. Test cases designed to test the programs are created either manually or automated. In manual testing, testing is done by a skilled tester. They designed the test case and check its validity. A test case may be valid or invalid or often, the same type of test cases may be redesigned. For a simple, small program, a lot of test cases is possible. So, selection of the right, valid, effective test case is a very important aspect of software testing. So, this is a big challenge in front of the tester to choose a unique, distinct, and valid test case. Manual testing is very time consuming, costly, and not too effective. It further involves a human skill that may be fully skilled, semi-skilled or not properly skilled. To overcome the problem related to manual testing like shortage of skilled manpower, reduce the time and cost of manual testing, it is mandatory to switch to automated software testing. With automated software testing, the overall cost of software development might be lowered drastically. Test automation provides a solution to many problems faced in manual testing

In automation, the involvement of humans is very less as test cases are executed by software which is faster in nature as compared to human beings. Moreover, test cases may be run 24×7, so automatically saving the time consumed in testing. So, the software can be released in time without delay. The time saved may be used by the engineering team to develop some extra or enhanced features of the software. There are several advantages of automated software testing over manual testing. Test automation helps the engineers to utilize their efforts in the creation or design of new features and free them from uninteresting or repetitive tasks. Testing using automatic testing is more reliable as compared to manual testing. As automatic testing started immediately after development without the need of so much skilled large team, so it is also called immediate testing. Few types of testing are not possible to execute without automation e.g., reliability tests cases, stress testing test cases.

the following metrics better the automation:

- No faults were found and their percentage
- The time involved in automation testing including every release cycle
- Minimal time is taken for release
- Consumer satisfaction index
- Production improvement

Benefits of Automation Testing are:

- Test speed
- Reliability in Test reports
- More consistency
- Less time and cost
- More accuracy
- No human involvement during execution

- Improvement inefficiency
- Test often and carefully
- Early time to market

Automation is one of the key elements of testing to reduce human intervention meant predictable, reliable, and reusable results and significant cost reduction and increased inter-supplier portability. The standards are helpful to meet the requirement and certifications [1], [2]. Test engineers must know several automated testing approaches and tools that assist test actions other than test execution [3]. Automotive electronics parts increased to improve comfort and features [4]. Because of cost, electronic parts consisted of 15% in the middle of 1990 and reached almost 20% in 2000, and they will reach 32% in 2010 [5]. As more features are adding up to the system, More ECUs are required. The complexity of programming and testing has become more challenging [6]. Hence, vehicle makers must spend more time and cost to validate the whole embedded systems.[7]. Carmakers and system vendors formed AUTOSAR in June 2003 to institute open standards for providing electric and electronic architecture [8]. EAST-ADL stands for Electronic Architecture and Software Tools is another tool for the automotive industry [9].

Image comparison transactions the resemblance between two images. It processes the identical image records from a reference image gathering to calculate the resemblance of each feature between the test image and each reference image and allocates weight to each functionality to fuse these features via the "query-adaptive weighting method." It targets the test image and its neighbourhood set selected from the repossession dataset as the test class. It utilises the image-to-class similarity to re-rank the repossession results [53]. Inserting soft-tissue patterns and interventional instruments into the image can considerably change the performance of some resemblance rates formerly used on 2D and 3D image registration [54]. Support Vector Machine (SVM) used a supervised learning method to compare and identify biomedical image processing [55]. A similarity index for images represents both pixel intensity changes and geometric alterations. Classification of many images can use these two parameters [56]. Magnetic resonance imaging (MRI) is widely used in medical investigations. Label and clustering unsupervised learning methods improve the accuracy of MRI. Image reconstruction and finely tuned parameters also improve the accuracy [57]. The adjustment interaction for the mathematical mutilation created by a wide-point lens introduces outspread twisting antiquities brought about by non-direct resampling [58]. The approval technique introduced is a significant advancement towards more nonexclusive recreations of biomechanically potential tissue distortions and evaluation of tissue movement recuperation utilising nonrigid picture enrolment. It is a reason for improving and analysing the changed nonrigid enlistment method for various clinical applications [59]. A novel method has been described to mark student attendance using image processing methods by comparing the faces of students [60]

Researchers apply natural language processing (NLP) computational techniques to understanding human language by the systems. Now, we are concentrating on the evolution of grammatical formalisms and parsing algorithms, appropriate semantic representations for word and sentence meaning. Natural language processing (NLP) is the area of artificial intelligence (AI) preoccupied with the automatic creation and understanding of human languages [61].

Theoretical issues in NLP are usually split into three areas:

- Syntax, the study of sentence structure
- Semantics, the study of context-independent meaning
- Pragmatics is the study of context-dependent meaning.

Sentence parsing has two objectives. First, a parser is used to find out influential groups of words and their organisation in a spoken or written text after the grammatical constraints of the appropriate language. Second, a parser must supply the units and identify the domains of other processes. Together, these two parser functions help to limit sentence ambiguity so that the meaning of an utterance can be recovered from the vast meaning potential of the language. Parsing concerns the incremental production of the syntactic description of texts. There are two opposing ways to view this process. It can be viewed as a series of choices provided by the grammar that results in a minimal set of alternatives. Alternatively, it can be seen as refining a description, true at each step, to arrive at an applicable syntactic description of a sentence. The distinction between these two views is simply whether alternatives are represented explicitly while parsing a sentence.

Class-based topographies expand the performance of natural language processing (NLP) tasks such as syntactic part-of-speech tagging, dependency parsing, sentiment analysis, and slot satisfying in natural language understanding (NLU), but not much has been stated on the primary causes for the performance enhancements [62].

A framework developed for testing methodology of validation of path planning and ADAS control algorithms for current and future automated vehicles [70]. Deeper is a simulation-based test generator that uses an evolutionary process, i.e., an archive-based augmented with a quality population seed, for generating test cases to test a deep neural network-based lane-keeping system [71].

## 2.3 SUMMARY

It is clear from the existing literature that most of the work-related test automation is using test management software and script development to automate manual efforts. No literature has focused on key-based fully automation from software requirements to test plan development and script generation. Hence, understating the importance of test script generation from the test plan. We also focused on image comparison algorithms which are part of the script generation library for test automation.

# Chapter 3 TEST AUTOMATION FRAMEWORK IMPLEMENTATION

## 3.1 INTRODUCTION

Embedded Software Development Life Cycle aims to deliver high-quality product software that satisfies all the requirements. It starts with planning and requirements analysis where senior people design products based on market requirements and technical experts' suggestions. Once the product design has been done, the next stage is to document SRS (Software Requirement Specification) the product requirements and get approval from the client. The SRS document decided the product architecture and was reviewed by top management for final changes. Based on the SRS document, Software developers develop the product software and develop test cases that cover all software functionalities. The software quality assurance team (SWQA) used the test cases and performed manual Regression or do automation testing. SWQA team used test management software's to automate the manual test scenarios. If any part of the software updates, the testing team must perform entire software testing to find any other aspects affected by part of the software update, called the regression test.

The industries follow any software life cycle development models, mainly waterfall model, Iterative model, Spiral model, V- model and Agile model. The waterfall model divides software development into different phases, and each phase depends on the previous phase. Moreover, it is not suitable for complex projects as it has high risks and uncertainty. The iterative model repeatedly improves the developing versions until the complete system is ready for large projects, but more resources are required. In the spiral model, the development covers three phases like identification, design, and build. Requirement changes can easily manage in the spiral model, but the process is complicated. In the Verification and Validation model, implementation of processes happens sequentially like a V-shape. Here each phase of development is directly linked with testing. The V -model is best suited for a small project where requirements are well defined, but it is not suitable for significant and ongoing projects. The agile model divides the product into minor incremental builds and a very realistic approach to software development. These builds follow the iteration method. Each iteration naturally takes from about one to two weeks. Every iteration contains sub functional teams working simultaneously on various areas.

Manual Regression is essential for software validation and is a time-consuming task. The solution is test automation only. Here we need to spend one-time scripting manual test cases using any test management software. Later, we can execute scripts many times to get the results report and reduce manual effort time. In embedded software industries, primarily using test management software are NI test stand and Dspace automation desk. The Test Stand is test management software and allows test engineers to develop automated test scripts using code modules written in any programming language. One of the popular languages is LabVIEW, and it is a graphical programming language. At the same time, the automation desk is mainly using in the automotive industry for hardware in loop test automation.

Software Testing is a crucial stage in the software development and maintenance cycle. It has been estimated that software testing involves between 30-50 per cent of software development. Depending on manual testing becomes costly, Inefficient and causes a high frequency of product release delay. We can script manual test cases using test automation tools, but this is costly, requiring highly skilled test engineers. Our framework gives the solution to this problem.

## 3.2 FRAMEWORK IMPLEMENTATION

The proposed script generation framework is divided into three phases. Phase-1 is the process of writing test cases as per the required format using a software requirements document. Phase -2 is the critical part that generates test scripts using intelligent logic. Phase -3 is about executing scripts and getting test reports. Test script execution depends on the test management software.

## 3.3 TEST CASE AUTHORING

Test case authoring plans to generate standard test cases and deals with Requirements documents, Natural language processing-based Test case generation (NLP TCG) and Test case editor. Test case editor is an interface where we can write test cases manually using software requirement documents. Test cases authoring built on pre-defined keys structure with sections. The test case editor user interface is shown in figure11.

Test steps are divided into three groups: setup, Main steps and Clean-up. The setup group covers pre-condition steps, the group of the Main steps covers the actual test steps of the test scenario, and the clean group covers the postconditions of the test scenario.

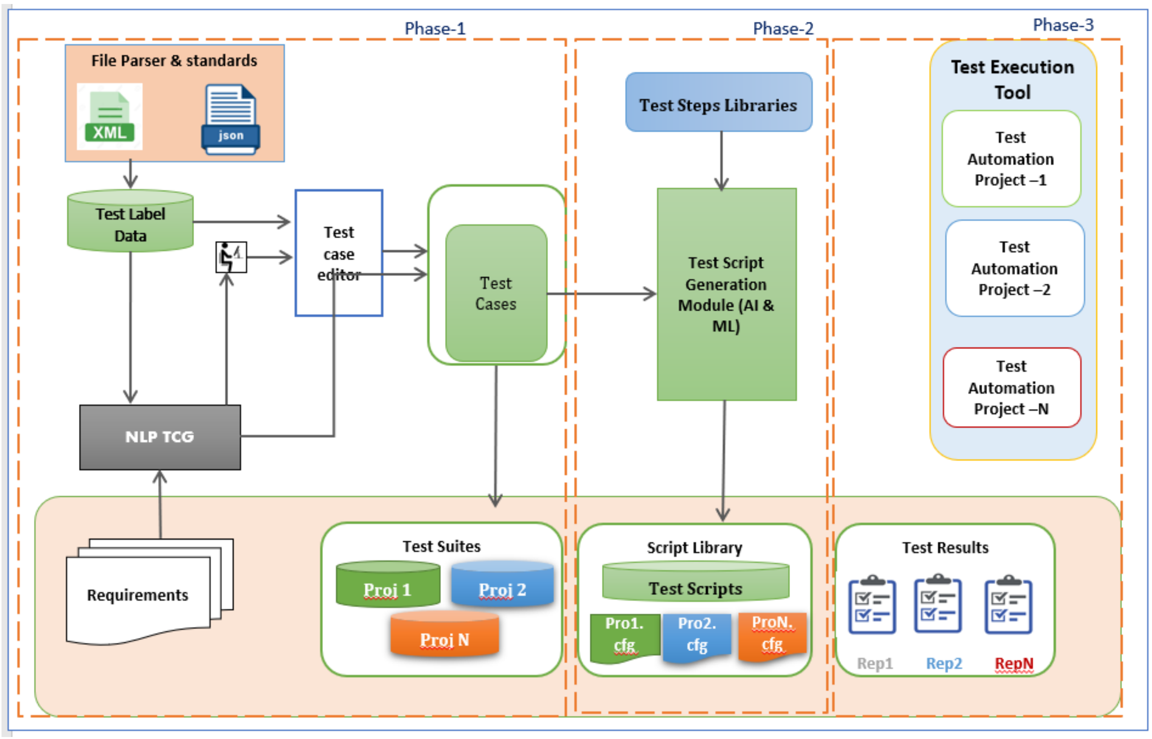


Figure 10. Framework block diagram

| **Action** | **Function/Service Type** | **Parameter** | **Value** | **Remarks** |
|---|---|---|---|---|
| Setup | | | | |
| | | | | |
| | | | | |
| | | | | |
| Main Steps | | | | |
| | | | | |
| | | | | |
| | | | | |
| Clean-up | | | | |
| | | | | |
| | | | | |
| | | | | |

Figure 11. Test case editor User Interface

Five sections were provided to write the test step to cover all functionalities.

