DEVELOP AND ENHANCE AN AUTOMATED FLOW (OPEN DEBUG (OD)) FOR THE PERIODIC SYSTEM MANAGEMENT INTERRUPT (PSMI) WHEN A FAILURE IS CAPTURED BY OPEN DEBUG (OD) USING PYTHON

NG SHIN HUAN

MASTER OF ENGINEERING (ELECTRONIC SYSTEMS)

FACULTY OF ENGINEERING AND GREEN TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

August 2023

DEVELOP AND ENHANCE AN AUTOMATED FLOW (OPEN DEBUG (OD)) FOR THE PERIODIC SYSTEM MANAGEMENT INTERRUPT (PSMI) WHEN A FAILURE IS CAPTURED BY OPEN DEBUG (OD) USING PYTHON

By

NG SHIN HUAN

A thesis submitted to

Faculty of Engineering and Green Technology,

Universiti Tunku Abdul Rahman,

in partial fulfilment of the requirements for the degree of

Master of Engineering (Electronic Systems)

August 2023

ABSTRACT

DEVELOP AND ENHANCE AN AUTOMATED FLOW (OPEN DEBUG (OD)) FOR THE PERIODIC SYSTEM MANAGEMENT INTERRUPT (PSMI) WHEN A FAILURE IS CAPTURED BY OPEN DEBUG (OD) USING PYTHON

NG SHIN HUAN

Consistency, reliability and accuracy of any implemented machine products are highly demanded by everyone in this world no matter which level of the workplace the user is. In order to achieve this goal, various advanced technologies are introduced with automated technique which is aimed to replace the traditional manual method – manpower. This applies the same in the post-Si environment. The debugger needs to understand the behaviour of that particular failure, hence PSMI tool is needed to capture it. However, this tool is needed to be used manually by every debugger. In order to automate it, this project will summarise the flow and that automated system will be inserted into a flow named Open Debug (OD). This has aimed to reduce the time taken and effort needed to capture the PSMI trace. In Literature Review, various fields are introduced with automated methods in order to achieve time and effort saving goal and at the same time increasing the efficiency and effectiveness of the results produced as compared to manual method. To design the automated way for PSMI capture, Python script is utilised. In Methodology, the design of the automated flow of the PSMI capture in the OD flow is explained in detailed as well as in the flow chart manner. For this developed Python script to be triggered, a failure has to be detected at the first stage. The trigger point used in the PSMI capture is mainly the memory trigger. This method is basically to detect the offset 0x28 of the error block address associated with the specific data depending on the error code. 4 projects are selected to test the developed script by comparing the manual method (the common way that all debuggers will use) and the automated way (using the implemented Python script that is inserted in the OD flow). The manual method time estimation for PSMI capture is computed based on survey from debuggers while the automated method time computation for PSMI capture is the average analysis from each of the project using the developed Python script. Based on the results obtained, using automated technique can greatly save 55% to 84% of the time used over the manual technique in capturing PSMI. In conclusion, by using automated approach for PSMI capture in the OD flow, it is proven that the time and effort can be effectively saved as compared to the manual approach.

ACKNOWLEDGEMENT

The accomplishment of this thesis involves many important people because they provide me a lot of guidance, supports and assistances throughout the thesis. They have played crucial roles in leading me the direction to succeed in complete this thesis within a fixed duration of time.

First and foremost, I would like to express my grateful thankfulness to all my family members. This is because they have supported me all the way in every moment of my life even though they may not be having any knowledge on this project. Special thanks to them of the encouragements and financial supports throughout the whole life in study in Universiti Tunku Abdul Rahman (UTAR).

Other than that, I also would like to thank to my supervisor, Ts Dr Lee Han Kee, from the Faculty of Engineering and Green Technology. Dr. Lee is very patient in answering all my questions and helping me to curb all the doubts I face without rushing. His non-stop supervisions, advices and assistances has directed me to achieve success in this project. Undeniably, with his fully supports, my thesis is able to complete on time.

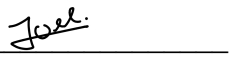
Moreover, I would like to take this opportunity as well to thank to Dr Joshua in UTAR. Dr Joshua has led me the way to overcome the obstacles faced in thesis report writing previously during the process of doing this project. His detailed explanations are easily to understand, allowing me to complete the writing process smoothly.

Lastly, I would like to thanks my classmates, friends and colleagues who have helped me throughout the entire project. They assist me in providing me their ideas and viewpoints, leading me to have better point of view in this project.

APPROVAL SHEET

This thesis entitled "**DEVELOP AND ENHANCE AN AUTOMATED FLOW (OPEN DEBUG (OD)) FOR THE PERIODIC SYSTEM MANAGEMENT INTERRUPT (PSMI) WHEN A FAILURE IS CAPTURED BY OPEN DEBUG (OD) USING PYTHON**" was prepared by NG SHIN HUAN and submitted as partial fulfilment of the requirements for the degree of Master of Engineering (Electronic Systems) at Universiti Tunku Abdul Rahman.

Approved by:



(Ts Dr Lee Han Kee) Date: …...18/8/2023……

Supervisor

Department of Electronic Engineering

Faculty of Engineering and Green Technology

Universiti Tunku Abdul Rahman

Faculty of Engineering and Green Technology

Universiti Tunku Abdul Rahman

Date: 18-8-2023

SUBMISSION OF THESIS

It is hereby verified that NG SHIN HUAN (ID No: 21AGM06717) has completed this thesis entitled "**DEVELOP AND ENHANCE AN AUTOMATED FLOW (OPEN DEBUG (OD)) FOR THE PERIODIC SYSTEM MANAGEMENT INTERRUPT (PSMI) WHEN A FAILURE IS CAPTURED BY OPEN DEBUG (OD) USING PYTHON**" under the supervision of Ts Dr Lee Han Kee from the Department of Electronic Engineering, Faculty of Engineering and Green Technology.

I understand that the University will upload softcopy of my thesis in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

Huan

_____

(NG SHIN HUAN)

DECLARATION

I, NG SHIN HUAN, hereby declare that the thesis is based on my original work except for quotations and citations which has been duly acknowledged. I also declared that it has not been previously or currently submitted for any other degree at UTAR or other institutions.