**Action:**

Action has the following key terms to define the action of the test step. We used four key terms, write, read, set, and check to cover the significant test actions. We can add new critical terms into action depending on the requirement.

| **Action** |
|---|
| Write |
| Read |
| Set |
| Check |

Figure 12 Sample Key set at Action Section

**Function/Service Type:**

The keys belong to this section to determine the service type of test step. For example, if we use protocols like LIN, CAN, or Ethernet in software development, writing these protocol bus messages and reading from these protocol bus steps can be determined here.

| **Function/Service Type** |
|---|
| ECU Signals |
| ERD Signals |
| CAN |
| LIN |

Figure 13 Sample key set at Function/Service type section

**Parameter:**

Test parameterisation is an efficient method and utilising parameters rather than fixed values. So, one can execute the same test with different input values of test steps used as inputs or expected outputs stored separately from the test steps.

The main benefits of parameterisation are as follows:

- By externalising the changing parts of a test case(s) as parameters, it is easier to manage complex test cases
- Users can automatically execute multiple variations of each test. Multiple Parameter values provided will cause several executions of each test for each variation of the parameters.

Test Parameterisation allows sharing information between multiple test cases. In this section, we can define a parameter.

Value:

In this section, we can define the parameter's value and define the expected output value.

The remarks section can explain any additional notes, which helps consider complicated test steps.

The following example gives clarity about the test case editor.

The oven is a home appliance product used for baking food items, and given below is sample oven software requirement.

The Oven light should turn on when the oven door is open or the oven light switch is turned on. Also, the count of the oven light ON and OFF and the door open and close should be stored.

Test case writing using the proposed framework test case editor is simple and easy to understand.

| **Action** | **Function/Service Type** | **Parameter** | **Value** | **Remarks** |
|---|---|---|---|---|
| Setup | | | | |
| Set | Function | TestUnit | PowerON | |
| Main Steps | | | | |
| Read | ECU Variable | OvenDoorOpenCloseCount | X_Count | Store value into X_Count |
| Read | ECU Variable | OvenLightOnOffCount | Y_Count | Store value into Y_Count |
| Set | ECU Variable | Oven Door | Open | |
| Check | ECU Variable | Oven Light | ON | |
| Set | ECU Variable | Oven Door | Close | |
| Check | ECU Variable | Oven Light | OFF | |
| Check | ECU Variable | OvenDoorOpenCloseCount | X_Count + 1 | |
| Check | ECU Variable | OvenLightOnOffCount | Y_Count + 1 | |
| Clean-up | | | | |
| Set | Function | TestUnit | PowerOFF | |

Figure 14 Sample Testcase

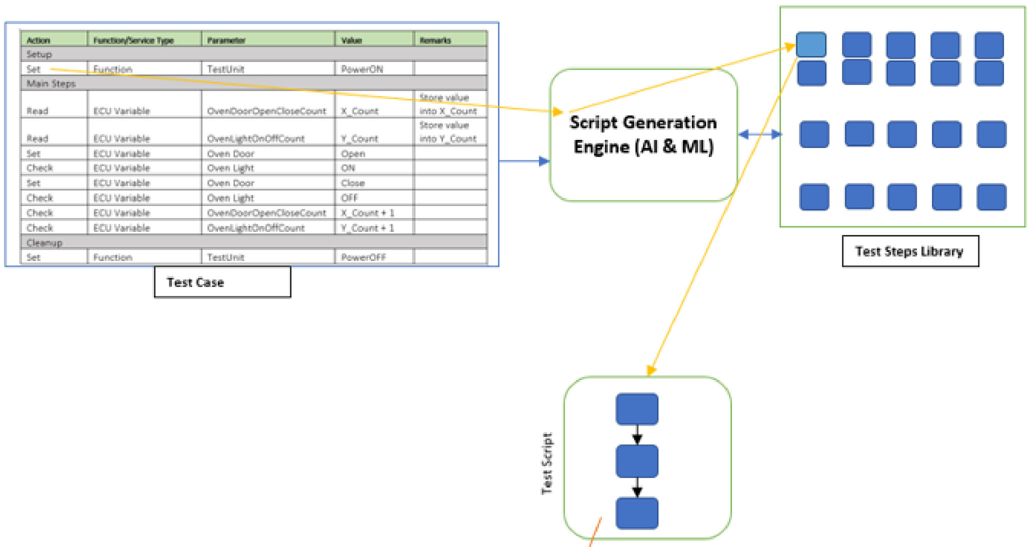
Embedded software development uses standard documents like JSON or XML files to maintain microcontroller variable data. Here we do parse these files and save them into test label data. Test label data applicable write test cases quickly. OvenDoorOpenCloseCount, Oven Door and oven light are the test labels parsed from the oven software JSON file.

## 3.4 TEST SCRIPT GENERATION

Test steps library based on test management software required to prepare before script generation. Test steps library is part test automation process. Test steps library need to fill with all basic functionality with respective test management software's code modules. The NI test stand, the LabVIEW code module, and the Dspsce automation desk python code were used to develop libraries. The test configuration is part of the proposed framework, which configure test management software.

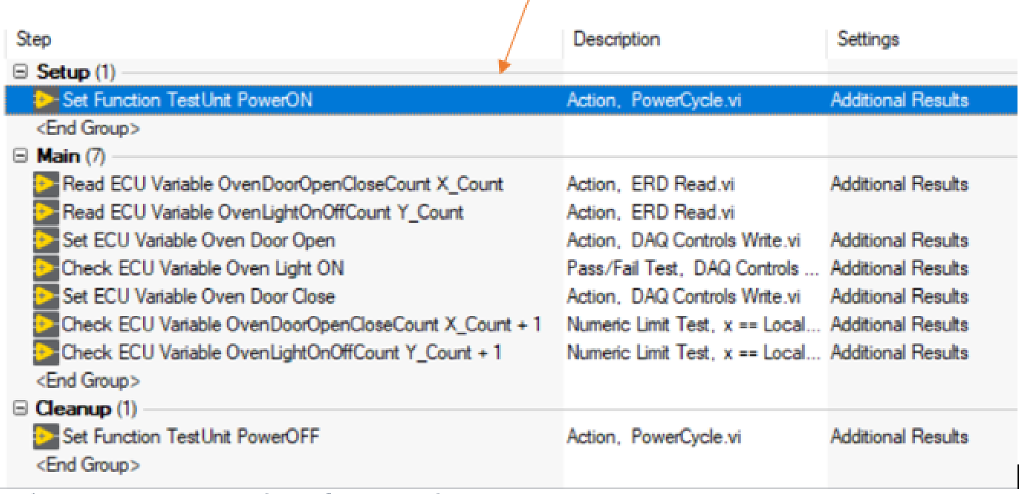
Script Generation Engine is the business logic that generates test scripts taking test cases as input and identify the functions based on key terms, then replace parameters and values.

For example, Step1 action key is Set, the Service type is Function, the Parameter is Test Unit, and Power ON. This combination identifies the test unit function and sets the write value as Power ON.



| Action | Function/Service Type | Parameter | Value | Remarks |
|---|---|---|---|---|
| Setup | | | | |
| Set | Function | TestUnit | PowerON | |
| Main Steps | | | | |
| Read | ECU Variable | OvenDoorOpenCloseCount | X_Count | Store value into X_Count |
| Read | ECU Variable | OvenLightOnOffCount | Y_Count | Store value into Y_Count |
| Set | ECU Variable | Oven Door | Open | |
| Check | ECU Variable | Oven Light | ON | |
| Set | ECU Variable | Oven Door | Close | |
| Check | ECU Variable | Oven Light | OFF | |
| Check | ECU Variable | OvenDoorOpenCloseCount | X_Count + 1 | |
| Check | ECU Variable | OvenLightOnOffCount | Y_Count + 1 | |
| Cleanup | | | | |
| Set | Function | TestUnit | PowerOFF | |

*Test Script generation block diagram*



| Step | Description | Settings |
|---|---|---|
| Setup (1) | | |
| Set Function TestUnit PowerON | Action, PowerCycle.vi | Additional Results |
| <End Group> | | |
| Main (7) | | |
| Read ECU Variable OvenDoorOpenCloseCount X_Count | Action, ERD Read.vi | Additional Results |
| Read ECU Variable OvenLightOnOffCount Y_Count | Action, ERD Read.vi | |
| Set ECU Variable Oven Door Open | Action, DAQ Controls Write.vi | Additional Results |
| Check ECU Variable Oven Light ON | Pass/Fail Test, DAQ Controls ... | Additional Results |
| Set ECU Variable Oven Door Close | Action, DAQ Controls Write.vi | Additional Results |
| Check ECU Variable OvenDoorOpenCloseCount X_Count + 1 | Numeric Limit Test, x == Local... | Additional Results |
| Check ECU Variable OvenLightOnOffCount Y_Count + 1 | Numeric Limit Test, x == Local... | Additional Results |
| <End Group> | | |
| Cleanup (1) | | |
| Set Function TestUnit PowerOFF | Action, PowerCycle.vi | Additional Results |
| <End Group> | | |

*Sample Test Case Auto-Generated script for Test Stand*

Figure 15. Block Diagram of Auto script generation

The general process takes a minimum of 6hrs to write test cases, and for scripting 8hrs.

So total time approximately 14hrs to automate this requirement testing.

Our proposed framework takes 4hrs to write test cases, and scripts generation and validation takes a maximum of 15minutes.

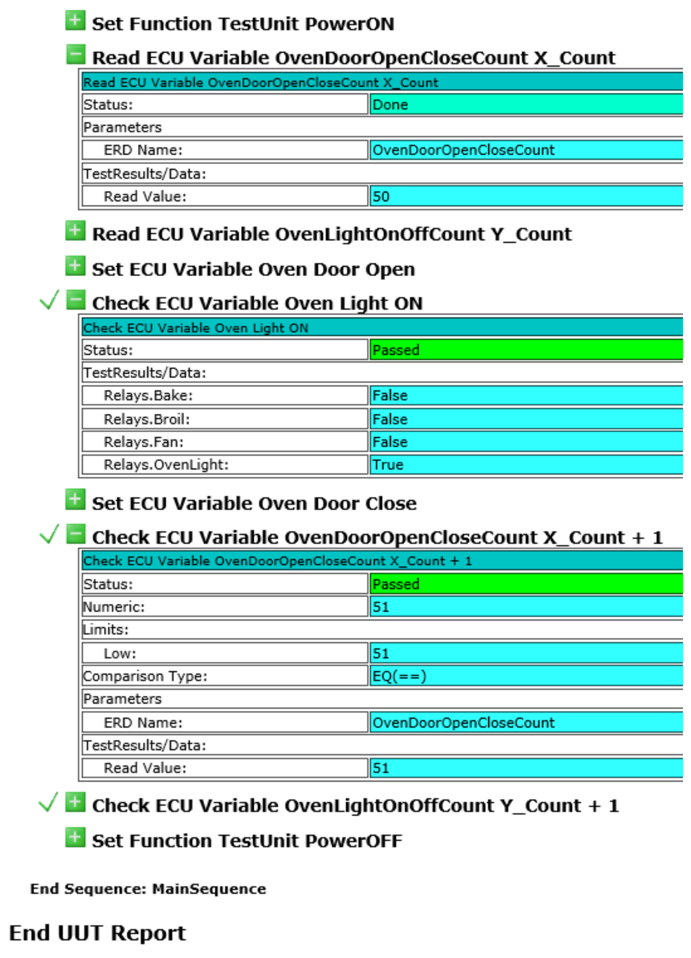
The Script generation Engine has almost had all combinations of logic to generate script generation. Moreover, it is practically not possible to write all combinations of logic. Here we can introduce AI and ML-based algorithms to predict the strange combination.

ML is part of the AI technique that will minimise the code to cover unknown cases. Supervised Machine learning algorithms are used to train the model using labelled data, such as an input where the desired output is unknown. We trained the Script generation model using the k-nearest neighbours (KNN) algorithm with pre-defined key section combinations to predict the test function from the test library for the unknown combination of input keys. However, we have not tested much prediction accuracy and precision. We will discuss the ML algorithm related information in future work.

## 3.5 TEST SCRIPT EXECUTION

The Scripts generated in phase-II are loaded into test management software used to execute the test scripts and get the automation report. We can see the sample automation report in figure 17.

As per the above table, we can save 5 hrs of time for one test case. Similarly, we can have 200 to 300 test cases for one software development depend on the project. In this way, we can save 1000 hrs to 1500 hrs of time per one software testing in 200 to 300 test cases with the proposed new test automation framework. However, this script generation has a limitation in that it cannot generate complicated test steps. In future, we are going to validate the Machine learning algorithm to get more accurate script generation.



Set Function TestUnit PowerON

Read ECU Variable OvenDoorOpenCloseCount X_Count

| Read ECU Variable OvenDoorOpenCloseCount X_Count | |
|---|---|
| Status: | Done |
| Parameters | |
| ERD Name: | OvenDoorOpenCloseCount |
| TestResults/Data: | |
| Read Value: | 50 |

Read ECU Variable OvenLightOnOffCount Y_Count

Set ECU Variable Oven Door Open

✓ Check ECU Variable Oven Light ON

| Check ECU Variable Oven Light ON | |
|---|---|
| Status: | Passed |
| TestResults/Data: | |
| Relays.Bake: | False |
| Relays.Broil: | False |
| Relays.Fan: | False |
| Relays.OvenLight: | True |

Set ECU Variable Oven Door Close

✓ Check ECU Variable OvenDoorOpenCloseCount X_Count + 1

| Check ECU Variable OvenDoorOpenCloseCount X_Count + 1 | |
|---|---|
| Status: | Passed |
| Numeric: | 51 |
| Limits: | |
| Low: | 51 |
| Comparison Type: | EQ(==) |
| Parameters | |
| ERD Name: | OvenDoorOpenCloseCount |
| TestResults/Data: | |
| Read Value: | 51 |

✓ Check ECU Variable OvenLightOnOffCount Y_Count + 1

Set Function TestUnit PowerOFF

End Sequence: MainSequence

End UUT Report

Figure 16 Test Automation Report generated by Test Stand

| Sl.No | Task | Time is taken by "Traditional Test Automation." | Time is taken by "new approach Test automation." |
|---|---|---|---|
| 1 | Test Case authoring | 4 hrs | 3 hrs |
| 2 | Test case validation by expert | 1 hr | 1 hr |
| 3 | Test Script development | 4 hrs | < 1 minute |
| 4 | Test Script validation | 30 minutes | 30 minutes |
| Total Time | | 9.5 Hrs | 4.5 hrs |

Table 5. Time Comparison Table

## 3.6 LIBRARY PREDICTION USING KNN ALGORITHM

The k-nearest neighbors (KNN) algorithm is a straightforward, simple-to-apply supervised machine learning algorithm used for a solution to classification and regression problems.

The following example gives clarity on how the unknown combination of the parameters can adapt to the nearest library code module. The reference database is auto-updated based on the known library list.

| **Action (Xi)** | **Function (Yi)** | **Paramter (Zi)** | **Code module vi** |
|---|---|---|---|
| Read | ECU Variable | OvenDoorOpenClose Count | GetECUVariable.vi |
| Read | Environmental Variable | OvenRunTime | GetEnvVariable.vi |
| Write | ECU Variable | OvenFanSpeed | SetECUVariable.vi |
| Read | PublicVariable | RTDTemp | ? |

Table 6 Reference Data

Here, Need to conisder X elements as "Read = 0" and "Write = 1". Also Y Elements as "ECU Variable= 0", "Environmental Variable = 1" and "PublicVariable = 2". Similarly, Z elements as "OvenDoorOpenClose Count = 0", "OvenRunTime = 1", "OvenFanSpeed = 2" and "RTDTemp = 3".

The following table demonstrates the numeric conversion of reference data.

| **Action (Xi)** | **Function (Yi)** | **Paramter (Zi)** | **Code module vi** |
|---|---|---|---|
| 0 | 0 | 0 | GetECUVariable.vi |
| 0 | 1 | 1 | GetEnvVariable.vi |
| 1 | 0 | 2 | SetECUVariable.vi |
| 0 | 2 | 3 | ? |

Table 7 Numeric representation of reference data

Now we have to calculate distance using the following formula.

$$Distance\ (f) = \sqrt{(Xq - Xp)^2 + (Yq - Yp)^2 + (Zq - Zp)^2}$$

| Action (Xi) | Function (Yi) | Paramter (Zi) | Distance | Code module vi |
|---|---|---|---|---|
| 0 | 0 | 0 | $\sqrt{(0-0)^2+(2-0)^2+(3-0)^2}$ = **2.6** | GetECUVariable.vi |
| 0 | 1 | 1 | $\sqrt{(0-0)^2+(2-1)^2+(3-1)^2}$ = **2.2** | GetEnvVariable.vi |
| 1 | 0 | 2 | $\sqrt{(0-1)^2+(2-0)^2+(3-2)^2}$ = **2.4** | SetECUVariable.vi |
| 0 | 2 | 3 | 0 | ? |

Table 8. Distance Calculation

If we consider K = 2, The nearest distances are 2.2 and 2.4. The most common nearest code module is "GetEnvVariable.vi". Similarly, unknown combination libraries get predicted libraries using this algorithm.

## 3.7 RESULTS AND DISCUSSION

The script generation using key parameters reduce the major time in the test script development. The test best configuration is tested with the automation desk and LabVIEW HIL setups. This research work makes a significant change in the embedded software test automation.

## 3.8 SUMMARY

This chapter deals with the major implementation work related to this research work. The test case authoring, script generation and test execution are the key elements for this research work.

# Chapter 4 IMAGES COMPARISON ALGORITHM IMPLEMENTATION

## 4.1 INTRODUCTION

Test automation is one of the essential aspects of the verification and validation of embedded software. Test automation plays a crucial role to meet the deadlines to launch a product. In terms of embedded software, display (graphical) validation is a bit complex to automate compared to text or numeric testing, where references are available for making comparisons. In the work presented, we addressed the problem of comparing digital images to find the similarity which can facilitate the embedded software test automation. Embedded testing automation is widely used to reduce development time and cost. However, still, we are not able to achieve 100% test automation due to dependency on a few manual observation aspects like the display output. Most of the embedded products have output displays that produce text, symbols or images and their testing is conducted manually. The manual observation involved can be automated with the presented work to reduce significant testing time and reduce paradox errors.

Advanced features are being rapidly implemented into embedded software in automotive, home appliances, transportation, medical equipment, communication, the energy sector. One upfront way to increase product reliability is to increase the testing on all the supported features by expanding the test effort within deadlines. The time and cost will also increase when we increase testing methods which cause product delay. That is why industries are looking for test automation wherever it is possible. Testing automation involves converting manual test cases into automation scripts that generate test results report. On the opposing, Manual Testing is conducted by a test engineer seated opposite a Test setup, cautiously performing a test steps. And record results Manual testing takes a substantial volume of time. Test automation can run as many times as possible within the deadline.

## 4.2 IMPLEMENTATION

The objective of this methodology is to determine the two input images are similar or not. The proposed algorithm has been implemented using Python OpenCV and LabVIEW platforms. LabVIEW is a graphical programming language, and it has its vision Development toolbox that offers hundreds of functions for developing and deploying image processing applications. However, it is an additional cost that is why we called python code into a LabVIEW program. OpenCV is open-source and has powerful image processing libraries.

The test image is grabbing with a digital image grabber from the device under test. Moreover, the reference image is the expected output from the device under test.

### Image processing Algorithm:

Figure 17 shows the flow chart representation of the image processing algorithm deployed in the presented study.

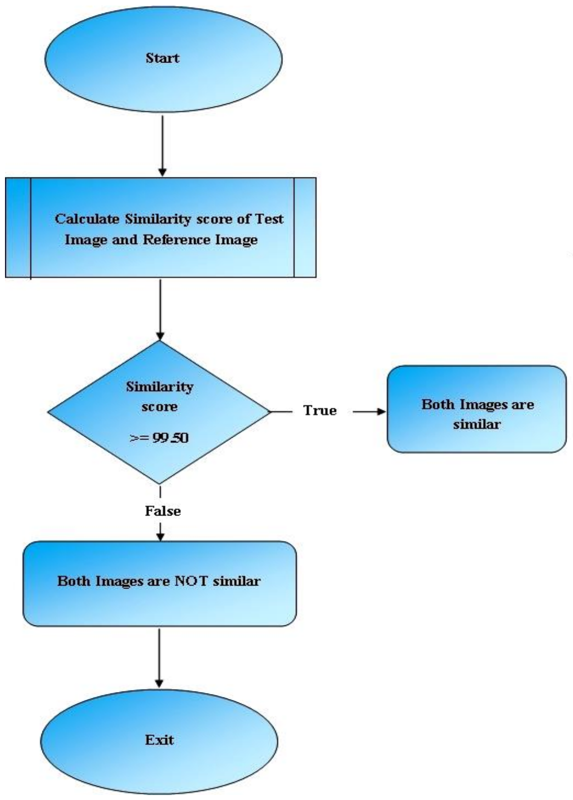

Start
Calculate Similarity score of Test Image and Reference Image
Similarity score >= 99.50
True
Both Images are similar
False
Both Images are NOT similar
Exit

Figure 17 Flow chart

In flowchart, we calculate similarity score of test image and reference image in step1. Based on similarity score we decide both images are same or not in further steps. Here considered both images are same only if the similarity score is greater than 99.50. Similarity calculation explained in the following steps.

The Similarity Score algorithm has the following steps:

**Step1:** Read Test Image and Reference image

Test_Img = T (R, G, B)

Ref_Img = R (R, G, B)

**Step 2:** Calculate Stand deviation and mean of the test image and reference image.

TestImagechnmean ($\overline{T}$) = $\frac{\sum T_i}{N}$

RefImagechnmean ($\bar{R}$) = $\frac{\sum R_i}{N}$

Teststdev ($\sigma_T$) = $\sqrt{\frac{\sum_{i=1}^{n}(Ti-\overline{T})^2)}{N-1}}$

Refstdev ($\sigma_R$) = $\sqrt{\frac{\sum_{i=1}^{n}(Ri-\bar{R})^2)}{N-1}}$

**Step3:** Apply the standard deviation formula to the test image and the reference image.

TestStdImg = ((($\sigma_T$>=cth) |(($\overline{T}$)<=bth)) *-1) +((($\sigma_T$) <cth) & ($\overline{T}$)>bth)) *1)

RefStdImg = ((($\sigma_R$>=cth) |(($\overline{R}$)<=bth)) *-1) +((($\sigma_R$) <cth) & ($\overline{R}$)>bth)) *1)

where cth = colour threshold and bth = black threshold.

**Step4:** Multiply both standard images.

3D Array of Image (AM) = (TestStdImg) * (TestStdImg)

**Step 5:** Take the sum of.AM rows to get a 1D array.

1D Array Sum (AS1) = Sum (AM)

**Step 6:** Take the sum of all elements of the AS1 array.

AS2 = Sum (AS1)

**Step 7:**Calculate the similarity score using the formula

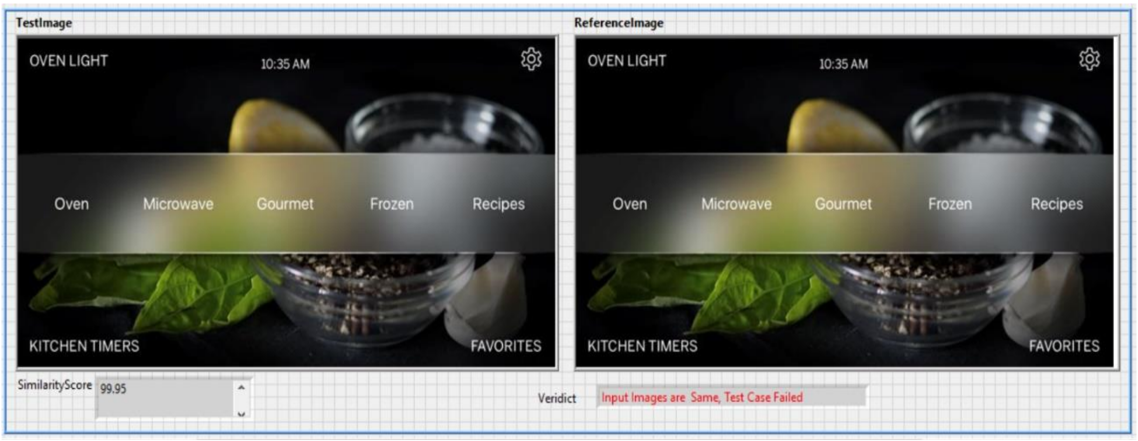
Similarity Score = $\frac{AS2}{(Image\ array\ height)*(Image\ array\ width)} * 100$

**Step 8:** Exit

The above algorithm was implemented in Python using OpenCV libraries and called in the LabVIEW program to visualise the results. This similarity score represents the percentage of similarity of the test image and reference image. The similarity score is less than 100% because of noise and errors. Hence tolerance of 0.5% has been considered in accepting the results.