Huan

_____

(NG SHIN HUAN)

Date: 18-8-2023

TABLE OF CONTENTS

LIST OF TABLES

LIFT OF FIGURES

LIST OF ABBREVIATIONS

ATE	Automated test equipment
BFS	Breadth first search
DFS	Depth first search
DUT	Device under test
ESSTE	Embedded software simulation testing environment
ETL	Extraction, transformation, loading
HFSS	High frequency structure simulator
IoT	Internet of Things
MATF	Multi-platform automatic testing framework
MISTA	Model-based integration and system test automation
NLP	Natural language processing
OCR	Optical character recognition
OD	Open Debug
OTL	Online teaching and learning
PSMI	Periodic system management interrupt
RHG	Reference help guide
RTL	Register transfer level
RTOS	Real time operating system
SUT	System under test
UUS	Unit under test
VB	Visual basic
VM	Virtual machine

CHAPTER 1 INTRODUCTION

1.1 Background

Consistency, reliability and accuracy of any implemented machine products are highly demanded by everyone in this world no matter which level of the workplace the user is. This phenomenon undeniably introduces countless challenges to all machine tools' developers and engineers in order to implement these cost-effective tools so that they are able to operate under extreme conditions effectively and efficiently. Therefore, various advanced technologies are introduced especially with most of the machines controlled and operated by manpower being gradually replaced by automated machines.

Open Debug (OD) is an automated process, which also consists of many sub-processes and is used to generate important folders and files for future debug usage when a failure is captured based on scripts written in Python. With OD, it does help debuggers to streamline and bucketize the failures captured known as screening process (Intel Wiki - OD).

PSMI is abbreviated from Periodic System Management Interrupt. The intended audiences who are always using are Post-Silicon (Post-Si) Debug teams. For validation team, the main purpose is to root-cause the bugs (reasons of the failures) before the product is released to the customers. The failures are usually found during testing flow. PSMI is one of the tools for debuggers to root-cause the failures at the RTL (register transfer level). PSMI is defined as a debug tool to reproduce the processor behaviour in a simulator model. It helps to provide a full view of the behaviour of the processor named as a PSMI trace such as how an instruction is executed in a microarchitecture behaviour or in the form of waveforms. The main advantage of using this debug tool is to eliminate the confusions as the reasons of the failures vary and understand in depth the microarchitecture behaviour (Intel Wiki - PSMI).

Based on the research done by Pravisha, Rupesh and Sonia (2019), they applied modified python scripts into the Regression Testing in the format of Python to automate this process. Indirectly, the usage of manpower was eliminated, thus significantly boosting the testing process. Another research team, Chongwu, Bin, Yongfeng and Chang (2009), conducted an experiment in real-time system known as Embedded software simulation testing environment (ESSTE) using test script. This has proven that the designed test script in Python has benefits over manual method because of its simplicity, controllability, portability, flexibility, reusability and expandability. With this, the time taken for the developing process was greatly minimised

especially in real-time operating system. Radhakrishna, SravanKiran and Ravikiran (2012) boosted the ETL (Extraction, Transformation, Loading) process system by inserting a command-based script to this system to automate the processes. These automations can greatly improve the overall performances such as having faster data processing in data and quality and reliability as compared to manual processes.

The automated penetration testing was carried out by Yaroslav, Andraian and Roman (2016) by utilising a developed script applied in UNIX environment. By this way, a safer and simpler method was created to carry out all the tasks related to the penetration testing. Unquestionably, using automated way in this system can improve the performances because this technique can eliminate most of the problems caused by the human factors. A solo researcher, named Roy Walker (2019), used a scripting method on the automated test equipment (ATE). The main rationales of designing this method were to automate the process, ease the building process and lastly speed up the validation process. Other than this, scripting way was cost-effective and the risk-low system design. The flexibility and the commonality of the programs would be minimised for different test assets. More importantly, 3Rs (repurpose, reduce and reuse) could be taken in order to apply for other systems. According to the research by B. Kovacevic, M. Kovacevic, D. Stefanovic and M. Loncarevic (2015), they conducted an automated verification test of set-top box media player functionality – performances, stress and functional tests. This test was aimed to reduce the time-consuming manual verification method. This proposed automated way testing indirectly boosted the testing accuracy, reduced the time taken needed and reduced manpower needed as compared to the manual way testing. Moreover, test reports were also generated through the automation so that monitoring can be performed easily.

1.2 Problem Statements

The automated processes in Open Debug (OD) do ease the debuggers to gather the same issues found based on screening process. However, this screening process is only able to understand the first-level debug. This is a debug process to understand the failing signature and the possible failing point when a failure is gathered. To understand the failure in depth, PSMI is still needed (Intel Wiki - OD). Unquestionably, PSMI is the main debug tool used by the Post-Silicon (Post-Si) debuggers to debug a failure in RTL level. However, the main drawback of using this debug tool is the debuggers have to use a manual way in order to capture the PSMI trace. This means that the debuggers have to at least understand the basic steps such as understanding the behaviour of that particular failure and how PSMI is able to communicate to the failure before proceeding to capture the PSMI trace. This communication is known as trigger point. Moreover, the steps to capture the PSMI trace is time-consuming as the debuggers need to wait for the installation flows to complete. Trial-and-error method may be needed for special cases, leading to have an effort-consuming issue. Besides, there are some environment issues that may happen around the silicon system, resulting in taking a longer period of time to wait for the issues solved before proceeding to PSMI capture. This includes the silicon board gets into shutdown due to unexpected error such as power shutdown, wire loose in connections and temperature increase in the silicon board. (Intel Wiki - PSMI).

1.3 Objectives

i) To design an automated flow (Open Debug) for the PSMI when a failure is captured by Open Debug (OD) using Python.

ii) To analyse the performance in term of time of this PSMI OD flow when different failing signatures such as exception, memory mismatch, VM Exit Error and register mismatch are captured using the developed Python script.

iii) To compare the performance in term of time of this designed automated PSMI flow in the OD with the manual approach used by the debuggers.