## 4.3 RESULTS AND DISCUSSION

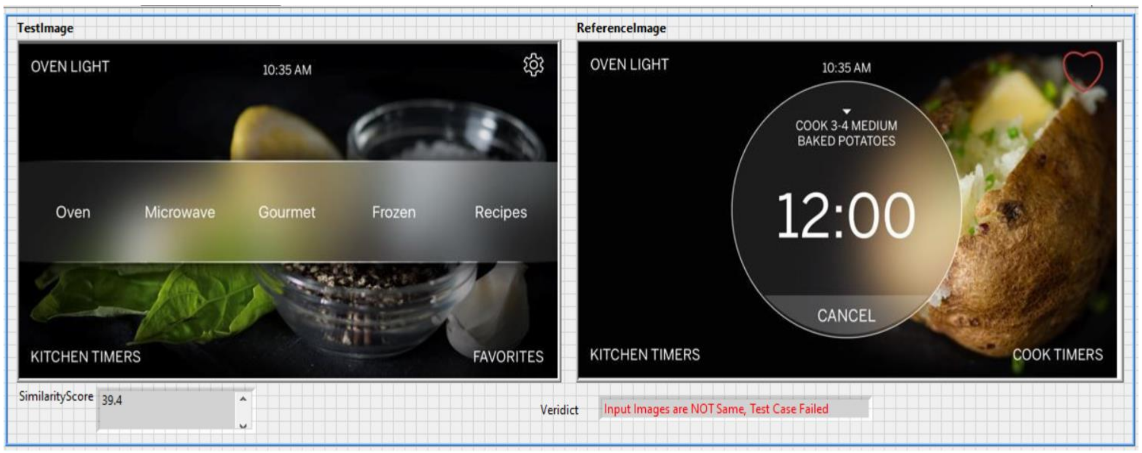
The algorithm was tested on 300 sample images, and the observed accuracy is more than 70%. With proper adjustments, the accuracy can be improved up to 90%. Case 1 results are shown in figure 2, where both images are the same. Case 2 results are shown in figure 3, where both images are not similar. Case 3 results are shown in figure 4, and both images are the same. Case 4 results are shown in figure 5, and here both images are not similar but greater than 99.5 %. Replica product recognition, image clustering, visual exploration, and recommendation tasks are performed with this technology in modern applications. Table 1 summarises the results achieved by implementing the algorithms for the test image dataset.



SimilarityScore 99.95

Veridict Input Images are Same, Test Case Failed

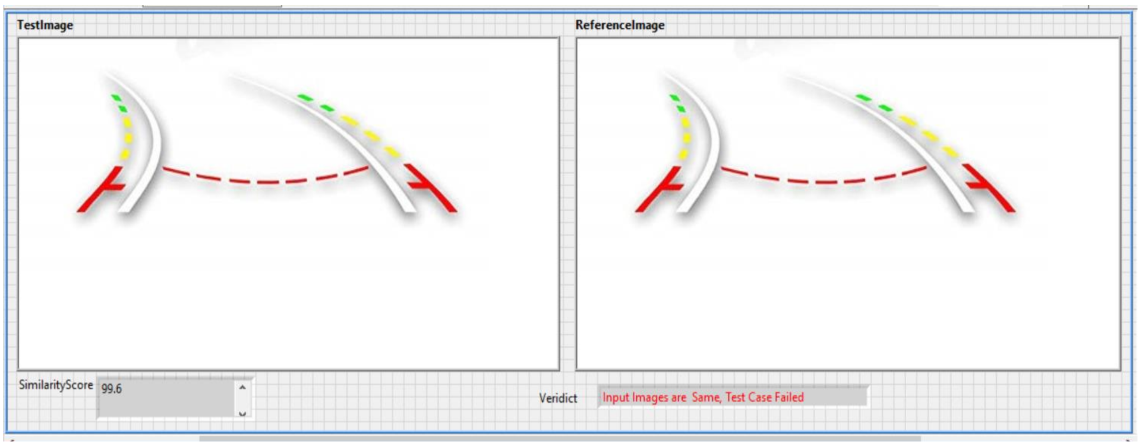
Figure 18 Case 1 Results with similarity score is above 99.5



SimilarityScore 39.4

Veridict Input Images are NOT Same, Test Case Failed

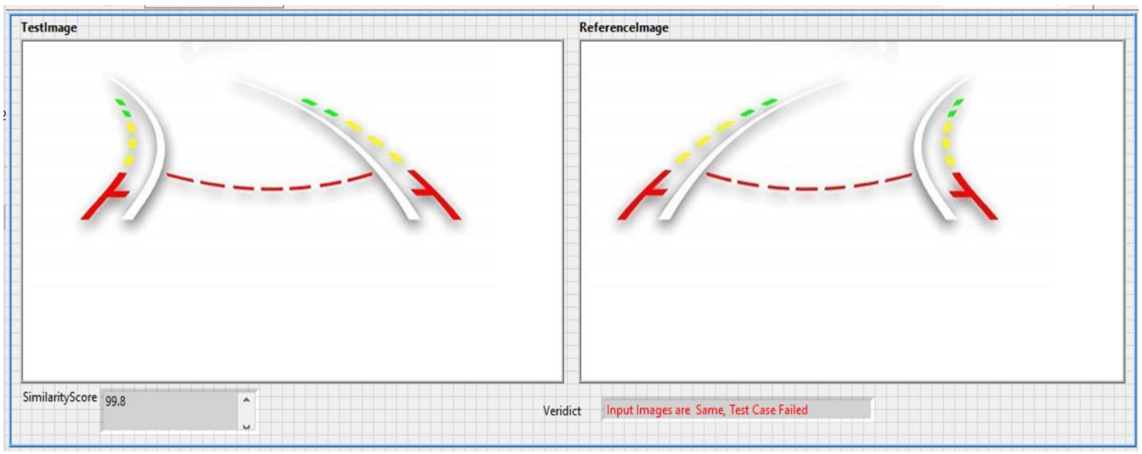
Figure 19 Case 2 results with a similarity score of less than 99.5



SimilarityScore 99.6

Veridict Input Images are Same, Test Case Failed

Figure 20 Case 3 Results with similarity score is above 99.5



SimilarityScore 99.8

Veridict Input Images are Same, Test Case Failed

Figure 21 Case 4 Results with similarity score is wrong

| S. No | Image category | Images Source | Total Samples | Accuracy at 99.5% similarity | Accuracy at 98% similarity |
|---|---|---|---|---|---|
| 1 | Automotive rear camera images | Google images | 250 | 90% | 80% |
| 2 | Home appliance: Oven Display images | Google images | 10 | 90% | 50% |
| 3 | Outdoor images captured by DSLR | Clicked with Canon DSLR | 10 | 70% | 20% |
| 4 | Night vision images | Google Images | 20 | 10% | 5% |
| 5 | Cartoon images | Drawn using Microsoft paint | 10 | 70% | 70% |
| 6 | Human faces | Google images | 10 | 70% | 70% |

Table 9 Result summary

As per case4 results, this algorithm has shown wrong results in such a scenario. This algorithm, with certain limitations, can be used in embedded software display test automation and which saves a significant amount of software testing time. Similarity tolerance and colour threshold values affect the accuracy of results. However, this algorithm has limitations and adding machine learning methods will improve accuracy. In the future, we can train the model with more reference images using machine learning techniques to determine similarity more accurately. Machine learning algorithms help in improving the testing accuracy and are helpful in test automation where the input is a digital image. This reduces the time and cost for test automation which enables the industries to launch the product as per plan and schedule.

## 4.4 SUMMARY

This chapter gives the novel image processing algorithm to compare test image and reference image which is used to automate the embedded vision software testin

# Chapter 5 IMAGE COMPARISON USING OCR TECHNIQUES

## 5.1 INTRODUCTION:

Every embedded product is getting updated with advanced features every year. Because of this reason, embedded software verification became significant and critical. Testing all functionalities together is a big challenge, which takes a considerable amount of time. Test automation helps to fast up all large test scenarios. However, due to technical challenges, some of the advanced features cannot automate, for example, LCD image verification. LCD image verification is about verifying the expected output image on the LCD screen. Automotive infotainment is a common feature in all automotive embedded software and has improved functionality to support all mobile OS like iOS, Android. These features need to monitor on the LCD screen as per the functionality. Image processing algorithms are practical to compare reference images and test images on the LCD screen. Nevertheless, the accuracy level of these algorithms is not as expected. This work gives a solution to this problem statement. This work presented a novel solution to compare test images and reference images using OCR techniques to get more accurate embedded software testing.

Embedded software impacts the quality of a product and in turn, most of the aspects of our society [1]. Its effectiveness is critical for many commercial and societal actions. As necessary, assuring the precision and quality of software systems and components becomes dominant. Embedded products (e.g., Automotive, Appliances, Medical devices) come with user interfaces where LCD, LED and other display devices are used to visualise the output for using them effectively. Users can select options from graphical displays based on the product features. All these aspects increase the product functionality along with the probability of bugs in software. The quality assurance (QA) team tests each of the features in detail to improve the software quality. However, as the number of test cases increases, the QA team needs more time, which leads to delay in the product release. Manual testing involves observing the expected results by human testers and reporting the results. Test automation is the process of converting manual observation into machine attainable by writing a programming script [2]. Once the automation script is ready, we can execute the procedure 'n' number of times, which helps to speed up the regression test. An image processing algorithm makes it helpful to automate manual processes like validating the expected image on the display units.

Digital image processing is about applying mathematical analysis to each pixel to find the required information about the image. A Digital image is a two-dimensional array considered with the Function of x horizontal coordinates and y vertical coordinates. At these coordinates, each combination has a value that is called pixel value. The digital images taken at different time frames have different intensity values. For this reason, identifying the similarity between the two images with an absolute difference is not accurate.

The Optical character recognition OCR is a typical image processing algorithm practical to extract text from digital images. This algorithm combines the mechanical and electrical transfer of scanned images of handwritten, typewritten text into software text. An OCR algorithm is the extraction of features based on pattern recognition.

## 5.2 IMPLEMENTATION

A novel algorithm modification of ABBYY FineReader has been proposed to recognise optical characters for test automation. The algorithm extracts text from a selected region of interest of the test image. The OCR algorithm was enhanced with training fonts used in embedded vision software. This program has been tested on Automotive Infotainment software connected to Apple Care. The flow chart in figure 22 is based on the algorithm implementation.

Image acquisition is the process of grabbing an image from a test setup using the industrial camera. The output of this step is a digital RGB image, as shown in figure 23.

Once the image is acquired, the vision algorithm processes the text extraction. Initially, the image is converted to grayscale, then the smoothing Filter is applied with the local average to remove the noise. Here the kernel size is 3*3.

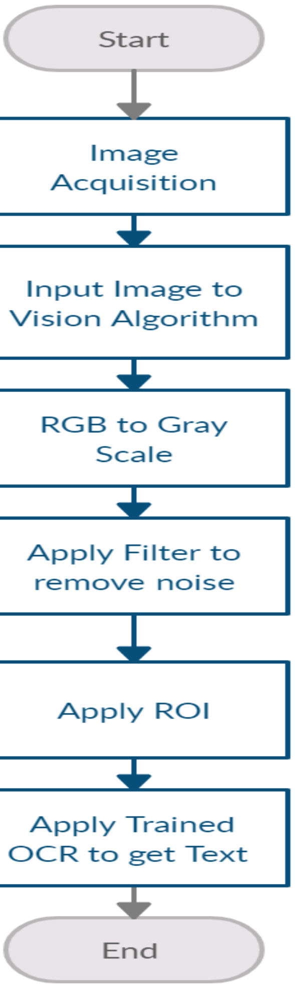

Start
Image Acquisition
Input Image to Vision Algorithm
RGB to Gray Scale
Apply Filter to remove noise
Apply ROI
Apply Trained OCR to get Text
End

Figure 22 Flow chart

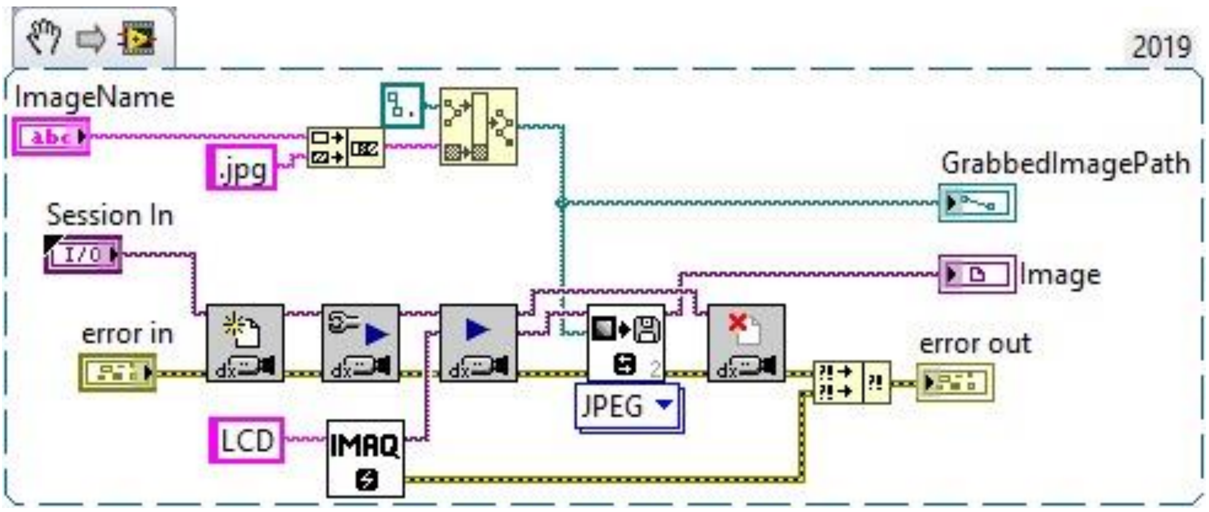
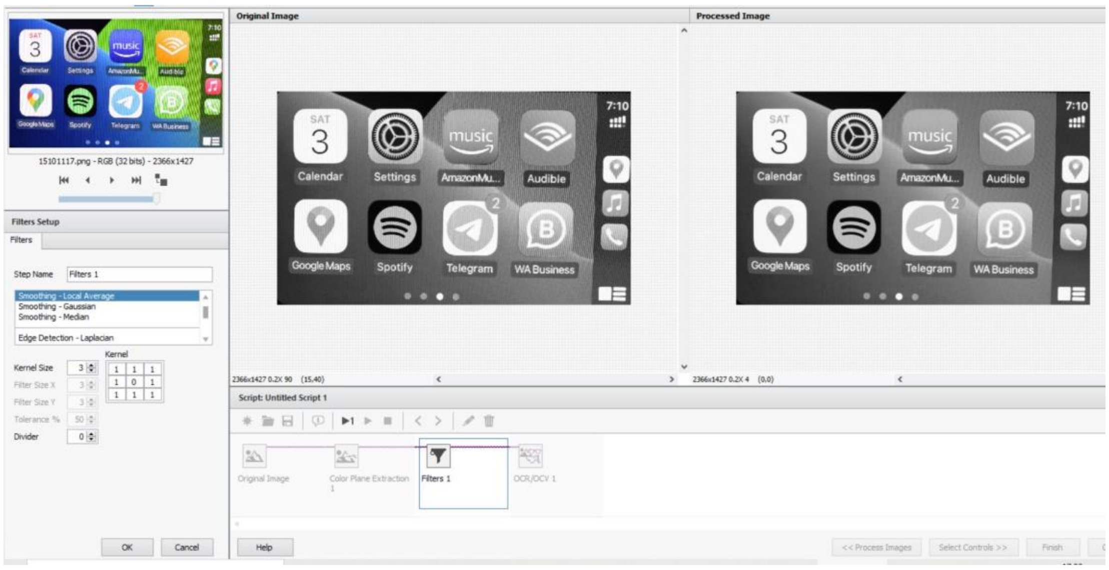


Figure 23 Grabbing Image

This Filter reduces the measure of the forced variety between a pixel and its next following pixel. Mean filtering replaces each pixel esteem in the picture with its neighbours' mean ('average') esteem, including itself. Figure 24 shows the output of the smoothing Filter.



15101117.png - RGB (32 bits) - 2366x1427

Step Name: Filters 1

Smoothing - Local Average
Smoothing - Gaussian
Smoothing - Median
Edge Detection - Laplacian

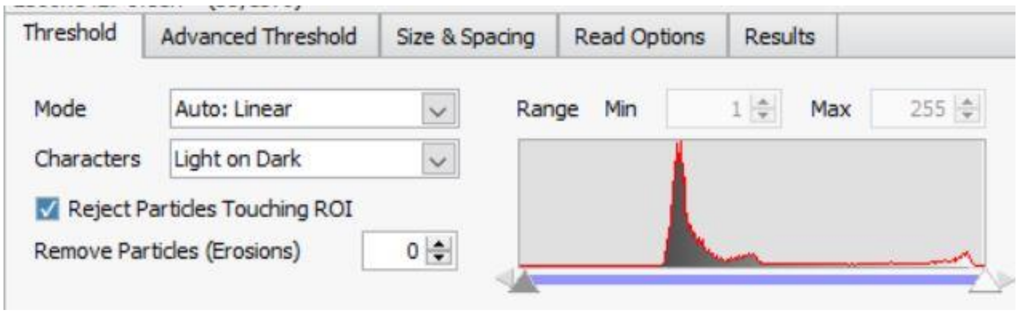
Kernel Size: 3
Filter Size X: 3
Filter Size Y: 3
Tolerance %: 50
Divider: 0

Kernel

|  |  |  |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

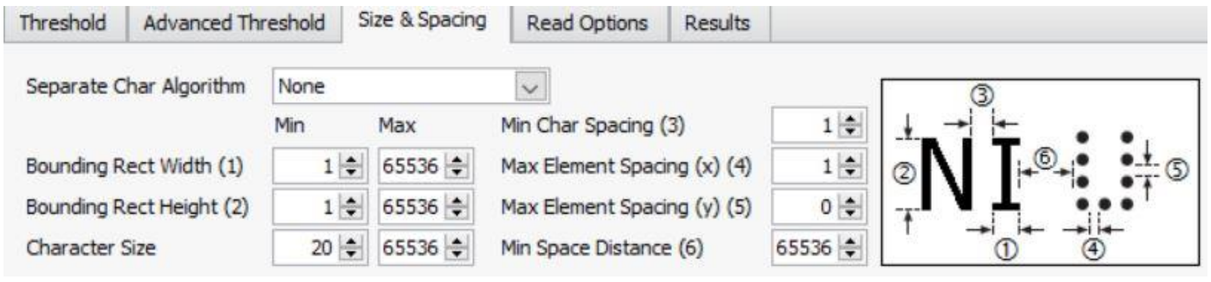
Figure 24 Image processing with smoothing Filter

After the smoothing process, the algorithm considers the ROI (region of interest) gas per the requirement—the threshold configuration represented in figure 24. The auto linear mode is selected by applying the character's light-dark background—the spacing and size configuration represented in figure 25. Flow chart steps implemented and shown in figure 27.



Mode: Auto: Linear
Characters: Light on Dark
☑ Reject Particles Touching ROI
Remove Particles (Erosions): 0
Range Min: 1 Max: 255

Figure 25 OCR algorithm threshold configuration


Threshold | Advanced Threshold | Size & Spacing | Read Options | Results

Separate Char Algorithm: None

| | Min | Max | | |
|---|---|---|---|---|
| Bounding Rect Width (1) | 1 | 65536 | Min Char Spacing (3) | 1 |
| Bounding Rect Height (2) | 1 | 65536 | Max Element Spacing (x) (4) | 1 |
| Character Size | 20 | 65536 | Max Element Spacing (y) (5) | 0 |
| | | | Min Space Distance (6) | 65536 |

Figure 26 Spacing and Size configuration

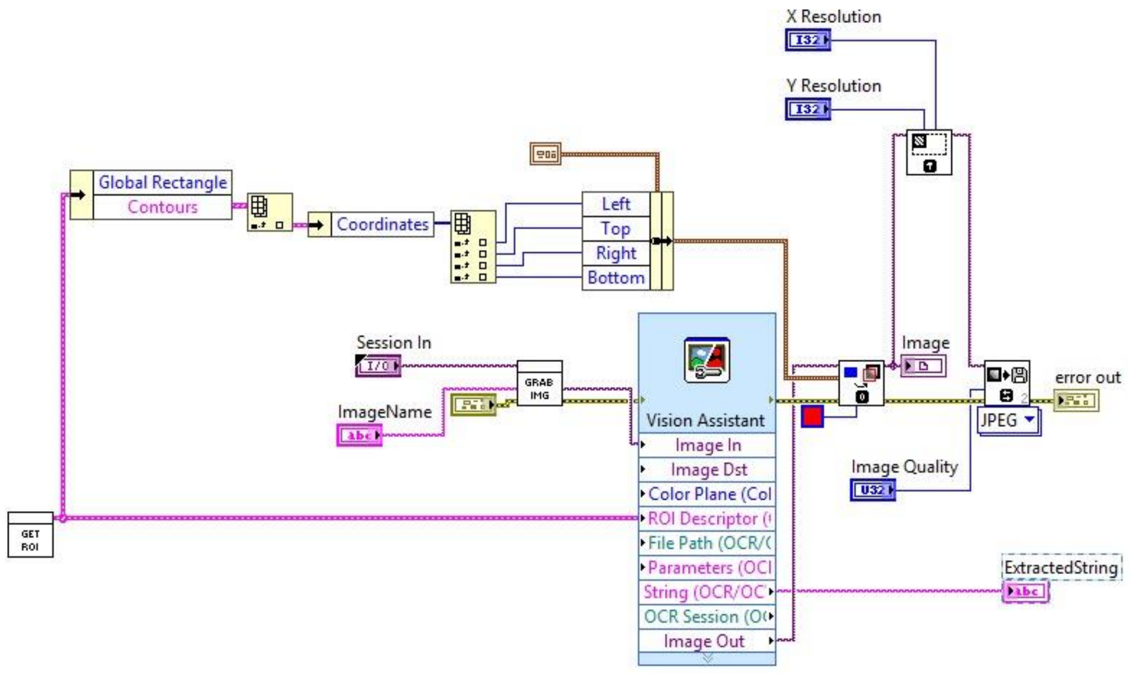
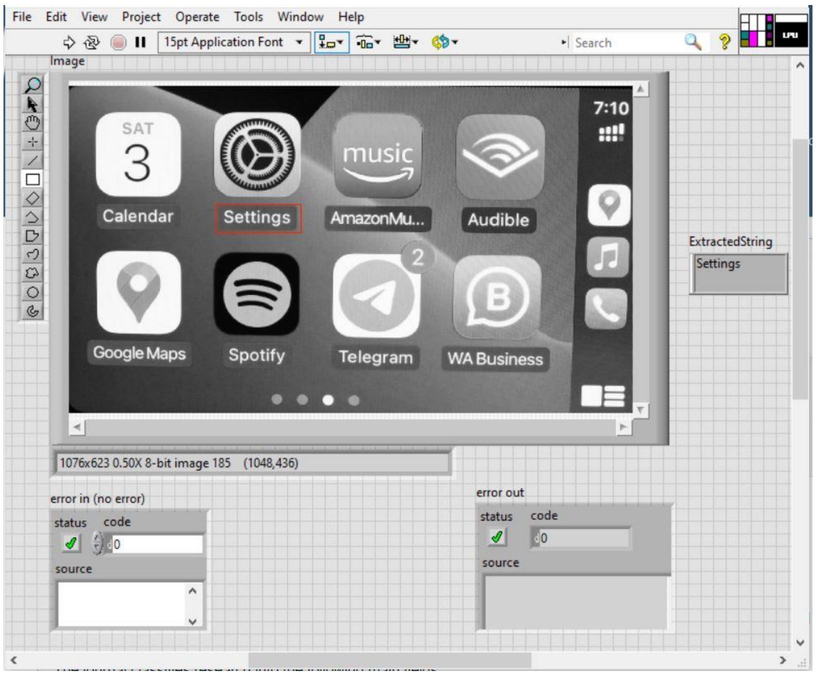


Figure 27 Block diagram of code

The results and front panel are shown in figure 28. The region of interest ROI is highlighted with red colour which can easily track and check with results. The algorithm extracted the text as "Settings" from the selected ROI and shows complete result analysis in figure 29. Figure 30 shows other ROI results. The sample training with the “S” character is shown in figure 31.