1.4 Research Scopes

The scopes of the research-based thesis are listed in the following:

i) The programming language used to develop the script of the automated flow of the PSMI capture is Python.

ii) The written script of the automated flow of the PSMI capture will be inserted into the Open Debug (OD) flow.

iii) The failing signatures will be used for testing with the designed automated flow comprises exception, memory mismatch, VM Exit Errors and register mismatch.

iv) Same PSMI interval used throughout the research. Special case such as the failure that passes with PSMI installation will be excluded from the test.

v) Automated time period for PSMI captured starts when the OD starts the PSMI capture flow until the completion of the PSMI capture as stated in Chapter 3.4.

vi) Manual time period for PSMI captured starts when the debugger starts the initial process to understand the failure's behaviour until the completion of the PSMI capture.

vii) The heartbeat used or the minimum time required to run the test is 300,000ms or 5 minutes.

1.5 Layout of the thesis

Chapter 1 has discussed the introduction, problem statements, objectives and the research scopes (limitations). In Chapter 2, literature review is presented alongside with the prior works done using automated technique and the summary of the prior works. After that, Chapter 3 discusses the methodology which includes the overall project flow, the design of the automated flow of the PSMI capture in the OD flow, the design of the trigger point and the Python Scripting for the PSMI capture flow in the OD flow. Next. in chapter 4, 2 different techniques known as manual method (manpower) and automated method (developed in the OD using Python scripting) are compared with discussions. Lastly, Chapter 5 presents the conclusion of the thesis, limitations of the experiment and the recommended future works. References and Appendices are also included after that as well.

In the next chapter, the literature review as well as prior works and their summary will be presented.

CHAPTER 2 LITERATURE REVIEW

2.1 Background

This chapter is to discuss the general viewpoints and concepts of the automated and manual techniques used. Other than this, the previous works done by different researchers in different fields will be taken account into in the following section. After the discussion of previous works, a small summary will be made to generate some perspectives for this project. In the next section, the general ideas of the automated method versus the manual method will be conversed.

2.1.1 Automated Method Versus Manual Method

In this modern world, every developed software system is aimed to be fast, high quality with high efficiency and effectiveness. This creates many challenges to all developers. Hence, test stage is usually rejected because this stage takes a lot of time and cost, resulting in time and effort wasting. As a result, due to the lack of testing, the quality of the product will be reduced (Chaini, 2015). Other than the company, in other fields such as examination or assessments, manual technique that is marking the submitted manually will be a tedious job if the questions are subjective-based such as long essay questions without any unique answers or programming-based questions. Even there are similar answers provided, if the number of students is a lot, say 1000, the time taken and the effort spent by the teachers will be incredibly high. According to Jayawardena (2018), the accuracy of the marking standard will decrease due to the exhaustion by the teacher when he or she is marking the paper over a long period of time. For programming questions, as stated by Vimalaraj (2022), to provide a good evaluation process, if the examiner examines the codes by observing the inputs and the respective outputs, this is not ample since the overall coding could be out of scope. This will result in inconsistent checking, hence having a lower accuracy. On the other hand, if the codes are checked one line by one line, it is obviously spending a lot of time and effort. Furthermore, Sinha (2022) also mentions that the performance evaluation could be not precise if being evaluated manually because the examiner may be in a bad mood or there is a special relationship between the examiner and the examinee. By gathering all the above aspects, it is undeniably that introducing automated flow will bring many benefits in many fields in order to increase the efficiency (more work done in a given time) and effectiveness (save time). Based on recent trends and researches, the test stage is of utmost importance and it is almost impossible without the any tool support (Chaini, 2015 and Roja 2017). The next section will discuss the previous works done by different researchers.

2.2 Prior Works

Many researchers have carried out experiments with the automated script to test the performances of a particular system. The next section is to discuss the previous works that were done by different researchers or teams. The method used, applications applied, the strengths and the weaknesses of that proposed scripts on that application by the researcher will also be discussed.

2.2.1 Automated Regression Testing and Data Analytics using Python

An experiment was done by Pravisha, Rupesh and Sonia (2019). They carried out an automated process by modifying the current scripts so that the automated regression testing would be able to perform in a variety of test environments. For instance, humidity of the environment, various temperatures including extreme ones, different variations and more were tested. During testing phase, the behaviour of the tested device will be gathered and analysed by using the computer. Since this was an automated system flow, the usage of manpower was eliminated, thus greatly boosting the testing process. Moreover, the modified scripts would be able to be altered to follow the requirements needed for the tester. The block diagram for this tester was classified into 3 parts: hardware module, software module and support packages. These were connected to the device under test (DUT) and the master computer. Figure 2.1 below shows the results of the ideal and measured voltage values.

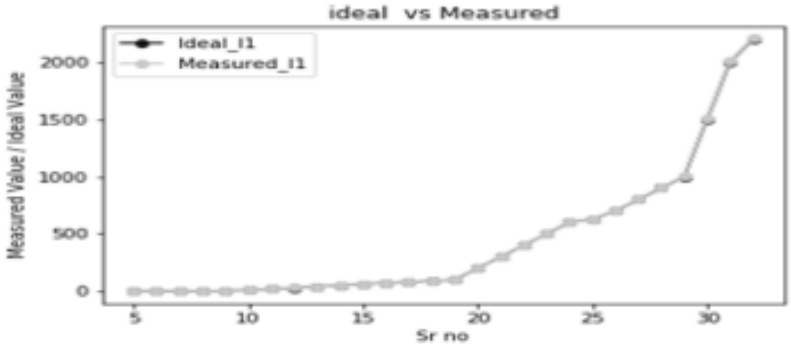


Figure 2.1 Ideal and measured voltage values (Pravisha, Rupesh and Sonia, 2019)

Based on Figure 2.1, it can be seen that the differences between the ideal and measured voltage values are almost zero or can be said the minimum difference. Undeniably, using automated script in performing this regression test has greatly improved the overall efficiency as the usage of manpower can be reduced to minimum.

2.2.2 Study on Real-Time Test Script in Automated Test Equipment

Chongwu, Bin, Yongfeng and Chang (2009) conducted research by performing some testing (Embedded software simulation testing environment, ESSTE) in the real-time embedded software system using test script. The objectives of using the designed test script in Python were because of its simplicity, controllability, portability, flexibility, reusability and expandability. With this, the time taken for the developing process was greatly reduced especially in real-time system. Figure 2.2 below shows the process block diagram of the developed automated system using the test script.