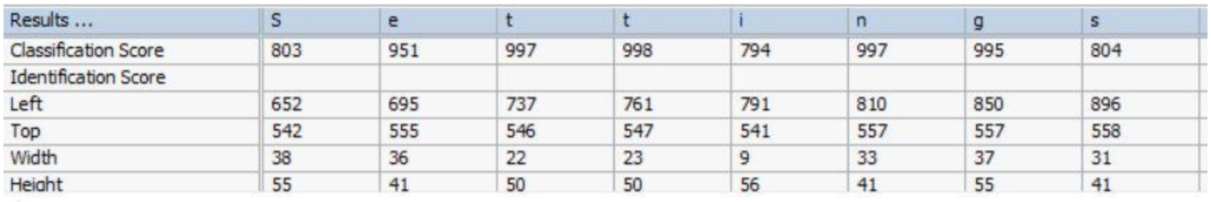
ExtractedString
Settings

1076x623 0.50X 8-bit image 185 (1048,436)

error in (no error)
status code
✓ 0
source

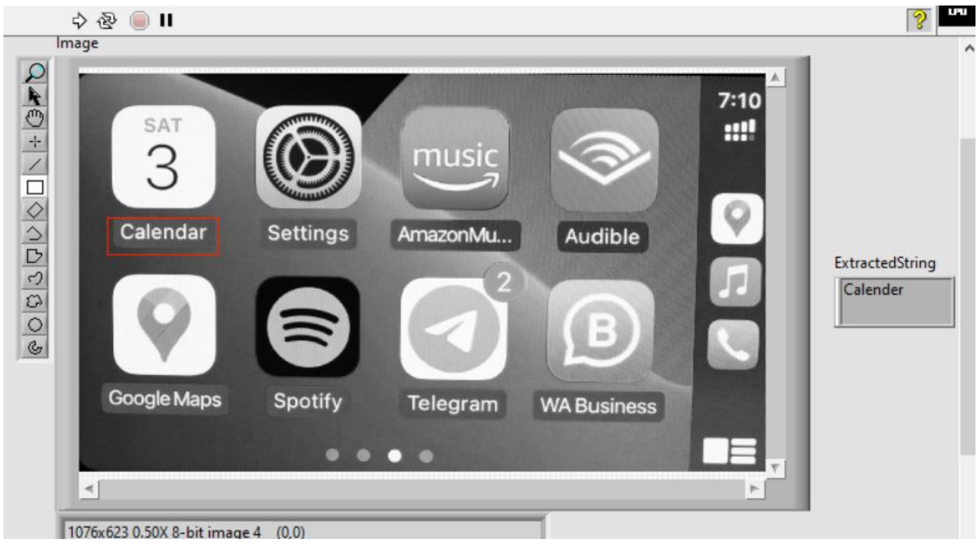
error out
status code
✓ 0
source

Figure 28 Front panel with result1



| Results ... | S | e | t | t | i | n | g | s |
|---|---|---|---|---|---|---|---|---|
| Classification Score | 803 | 951 | 997 | 998 | 794 | 997 | 995 | 804 |
| Identification Score | | | | | | | | |
| Left | 652 | 695 | 737 | 761 | 791 | 810 | 850 | 896 |
| Top | 542 | 555 | 546 | 547 | 541 | 557 | 557 | 558 |
| Width | 38 | 36 | 22 | 23 | 9 | 33 | 37 | 31 |
| Height | 55 | 41 | 50 | 50 | 56 | 41 | 55 | 41 |

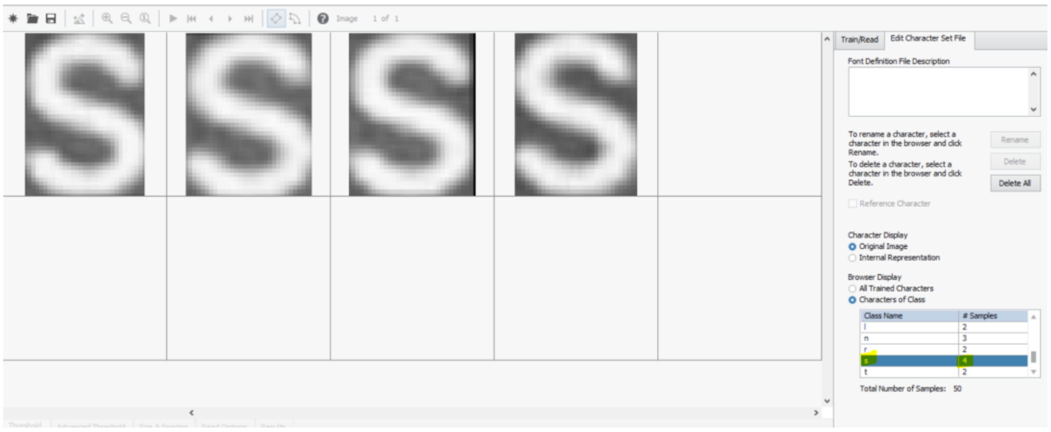
Figure 29 Result1 score details



ExtractedString
Calender

1076x623 0.50X 8-bit image 4 (0,0)

Figure 30 Front panel with the result



Font Definition File Description

To rename a character, select a character in the browser and click Rename.
To delete a character, select a character in the browser and click Delete.

Reference Character

Character Display
[✓ Original Image]
Internal Representation

Browser Display
All Trained Characters
[✓ Characters of Class]

| Class Name | # Samples |
|---|---|
| l | 2 |
| n | 3 |
| r | 2 |
| s | 4 |
| t | 2 |

Total Number of Samples: 50

Figure 31. Sample trained character "s"

## 5.3 RESULTS AND DISCUSSION

ABBYY FineReader is more accurate than Tesseract on images with only standard font text fields or only special font. Especially on the latter, on which Tesseract results have approximately 2.7 average Levenshtein distance per correct character. On the other hand, Tesseract handles images with only handwritten font text fields better. GOCR performs much worse than the others on all values, finding a lot more text in images than they contain, but performs better on special and standard fonts than handwritten. The reason GOCR gets such high results is that it finds characters in details in the background.

| Tesseract | ABBYY FineReader | GOCR |
|---|---|---|
| 90% | 92% | 75% |

Table 10 Accuracy Comparision of different algorithm

This work improved the accuracy of test automation results for image comparison in embedded vision software. Instead of comparing pixels, extracting the expected string or icon from the region of interest can give more accuracy. As increasing the features of infotainment in automotive and other industries, this algorithm helps to fast up the software testing. This traditional pixel-perfecting method and the OCR method combined gives better results based on the requirement.

## 5.4 SUMMARY

This chapter gives the alternate method to compare test images and test images for embedded vision software test automation. The OCR method is suitable for different languages so that the scope of this work is across all languages.

# Chapter 6 REQUIREMENTS TO TEST CASES GENERATION USING NLP

## 6.1 INTRODUCTION

Data Science usage progressively advanced without wanting to change the existing hardware. Also, Artificial intelligence (AI) Algorithms can apply to suitable processors and operating system computers. Natural Language Processing (NLP) is one of the subsets of AI that adds intelligent features like human’s ability to read and understand. The NLP-based application has already been used and applies to medical, media, financial, human resources, and more. Most of the Data is found in text format, which includes symbols and few other things. Since some data contains rich information, it is a need to extract data and use it accordingly. To complete it, this requires that NLP can analyses our data and perform functions such as dynamic analysis, cognitive facilitator, time filtering, targeting misinformation, and real-time language translation. To achieve the best results, the NLP system must have a high-level algorithm to enhance a good understanding of text and symbols. There are many methods available to assist the system to better understand text and icons. Text separation, semantic vector, word embedding, possible language model, sequential labelling, and word editing.

Text classification processes the text and divides it into similar groups of words. Text classification algorithms process the input text then assign a set of pre-defined tags and symbols. NLP is applied for emotion analysis, topic discovery, and language recognition. There are mainly three text classification methods rules-based, machine system, and hybrid.

In the rule-based method, texts are divided into a labelled group using a set of handicraft linguistic rules. Those handicraft linguistic rules contain users to define a list of words that are categorised by clusters. For example, words like Narender Modi and Rahul Gandhi would be considered into politics group. People like Sachin and Dhoni would be considered into the sports group. A machine-based classification system learns to make a classification based on historical observation from the data sets. User data is prelabelled as tarin and test data. It collects the classification strategy from the past inputs and discovers it continuously. The third method of text classification is the Hybrid Approach. Hybrid based method is the use of the rule-based system to create a tag and use machine learning to train the system as well as create a rule.

NLP is a subset of artificial intelligence that deals with machines and human language. How to program machines to understand and process the natural language data. In the product life cycle development, mostly the test cases developed by software developers using software requirement documents. This work involves understanding natural language and must derive a logical requirement to test each feature of the software.

The complexity of software installed in sensitive security environments, e.g., automotive and avionics, has grown significantly over the years. In such cases, soft material testing plays an important role in ensuring that the system meets all operational requirements. Such a system test task is often referred to as an acceptance test. Contrary to validation, which aims to detect bugs, acceptance testing is a confirmation function performed at the end of a life of developmental life to show compliance with requirements. Acceptance testing is mandatory and regulated by international standards. For example, the balance between the requirements and conditions of acceptance testing is enforced by the standards of embedded security systems. Exploratory cases are found in the occupational safety requirements expressed in the Nature language. Sometimes experimental cases are also found using rational mathematical models. Using a System Model Case System, Acceptance Tests Generation (UMTG) certifies the implementation of practical, systematic test cases, which accepts acceptance from the definition of needs in the native language, and reduce the efforts required to generate test cases and ensure coverage requirements. Several methods can produce test cases including input data right from NLP requirements specification. Token-making, branded business acknowledgement, and POS tagging are well-known in the software engineering community because they have been adopted by many methods including NLP. Business recognition and POS marking are often used in the software as they have been implemented by many methods including NLP techniques. Transforming needs into Work drawings, Statecharts and sequence diagrams, UML-supported models help generate test cases. Linear Discriminant Analysis is a powerful method for data category and reduction. It is helpful where the within-class frequencies are disproportionate, and their routines have been estimated on at random produced test data.

A context-free grammar is CFG is a type of grammar which is finding the relation between the alphabets how we are going to create some kind of a superset from the set of grammar rules it is a language altogether that is known as context-free grammar which is generally used in systems because system understands context-free grammar more easily than English words. Syntax is the study of rules governing the way words are combined to form sentences in a language. Sentences are composed of discrete units combined by rules. So whenever you create a sentence there's always some rule that you need to start with some identifiers may be the is a then you have a certain verb or a noun coming into the picture then you have a certain verb coming into the picture then maybe some adjectives comes into the picture. so there are rules for creating a sentence. you cannot create a sentence without any rules. we have to have some rules like we have to specify some noun then work then adjective then prepositions. so these rules are called syntax. this study of rules governing the wave words is combined. sentences are composed of discrete units called rules. every sentence has a certain rule in it whether it's a past continuous or present continuous or present perfect or past perfect or whether it's a simple sentence simple tense so we have certain rules for defining a sentence and that is defined by a syntax.

## 6.2 IMPLEMENTATION

The proposed method is shown in fig.1 and it takes input as a requirement in the English language and gives the output as test cases.

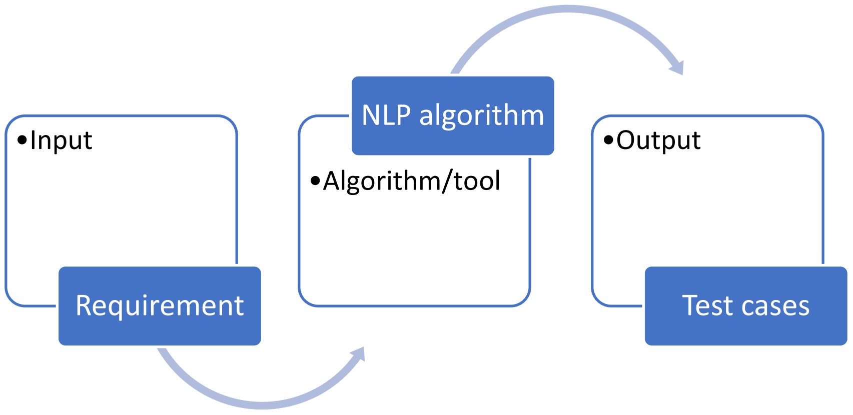


Fig. 32. Block diagram of the proposed system

Software Requirement: After The Oven Power is on, If the Oven door open, Oven Light must turn on and must record the oven light on/off count in OvenLightOpenCloseCount. The Oven Door open/close count must record in OvenDoorOpenCloseCount.