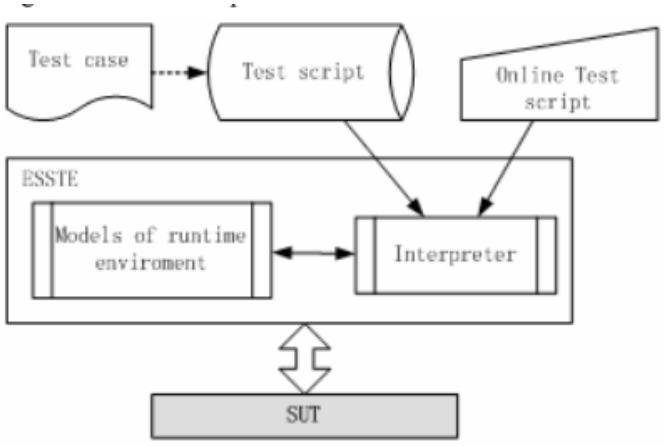


Figure 2.2 Process block diagram of the developed automated system using the test script (Chongwu, Bin, Yongfeng and Chang, 2009)

A test case was then injected into this designed script to check for the performances. Figure 2.3 shows the flow of the test cases used. Figure 2.4 shows the performances obtained based on test case in Figure 2.3.

1) Execute a test case named "takeoff_airline", to make UAV fly along the airline preinstalled;
2) Confirm the aircraft's status, if it's not flying airline, record this test case failed;
3) From the testing began between 120s to 720s, make communication failure(Set the fault flag of remote control to 1);
4) During communication failure, check the status of the aircraft per second, UAV should make a left circled flight(The roll angle should be held at -5 degree);
5) 10 minutes later from communication failure, confirm whether the plane returns to base (Flag "return_to_base_flag" should be set to 1, which can be watched from telemetry);
6) At 730s, recover the communication (Set the fault flag of remote control to 0);
7) Send a command of "hold height at 2000 meters" to FCMS through remote control;
8) At 800s, confirm if the height of UAV is held at 2000m (The error is less than 20m). If the height is not in the correct range, record this test case failed.

Figure 2.3 Test case flow use for the test script (Chongwu, Bin, Yongfeng and Chang, 2009)

Test item	execution time
100 sequential order statements (No conditional statement, no loop statement, no function call)	3.368 ms
1000 sequential order statements (No conditional statement, no loop statement, no function call)	19.841 ms
2000 loops (Each loop implement a simple statement)	2.342 ms
150 conditional statements	7.341 ms
100 function calls (The function is simple with one formal parameter and one return value)	2.637 ms

Figure 2.4 Performances collected based on different test items (Chongwu, Bin, Yongfeng and Chang, 2009)

Based on the figures, using test script in Python language is able to achieve the goals for real-time operating system (RTOS). Furthermore, it can be observed that the execution time for every test is just measured in millisecond which is very effective and efficient in RTOS. To wrap up, Python is a world-wide use language in test script and the open source is available anywhere due to its simplicity, controllability, portability, flexibility, reusability and expandability.

2.2.3 Automating ETL Process with Scripting Technology

Research was carried out by a team member including Radhakrishna, SravanKiran and Ravikiran (2012). They enhanced the ETL (Extraction, Transformation, Loading) process system by injecting a command-based script to this system to automate the processes. The purpose of proposing this automated flow into ETL system was to reduce the manual handling of those processes as well as develop a tool so that ETL process was able to support the script to automate the processes. Figure 2.5 shows the general idea of the command-based technique to automate the ETL process. Figure 2.6 shows the example of the test case being used for generating a script.

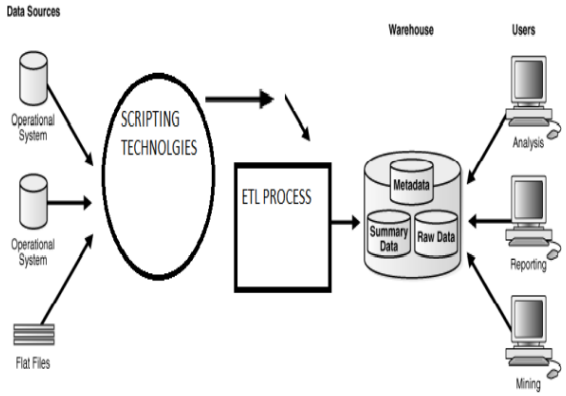


Figure 2.5 General idea of the script technique in ETL process (Radhakrishna, SravanKiran and Ravikiran, 2012)

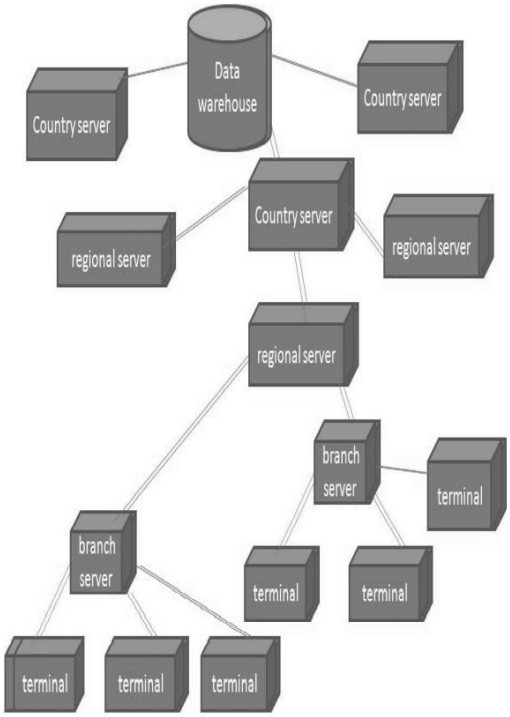


Figure 2.6 Test case being used to generate a script (Radhakrishna, SravanKiran and Ravikiran, 2012)

Based on Figure 2.6, as time went, the overall organization would grow bigger and bigger. A main problem here would be how to manage the business data as the data could be in different kind of formats and very difficult to be sent to all servers with only single platform. In order to accomplish this, this team developed an automated flow by processing numerous data from different areas and storing them in a warehouse which allowed the branch servers to get the data easily. As a conclusion, future tools such as automations can greatly improve the overall performances such as having faster data processing in data and quality and reliability as compared to manual processes.

2.2.4 Manual and Automated Penetration Testing. Benefits and Drawbacks. Modern Tendency

A research team from Ukraine, Yaroslav, Andraian and Roman (2016), carried out an experiment between a manual (customized approach) and an automated penetration testing. The automated penetration testing was using a developed script applied in UNIX environment. The main aim of doing this experiment was to create a safer and simpler method to carry out all the tasks related to the penetration testing. Figure 2.7 (left and right) shows the performances comparison between the traditional (manual) method and python script (automated) method.

	Manual	Automated
Testing Process	Manual, non-standard process; Labor and capital intensive; High cost of customization;	Fast, standard process; Easily repeatable tests;
Vulnerability/ Attack Database Manage-ment	Maintenance of database is manual; Need to rely on public database; Need re-write attack code for functioning across different platforms;	Attack database is maintained and updated; Attack codes are written for a variety of platforms;
Reporting	Requires collecting the data manually;	Reports are automated and customized;
Cleanup	The tester has to manually undo the changes to the system every time vulnerabilities found;	Automated testing products offer clean-up solutions;
Training	Testers need to learn non-standard ways of testing; Training can be customized and is time consuming.	Training for automated tools is easier than manual testing.

Figure 2.7 Summary of using manual and automated ways (Yaroslav, Andraian and Roman, 2016)

Based on Figure 2.7, it is very obvious that an automated technique being used in penetration testing does bring more benefits as compared to a traditional technique. Undeniably, using automated way in this system can improve the overall performances effectively and efficiently because this method can eliminate most of the problems caused by the human factors. However, the users have to at least understand the programming language as well as the system structure in depth before applying the automated way in this system.

2.2.5 Pattern System Design: An Approach to Automating the Design of Automated Test Equipment

A researcher named Roy Walker (2019) used an automated approach (scripting method) on the automated test equipment (ATE). The main rationales of designing this method were to automate the process, ease the building process and lastly speed up the validation process. To design this, the researcher used the original I/O patterns, routing modes and cabling schemas and then injected the system by using Unit Under Test (UUT) I/O equipment. One of the advantages here was the current testing method was good to be applied, this could be reconfigured for a brand-new test based on requirements of the new system. The example of the I/O system used by the researcher is shown in Figure 2.8.

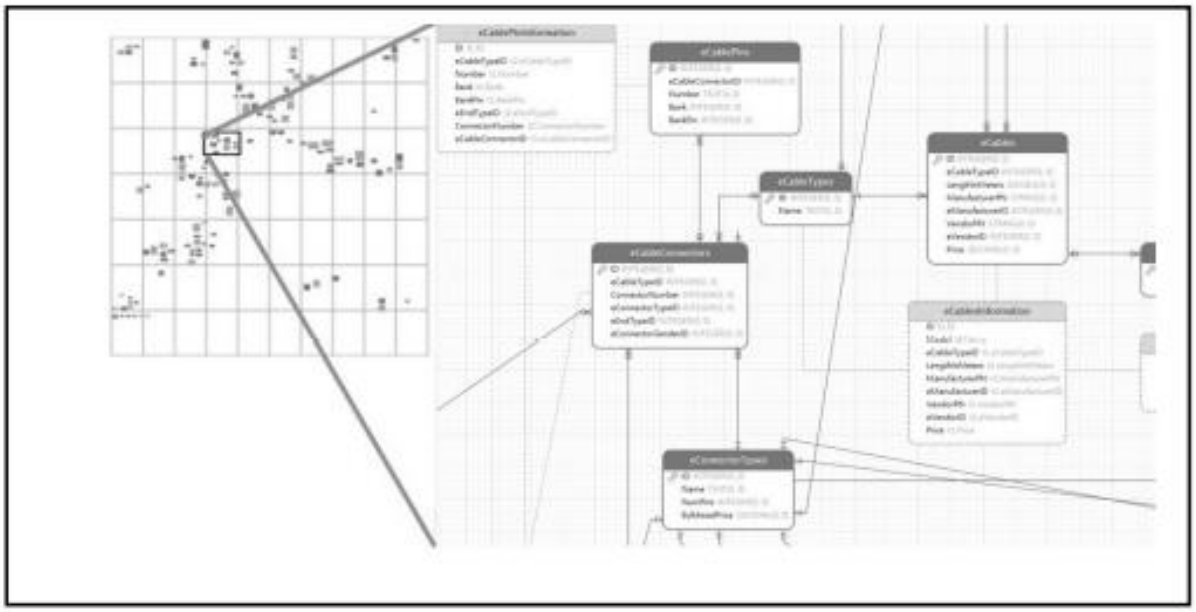


Figure 2.8 Example of I/O patterns in the pattern library (Roy Walker, 2019)

There were three benefits proven of applying the automated system into ATE as compared to using a traditional way method. Firstly, it was cost-effective and the risk-low system design. Secondly, the flexibility and the commonality of the programs would be minimised for different test assets. Lastly, 3Rs (repurpose, reduce and reuse) could be taken in order to apply for other systems.

2.2.6 Verification of Set-Top Box Media Player Functionality Using Automated Test System

A research team including B. Kovacevic, M. Kovacevic, D. Stefanovic and M. Loncarevic (2015) conducted an automated verification test using Python scripting of set-top box media player functionality – performances, stress and functional tests. This test was aimed to reduce the time-consuming manual verification method which lasted for at least 20 working days. They split this procedure into 3 parts which consisted of creating test cases, executing tests automatically as well as collecting and analysing results for future use. Figure 2.9 below shows the system overview. It was a simulation purpose when the user inputted the commands to the media player and the media player responded this with an output shown. Figure 2.10 showed the test results (based on bugs found) based on automatic and manual tests.