Tokenization is the NLP algorithm that converts the large text into small sections called tokens. Those tokens are extremely beneficial for discovering patterns and are regarded as a ground step for halting and classification.

Natural Language Toolkit is a powerful module NLTK tokenize sentences which added includes sub-parts

- sentence tokenize
- word tokenize

**Tokenization of Sentences**

Tokenization of sentences divides the large paragraph of text into small sentences. The below example shows how the NLTK tokenizer does the tokenization of sentences.

```
from nltk.tokenize import sent_tokenize

text = " After The Oven Power on, If the Oven door open, Oven Light must turn
on and must record oven light on/off count in OvenLightOpenCloseCount. The
Oven Door open/close count must record in OvenDoorOpenCloseCount."
```

```
print(sent_tokenize(text))

Output: [‘After The Oven Power on’, ‘If the Oven door open’, ‘Oven Light mu
st turn on’, ‘must record oven light on/off count in OvenLightOpenCloseCoun
t’, ‘The Oven Door open/close count must record in OvenDoorOpenCloseCount’]
```

### Tokenization of words

**Word_tokenize()** method used to split a sentence into words. The output of word tokenization is to Data Frame for better text identifying in machine learning applications.

```
from nltk.tokenize import word_tokenize
text = " After The Oven Power on, If the Oven door open, Oven Light must tu
rn on and must record oven light on/off count in OvenLightOpenCloseCount. T
he Oven Door open/close count must record in OvenDoorOpenCloseCount."
print(word_tokenize(text))

Output: [“After” ,“Oven” ,“Power on”, “If” ,the “OvenDoor” ,“open”, “Oven L
ight “, must “turn on” and must “record”, “oven light” “on/off count”, in “
OvenLightOpenCloseCount” and “Oven Door open/close “count” must record in “
OvenDoorOpenCloseCount”
]
```

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

```
from tensorflow.keras.preprocessing.text
import Tokenizer

sentences = [
    'After The Oven Power on',
    'If the Oven door open',
    'Oven Light must turn on','must record oven light on/off coun
t in OvenLightOpenCloseCount',
    'The Oven Door open/close count must record in OvenDoorOpenCl
oseCount'
]

tokenizer = Tokenizer(num_words)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