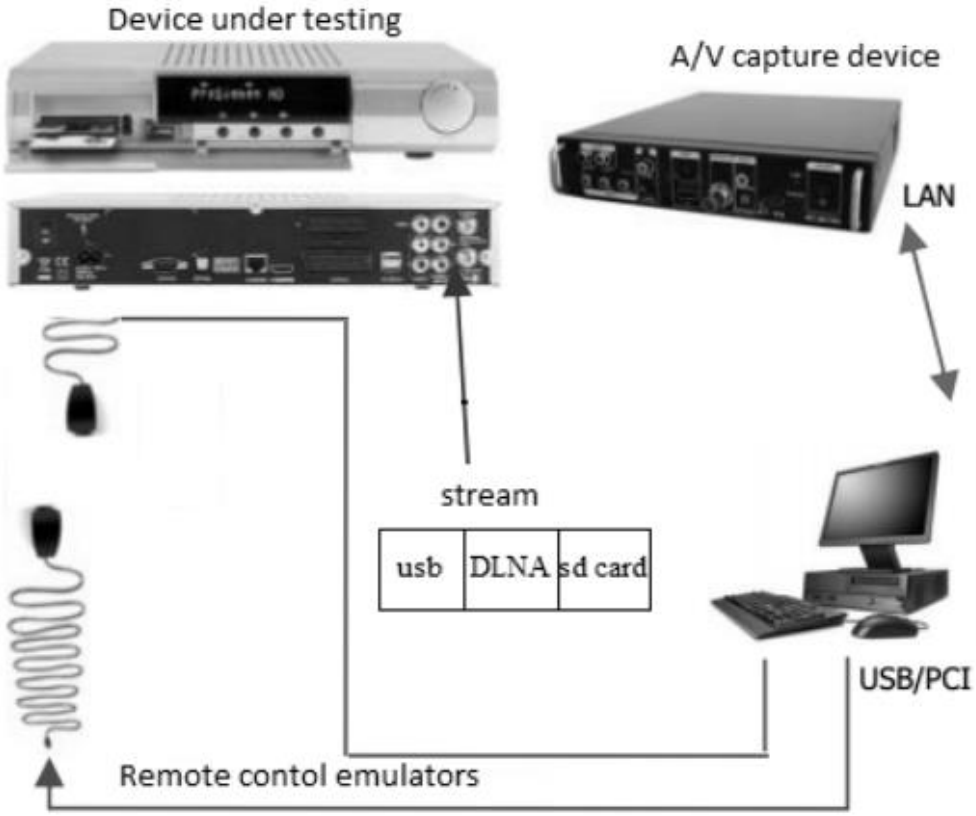


Figure 2.9 Overview of the proposed automated system (B. Kovacevic, M. Kovacevic, D. Stefanovic and M. Loncarevic, 2015)

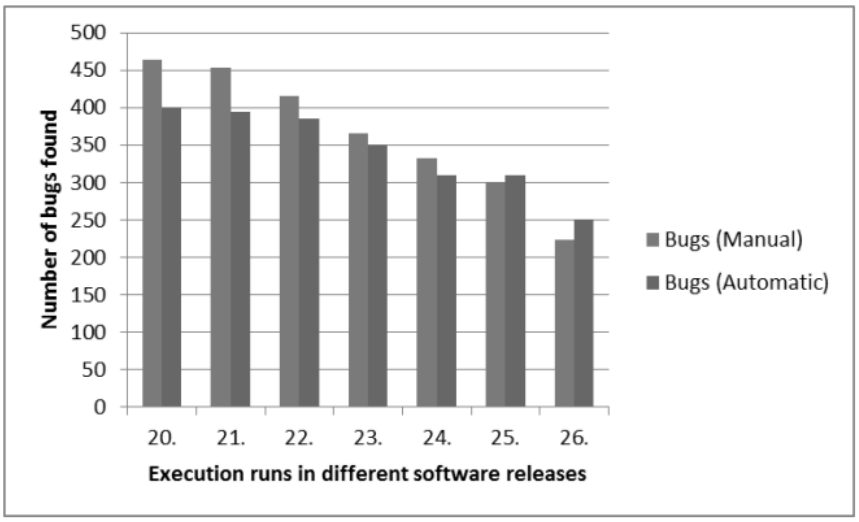


Figure 2.10 Automatic test (right) versus Manual test (left) results (B. Kovacevic, M. Kovacevic, D. Stefanovic and M. Loncarevic, 2015)

Based on Figure 2.10, it can be seen that using either method can produce almost the same bugs. This has proven that both tests were equally in testing the media players.

In conclusion, the proposed method (automated way testing) did boost the testing precision, save a lot of time and reduce manpower needed as compared to the manual way testing. Test reports including the graphs plotted in this proposed system were also generated through the automation so that any issue happened while running the test cases can be monitored easily. This paper recommended to have a less time-consuming during the generation of multimedia files.

2.2.7 Automated Marks Entry Processing in Handwritten Answer Scripts using Character Recognition Techniques

3 researchers from the team (Koushik, Chengappa and Chendan, 2022) utilized a unique tool namely optical character recognition (OCR) to automatically count the marks of the online examinations papers or assessments gathered from schools or universities using Python language. As we know, it is the Covid-19 outbreak starting period. Almost whole world had carried out an online teaching-learning (OTL) method. Undeniably, all coursework and examinations were also conducted online. As the number of students increases, the number of papers needed to be marked also increases, this leads to the amount of time needed to count the marks obtained by each student to be increasing significantly. Therefore, this team invented an automated marks entry processing system by scanning the image of the handwritten sheet. The key point here is the person who marks the papers has to use red pens as the system operates to detect the red-coloured digits including fractions. After that, all the red-digits numbers are

scanned, a sum of the marks will be generated. Figure 2.11 shows the block diagram of the system proposed by this team of researchers. Figure 2.12 shows the comparison between marks counting between OCR and manual (M) ways. IWR, ICR and CNN are the various kind of OCR methods.

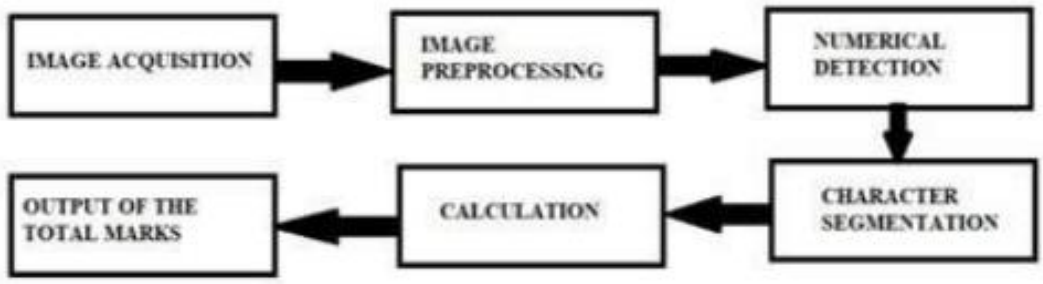


Figure 2.11 Block diagram of the image acquisition (Koushik, Chengappa and Chendan, 2022)

M	OCR (IWR)	M	OCR (ICR)	M	OCR (CNN)
5	5	3	3	2	2
4	4	9	1	1	1
2	2	5	5	6	6
7	7	1	1	3	3
1	1	6	6	2	2
M = Manual Counting O = OCR Counting					

Figure 2.12 Comparison between marks counting between OCR and manual ways (Koushik, Chengappa and Chendan, 2022)

Based on Figure 2.12, it is obvious that most of the automated ways (OCR) were able to duplicate the same counted marks as the manual way. According to the research, the time taken to count the marks were relatively shorter. In summary, this proposed OCR system not only saves the length of time needed to calculate the scores obtained, but also increasing the degree of accuracy such as the detection of fractions. Hence improving the overall efficiency. Furthermore, some tools from Python are used as well to analyse the captured images and then match the marks obtained based on the database invented by the team for the final results.

2.2.8 An Automated Test Generation Technique for Software Quality Assurance

An automated test was conducted and presented by a group of researchers (Dianxiang, Weifeng Michael, Lijo and Linzhang, 2015). This test or system was known as Model-based Integration and System Test Automation (MISTA). This was aimed to examine the software security and integrated functional systems. This method was using system under test (SUT) to automate the test generation by emphasising on the main features to be examined instead of gathering all behaviours of the software systems especially in term of fault detection. The main function of MISTA was to translate the test cases generated from the test model (software system) into the executable test code. MISTA varied in scripting languages such as Java, C language (including C++ and HTML. Figure 2.13 shows the context diagram of MISTA.

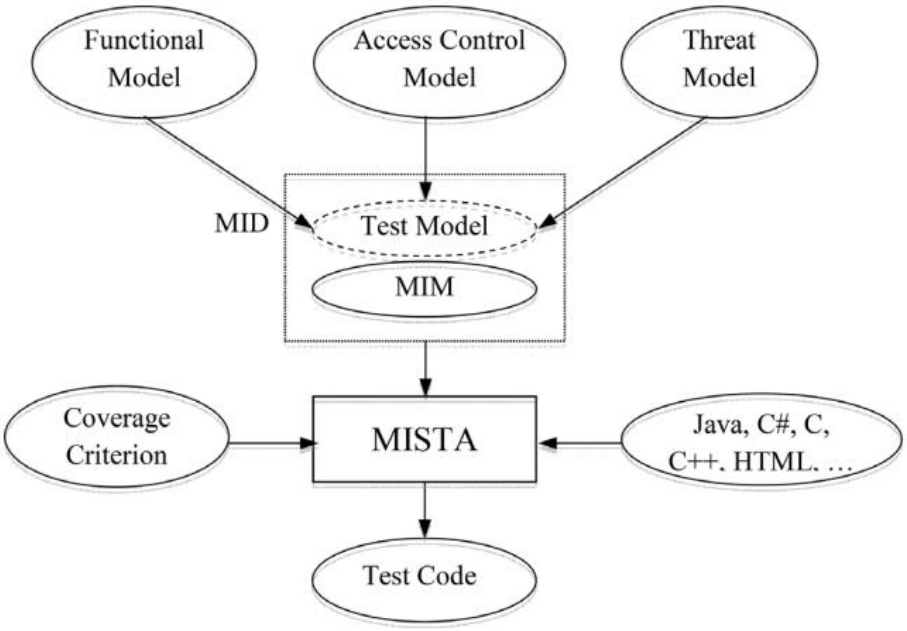


Figure 2.13 Context diagram of MISTA (Dianxiang, Weifeng Michael, Lijo and Linzhang, 2015)

To wrap up, the implementation of MISTA has greatly improved the efficiency and the effectiveness in detecting the fault in the software systems. Next, this implementation was able to produce executable test cases according to the criteria needed in the test models. MISTA also supports various kind of programming languages such as C language and Java. This method is also an extensible architecture (flexible), making it easily to introduce a brand-new test based on the requirements needed.

2.2.9 An Automated System to Calculate Marks from Answer Scripts

A group of 5 researchers from University of Bangladesh (Rizvee, Arefin, Khan, Islam and Rabbi, 2022) did an exclusive system for teachers who have to calculate the total score of the marked assessments manually. This literature was mainly to implement an automated script to sum up the handwritten marks using recognition Python tool to detect the handwritten characters. In other words, this proposed system eased the tasks of the teachers in counting the total scores obtained by each assessment. After identifying the red handwritten marks (or red pixel images), the identified marks (input) will be passed via a machine learning architecture (database provided by the researchers) in order to report the most precise marks. Figure 2.14 shows the summary concept or flow diagram of the developed automated mark calculation system.

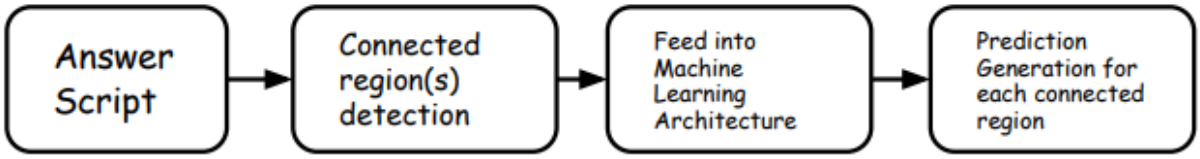


Figure 2.14 Summary of the developed automated mark calculation system (Rizvee, Arefin, Khan, Islam and Rabbi, 2022)

As a summary, this developed system which automatically count the total marks of the papers has dramatically improved the efficiency and accuracy in calculation and at the same time reducing the time taken needed. Other than this, this proposed system is having high flexibility. In other words, this system may be modified for future use such as detecting colours other than red or detecting other kind of characters such as alphabets or Latin.