```
Output: {'oven': 1, 'the': 2, 'on': 3, 'must': 4, 'door': 5, 'open':
6, 'light': 7, 'record': 8, 'count': 9, 'in': 10, 'after': 11,
'power': 12, 'if': 13, 'turn': 14, 'off': 15,
'ovenlightopenclosecount': 16, 'close': 17, 'ovendooropenclosecount':
18}
```

Here are the following conditions scripted to write a test case using the split tokens.

- The "Power", On" words are considered preconditioning steps and they will link to the test unit. Also, it will add the Power Off step at postcondition.
- The parameters are identified and look into the database to find relative functions and values. "OvenDoorOpenCloseCount" and "OvenLightOnOffCount" are ECU parameters.
- The "Record" word identifies as some parameter count should store in some variable. After the specific action, it will verify the count. Also to compare with previous values, it will read this before the action and store it in a new local variable.
  i.e, "Read ECU variable OvenDoorOpenCloseCount X_Count", Then after the oven door open and close it will write the verify step as below.
  "Check ECU variable OvenDoorOpenCloseCount X_Count+1"

The above requirement is processed by the NLP tokenization algorithm and will give the following output which is shown in Table 11. Table 11 is the standard test case which is the base input of this research object 1. In this, setup steps are the precondition of the test case, main steps are the actual test steps and cleanup steps are posted condition steps. Action keywords are used to find the right library based on the function keyword. Parameter and values are inputs to the selected library function.

| **Action** | **Function/Service Type** | **Parameter** | **Value** |
|---|---|---|---|
| **Setup** | | | |
| Set | Function | Test Unit | Power ON |
| **Main Steps** | | | |
| Read | ECU Variable | OvenDoorOpenCloseCount | X_Count |
| Read | ECU Variable | OvenLightOnOffCount | Y_Count |
| Set | ECU Variable | Oven Door | Open |
| Check | ECU Variable | Oven Light | ON |
| Set | ECU Variable | Oven Door | Close |
| Check | ECU Variable | Oven Light | OFF |
| Check | ECU Variable | OvenDoorOpenCloseCount | X_Count + 1 |
| Check | ECU Variable | OvenLightOnOffCount | Y_Count + 1 |
| **Clean-up** | | | |
| Set | Function | Test Unit | Power OFF |

Table 11. System generated test case

## 6.3 RESULTS AND DISCUSSION

The proposed method is very useful in the embedded software testing process. Also, developers and test engineers have very clear tracking of software requirements. Of course, there is some limitation in the automation. The maximum complexity goes to 10 to 20 % project, and this can be handled manually. The complexity of manual work can automate with help of the intelligent logic and learning input data. The chunking is more helpful in terms of general English software requirements. Chunking is a process of extracting phrases from unstructured text. Instead of just simple tokens which may not represent the actual meaning of the text. The current work is a proof of concept for converting requirements to test cases. The full-fledged implementation required more algorithms and conditions to understand the sentences.

## 6.4 SUMMARY

The chapter gives the proof of concept that how the software requirements can convert to test cases as per designed standards. The complete solution required to add more NLP algorithms includes POS tagging and chunking.

# Chapter 7 CONCLUSION AND FUTURE SCOPE OF WORK

## 7.1 INTRODUCTION

Literature survey manifests that focus on the issue related to software testing has given not much focus by academicians and researchers. The biggest challenge in front of a software tester in software testing is the generation of valid and effective test cases. This research work of testing automation reduces the significant time taken by verification and validation. It is not possible to generate 100% test scripts from test cases. Some complex test cases have to depend on manual script development. As per this work analysis, 70 to 80% of test cases are simple and easy for auto script generation. The remaining 20 to 30 % required manual efforts. However, overall, it reduces the time for verification and validation. This tool has the scope of commercial revenue generation potential so that it has to meet automotive and other standards.

## 7.2 CONCLUSIONS

Some of the key summary points based on the research work have been stated as:

1. Objective 1 is a novel method to generate test scripts using key label data and test configuration data.
2. Objective 2 is part of script generation and a powerful library to compare test images and reference images.
3. Objective 3 is a novel method for converting software requirements to standard test cases.
4. The combination of all these objectives makes a powerful tool for embedded software verification and validation.

## 7.3 LIMITATIONS AND FURTHER SCOPE OF THE RESEARCH WORK

The automotive industry uses to spend a minimum of 48 months to release any new vehicle. Now this development time is reduced to 18 months. There is a possibility to reduce more time with the help of automation. The test automation will reduce a large amount of time.

The limitation of this research work follows.

1. The objective 1 script generation depends on the test bench configuration and pre-developed libraries. This libraries development is part of both the existing test automation method and implemented new test automation method. But if the test configuration changes, the entire implementation has to modify as per the new configuration.
2. The images comparison algorithm accuracy varies depending on the different scenarios. So the similarity tuning is required to achieve better results. The combination of OCR and image similarity algorithms gives more reliability.
3. NLP based software requirements to test cases is bug challenging because the software requirement language may not understand by this research work implemented algorithms. In this case, more analysis and development is required.

## 7.3 PUBLICATIONS

**Published:**

[1] Perala, S., Roy, A. "**A novel framework design for test case authoring and auto test scripts generation**", Turkish Journal of Computer and Mathematics Education, 2021, 12(6), pp. 1479,1487, ISSN 1309-4653

[2] Perala, S., Roy, A. "**A review on test automation for test cases generation using NLP techniques",** Turkish Journal of Computer and Mathematics Education, 2021, 12(6), pp. 1488.1491, ISSN 1309-4653.

[3] Srinivas Perala, Dr Ajay Roy and Koushik "**A Novel Method of Test Automation for Testing Embedded Software**" Think India Journal, Vol-22-Issue-37-December - 2019, ISSN 0971-1260.

[4] Srinivas Perala, Dr Ajay Roy and Dr Sandeep Ranjan "**Image Processing Algorithm to Compare Test image with Reference Image to Validate Embedded Software of Display Application**" Conference world, Jan-2021, ISBN 97881-948668-6-2.

[5] Perala, S., Roy, A., Ranjan, S. (2022). **Optical Character Recognition for Test Automation Using LabVIEW.** In: Karrupusamy, P., Balas, V.E., Shi, Y. (eds) Sustainable Communication Networks and Application. Lecture Notes on Data Engineering and Communications Technologies, vol 93. Springer, Singapore. https://doi.org/10.1007/978-981-16-6605-6_36

**Communicated:**

[1] Perala, S., Roy, A. "**Implementing Hardware-in-the-Loop Test automation on AWS**", ACM Journal on Emerging Technologies in Computing Systems,SCI Journal, ISSN:1550-4832.

# Chapter 8 BIBLIOGRAPHY

[1] M. Portolan, "Automated Testing Flow: The Present and the Future," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 2952-2963, Oct. 2020, doi: 10.1109/TCAD.2019.2961328.

[2] IEEE Std 1149.1-2001, "IEEE Standard Test Access Port and BoundaryScan Architecture", IEEE, USA, 2001 [2] A.C. Evans, "The New ATE: Protocol Aware", 2007 IEEE International Test Conference (ITC 07), Year: 2007

[3] H. Moon, G. Kim, Y. Kim, S. Shin, K. Kim and S. Im, "Automation Test Method for Automotive Embedded Software Based on AUTOSAR," 2009 Fourth International Conference on Software Engineering Advances, 2009, pp. 158-162, doi: 10.1109/ICSEA.2009.32.

[4] http://kidbs.itfind.or.kr/WZIN/jugidong/1278/1278 01.htm [June 16, 2009]

[5] http://www.autosar.org

[6] Seongsoo Hong, Technology Trends in Automotive RTOS and Component Middleware. A tutorial review, The Korean Society of Automotive Engineers (KSAE) Symposium, Seoul, 2006, 35- 59.

[7] Wooseok Yoo, AUTOSAR-based Software Architecture for Automotive Software, A tutorial review, The Korean Society of Automotive Engineers (KSAE) Symposium, Seoul, 2006, 60- 65.

[8]D.Kaleita and N.Hartmann, "Test Development Challenges for Evolving Automotive Electronic Technologies "SAE 2004, 2004-21-0015, Oct. 2004

[9] Electronic Architecture and Software Tools – Architecture Description Language, Embedded Electronic Architecture Std. 1.02, June 2004

[10] A. Sung, S. Kim, Y. Kim, Y. Jang and J. Kim, "Test Automation and Its Limitations: A Case Study," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1208-1209, doi: 10.1109/ASE.2019.00139.

[11] V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution," in *IEEE Software*, vol. 34, no. 2, pp. 90-96, Mar.-Apr. 2017. doi: 10.1109/MS.2017.34

[12] Amelio, A. (2019). A new axiomatic methodology for image similarity. Applied Soft Computing, 81, 105474. doi:10.1016/j.asoc.2019.04.043

[13] Elgharbawy, M., Scherhaufer, I., Oberhollenzer, K., Frey, M., & Gauterin, F. (2018). Adaptive functional testing for autonomous trucks. International Journal of Transportation Science and Technology. doi:10.1016/j.ijtst.2018.11.003

[14] Rauf, E. M. A., & Reddy, E. M. (2015). Software Test Automation: An Algorithm for Solving System Management Automation Problems. Procedia Computer Science, 46, 949–956. doi:10.1016/j.procs.2015.01.004

[15] Raikwar, S., Jijyabhau, W. L., Arun Kumar, S., & Sreenivasulu Rao, M. (2018). Hardware-in-the-Loop Test Automation of Embedded Systems for Farm Tractors. Measurement. doi:10.1016/j.measurement.2018.10.014

[16] Ma, C., & Provost, J. (2019). Introducing plant features to model-based testing of programmable controllers in automation systems. Control Engineering Practice, 90, 301–310. doi:10.1016/j.conengprac.2019.07.006

[17] Tumasov, A. V., Vashurin, A. S., Trusov, Y. P., Toropov, E. I., Moshkov, P. S., Kryaskov, V. S., & Vasilyev, A. S. (2019). The Application of Hardware-in-the-Loop (HIL) Simulation for Evaluation of Active Safety of Vehicles Equipped with Electronic Stability Control (ESC) Systems. Procedia Computer Science, 150, 309–315. doi:10.1016/j.procs.2019.02.057

[18] Kübler, K., Schwarz, E., & Verl, A. (2019). Test case generation for production systems with model-implemented fault injection consideration. Procedia CIRP, 79, 268–273. doi:10.1016/j.procir.2019.02.065

[19] Di Mare, G., Vico, F., Crisci, F., Montieri, A., Amoroso, D., Marino, B., … D’Avino, C. (2018). An innovative real-time test setup for ADAS's based on vehicle cameras. Transportation Research Part F: Traffic Psychology and Behaviour. doi:10.1016/j.trf.2018.05.018

[20] Biassoni, F., Ruscio, D., & Ciceri, R. (2016). Limitations and automation. The role of information about device-specific features in ADAS acceptability. Safety Science, 85, 179–186. doi:10.1016/j.ssci.2016.01.017

[21] Zhou, J., & Re, L. del. (2017). Reduced Complexity Safety Testing for ADAS & ADF. IFAC-PapersOnLine, 50(1), 5985–5990. doi:10.1016/j.ifacol.2017.08.1261

[22] Chan, M. S. C., del Rio-Chanona, E. A., Fiorelli, F., Arellano-Garcia, H., & Vassiliadis, V. S. (2016). Construction of global optimisation constrained NLP test cases from unconstrained problems. Chemical Engineering Research and Design, 109, 753–769. doi:10.1016/j.cherd.2016.03.015

[23] Elallaoui, M., Nafil, K., & Touahni, R. (2018). Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques. Procedia Computer Science, 130, 42–49. doi:10.1016/j.procs.2018.04.010

[24] D. Xu, W. Xu, M. Kent, L. Thomas, and L. Wang, "An Automated Test Generation Technique for Software Quality Assurance," in *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 247-268, March 2015, doi: 10.1109/TR.2014.2354172.

[25] Liao, L., Zhao, Y., Wei, S., & Zhao, Y. (2018). Improving Image Similarity Estimation via Global Distance Distribution Information Neurocomputing. doi:10.1016/j.neucom.2018.12.059

[26] M. Wahler, E. Ferranti, R. Steiger, R. Jain, and K. Nagy, "CAST: Automating Software Tests for Embedded Systems," *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, Montreal, QC, 2012, pp. 457-466.

[27] B. Broekman and E. Notenboom, *Testing Embedded Software*. Pearson Education, 2003.

[28] E. P. Enoiu, A. Cauevic, D. Sundmark, and P. Pettersson, "A Controlled Experiment in Testing of Safety-Critical Embedded Software," *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Chicago, IL, 2016, pp. 1-11.

[29] Byeongdo Kang, Young-Jik Kwon, and R. Y. Lee, "A design and test technique for embedded software," *Third ACIS Int'l Conference on Software Engineering Research, Management and Applications (SERA'05)*, Mount Pleasant, MI, USA, 2005, pp. 160-165

[30] S. Thummalapenta, S. Sinha, N. Singhania and S. Chandra, "Automating test automation," *2012 34th International Conference on Software Engineering (ICSE)*, Zurich, 2012, pp. 881-891. doi: 10.1109/ICSE.2012.6227131.

[31] P. Kadekar and A. Wakankar, "Automated Test Environment for On-Board Diagnostics Counters for an Automotive On-Road Application," *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, 2018, pp. 409-413

[32] R. M. Bhide and V. S. Kulkarni, "Automated testing tool for engine software testing," *2016 International Conference on Automatic Control and Dynamic Optimisation Techniques (ICACDOT)*, Pune, 2016, pp. 940-942.

[33] P. Shende, D. Scarfe, W. Meek, and S. Krishna, "Automating HIL test setup and execution," *2008 IEEE AUTOTESTCON*, Salt Lake City, UT, 2008, pp. 118-121.

[34] G. Amato and P. Savino, "Searching by Similarity and Classifying Images on a Very Large Scale," *2009 Second International Workshop on Similarity Search and Applications*, Prague, 2009, pp. 149-150.

[35] Z. Zeng, H. Li, W. Liang, and S. Zhang, "Similarity-based image classification via kernelised sparse representation," *2010 IEEE International Conference on Image Processing*, Hong Kong, 2010, pp. 277-280

[36] Y. Choi, K. Kim, Y. Nam and W. Cho, "Retrieval of Identical Clothing Images Based on Local Color Histograms," *2008 Third International Conference on Convergence and Hybrid Information Technology*, Busan, 2008, pp. 818-823.

**[**37**]** C. Lin, Y. Ching and Y. Yang, "Automatic Method to Compare the Lanes in Gel Electrophoresis Images," in *IEEE Transactions on Information Technology in Biomedicine*, vol. 11, no. 2, pp. 179-189, March 2007.

[38] A. Ansari, M. B. Shagufta, A. Sadaf Fatima and S. Tehreem, "Constructing Test cases using Natural Language Processing," *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, Chennai, 2017, pp. 95-99

[39] S. Vemuri, S. Chala and M. Fathi, "Automated use case diagram generation from textual user requirement documents," *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, Windsor, ON, 2017, pp. 1-4

[40] D. Winkler, R. Hametner, T. Östreicher, and S. Biffl, "A framework for automated testing of automation systems," *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, Bilbao, 2010, pp. 1-4.

[41] A. Bansal, M. Muli and K. Patil, "Taming complexity while gaining efficiency: Requirements for the next generation of test automation tools," *2013 IEEE AUTOTESTCON*, Schaumburg, IL, 2013, pp. 1-6

[42] *Abhishek Shukla, Lina Nath*, *"*Automation Desk: A Universal Tool for Creating & Managing Automation Tasks**",** *Adv. Res. Power Electro. Power Sys***.** ADR Journals, 2014

[43] Sun Rui and Zhong Deming, "Translating software requirement from natural language to automaton," *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, Shengyang, 2013, pp. 2456-2459.

[44] G. Grano, T. V. Titov, S. Panichella and H. C. Gall, "How high will it be? Using machine learning models to predict branch coverage in automated testing," *2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, Campobasso, 2018, pp. 19-24.

[45] M. Helali Moghadam, M. Saadatmand, M. Borg, M. Bohlin and B. Lisper, "Machine Learning to Guide Performance Testing: An Autonomous Test Framework," *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Xi'an, China, 2019, pp. 164-167

[46] C. E. Tuncali, G. Fainekos, H. Ito and J. Kapinski, "Simulation-based Adversarial Test Generation for Autonomous Vehicles with Machine Learning Components," *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, 2018, pp. 1555-1562.

[47] dSPACE Product Information, http://www.dSPACE.da (date:01-05-2019)

[48] Li, M., Bai, H., & Krishnamurthi, N. (2019). A Markov Decision Process for the Interaction between Autonomous Collision Avoidance and Delayed Pilot Commands. IFAC-Papers OnLine, 51(34), 378–383. doi:10.1016/j.ifacol.2019.01.012

[49] Sarkar, S., Raj, R., Vinay, S., Maiti, J., & Pratihar, D. K. (2019). An optimisation-based decision tree approach for predicting slip-trip-fall accidents at work. Safety Science, 118, 57–69. doi:10.1016/j.ssci.2019.05.009

[40] Dauda, K. A., Pradhan, B., Uma Shankar, B., & Mitra, S. (2019). Decision tree for modeling survival data with competing risks. Biocybernetics and Biomedical Engineering. doi:10.1016/j.bbe.2019.05.001

[51] C. J. Budnik, W. K. Chan and G. M. Kapfhammer, "Bridging the Gap Between the Theory and Practice of Software Test Automation," 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, pp. 445-446, doi: 10.1145/1810295.1810421.

[52] Hou, D., Wang, S., & Xing, H.: Query-Adaptive Remote Sensing Image Retrieval Based on Image Rank Similarity and Image-to-Query Class Similarity. IEEE Access, 8, 116824-116839 (2020).

[53] Penney, G. P., Weese, J., Little, J. A., Desmedt, P., & Hill, D. L.: A comparison of similarity measures for use in 2-D-3-D medical image registration. IEEE Transactions on Medical Imaging. 17(4), 586-595 (1998).

[54] Rahman, M. M., Antani, S. K., & Thoma, G. R.: A Learning-Based Similarity Fusion and Filtering Approach for Biomedical Image Retrieval Using SVM Classification and Relevance Feedback. IEEE Transactions on Information Technology in Biomedicine. 15(4), 640-646 (2011).

[55] Sampat, M. P., Wang, Z., Gupta, S., Bovik, A. C., & Markey, M. K.: Complex Wavelet Structural Similarity: A New Image Similarity Index. IEEE Transactions on Image Processing. 18(11), 2385-2401 (2009).

[56] Bai, X., Zhang, Y., Liu, H., & Chen, Z.: Similarity Measure-Based Possibilistic FCM With Label Information for Brain MRI Segmentation. IEEE Transactions on Cybernetics. 49(7), 2618-2630 (2019).

[57] Bustin, A., Voilliot, D., Menini, A., Felblinger, J., de Chillou, C., Burschka, D., ... & Odille, F..: Isotropic Reconstruction of MR Images Using 3D Patch-Based Self-Similarity Learning. IEEE Transactions on Medical Imaging. 37(8), 1932-1942 (2018)

[58] Kim, D., Park, J., Jung, J., Kim, T., & Paik, J.: Lens distortion correction and enhancement based on local self-similarity for high-quality consumer imaging systems. IEEE Transactions on Consumer Electronics. 60(1), 18-22 2014.

[59] Schnabel, J. A., Tanner, C., Castellano-Smith, A. D., Degenhard, A., Leach, M. O., Hose, D. R., ... & Hawkes, D. J.: Validation of nonrigid image registration using finite-element methods: application to breast MR images. IEEE Transactions on Medical Imaging. 22(2), 238-247 2003.

[60] Kumar, A., Verma, P. K., Perala, S., & Chadha, P. R.: Automatic Attendance Visual by Programming Language Lab VIEW. IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems. 1-5 (2016).

[61] J. Hirschberg, B. W. Ballard and D. Hindle, "Natural language processing," in AT&T Technical Journal, vol. 67, no. 1, pp. 41-57, January-February 1988, doi: 10.1002/j.1538-7305.1988.tb00232.x.

[62] A. Celikyilmaz, R. Sarikaya, M. Jeong and A. Deoras, "An Empirical Investigation of Word Class-Based Features for Natural Language Understanding," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 24, no. 6, pp. 994-1005, June 2016, doi: 10.1109/TASLP.2015.2511925.

[63] Agrawal, H. (1999), "Efficient Coverage Testing Using Global Dominator Graphs", Proceedings of ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, held at Toulouse, France, Vol. 24, No.5, pp. 11-20.

[64] Briand, L., Labiche, Y., Buist, K. and Soccar, G. (2002), "Automating impact analysis and regression test selection based on UML designs", Proceedings of International Conference on Software Maintenance, pp. 252-261.

[65] Castro, L. N. and Timmis, J. I. (2002), "Artificial immune systems: a new computational intelligence approach", Springer-Verlag, London, UK.

[66] Causevic, A., Sundmark, D. and Punnekkat, S. (2012), "Test case quality in test-driven development: a study design and a pilot experiment", Proceedings of 16th International Conference on Evaluation and Assessment in Software Engineering (EASE 2012), pp. 223 – 227.

[67] Chen, Y. F., Rosenblum, D. S. and Vo, K. P. (1994), “Test tube: A system for selective regression testing”, Proceedings of the 16th International Conference on Software Engineering (ICSE 1994), Los Alamitos, CA, USA, pp. 211-220.

[68] Elbaum, S., Malishevsky, A. G. and Rothermel, G. (2002), “Test case prioritization: a family of empirical studies”, IEEE Transactions on Software Engineering, Vol. 28, No. 2, pp. 159-182

[69] D. Xu, W. Xu, M. Kent, L. Thomas and L. Wang, "An Automated Test Generation Technique for Software Quality Assurance," in IEEE Transactions on Reliability, vol. 64, no. 1, pp. 247-268, March 2015, doi:10.1109/TR.2014.2354172.

[70] Ray Lattarulo, Joshué Pérez, Martin Dendaluce, A complete framework for developing and testing automated driving controllers, IFAC-PapersOnLine, Volume 50, Issue 1,2017,Pages 258-263, ISSN 2405-8963.

[71] M. H. Moghadam, M. Borg and S. J. Mousavirad, "Deeper at the SBST 2021 Tool Competition: ADAS Testing Using Multi-Objective Search," 2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST), 2021, pp. 40-41, doi: 10.1109/SBST52555.2021.00018.