2.2.10 Comparing the effort and effectiveness of automated and manual tests (An industrial case study)

A comparison experiment was done by Dobles, Martinez and Lopez (2019) between manual tests and automated tests. The main comparisons between these 2 methods were the effort such as test time consumption including the construction of codes and test execution time as well as effectiveness such as number of faults found during the test execution. The chosen software under test (SUT) is the set of Java web applications. Figure 2.15 and Figure 2.16 show the effort required and the number of failures detected by the way between automated test and manual test respectively.

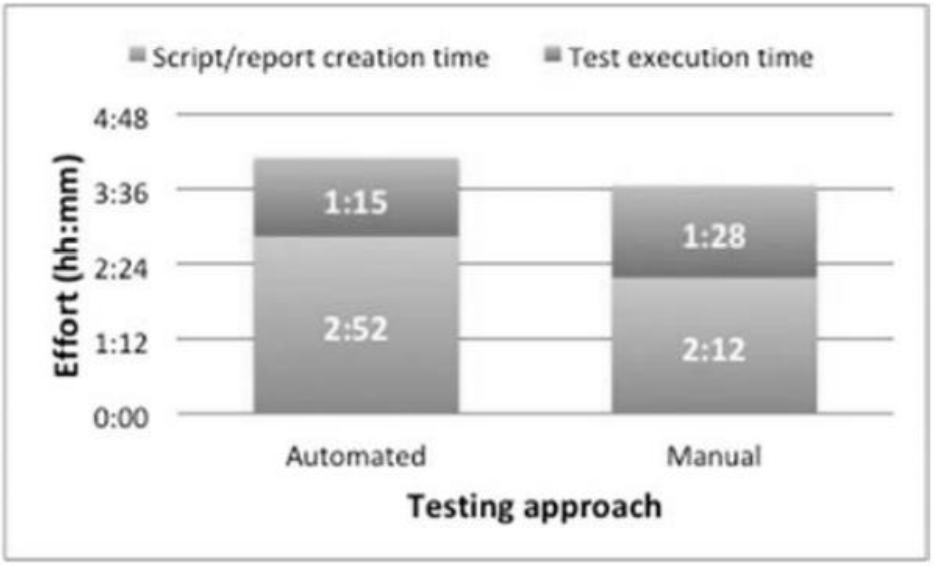


Figure 2.15 Effort required in developing script (top – creation time, bottom – execution time) between manual and automated ways (Dobles, Martinez and Lopez, 2019)

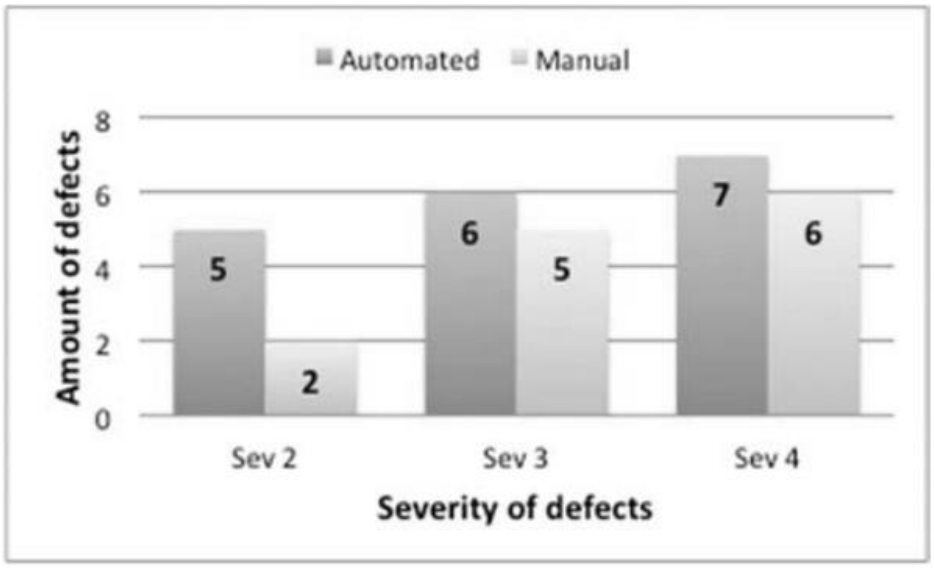


Figure 2.16 Number of failures detected by automated (left) and manual (right) ways (Dobles, Martinez and Lopez, 2019)

Based on the Figure 2.15, it is obvious that the time required to make the script for the automated way is longer than the manual way because of the coding knowledge criteria. However, this costs only in the creation time (for similar usage, only modification is needed, high flexibility). Once it is successfully developed, this "drawback" will be useful for regression runs especially in bug-hunting as compared to manual method. This is because manual testing required manpower which means he/she will get tired after working for a long period of time. As a result, the focus will be dropped due to exhaustion, indirectly unintentionally reject the test which could lead to a defect. As a summary, automated testing technique is much more effective and efficient than manual technique.

2.2.11 Automated Question Generation Tool for Structured Data

Shirudem Totala and Nikhar (2015) carried out a research tool to automatically generate questions for assessments or examinations based on structured data using Natural Language Processing (NLP). For any school, college or university, the teachers or lecturers have to brainstorm to set the questions for examinations or coursework for their students. This would take a couple of time and effort as well as cost as time goes because the question pools that would be thought of by the question writers will become limited. Thus, this research was aimed to implement an automatic question generation system which mainly focused in generating questions in English Language form in any type such as objective questions, subjective questions, data analyses and more given a scope of data. Once obtaining the domain of the data, the data will be pre-processed, then these data will be used as the template to generate the questions based on the criteria needed. Two main aspects were concerned in this research, that are Matched (questions that are the same as question writers) and Missed (questions that are different from question writers). Figure 2.17 shows the overall designed system flow diagram for the automated question generator system.

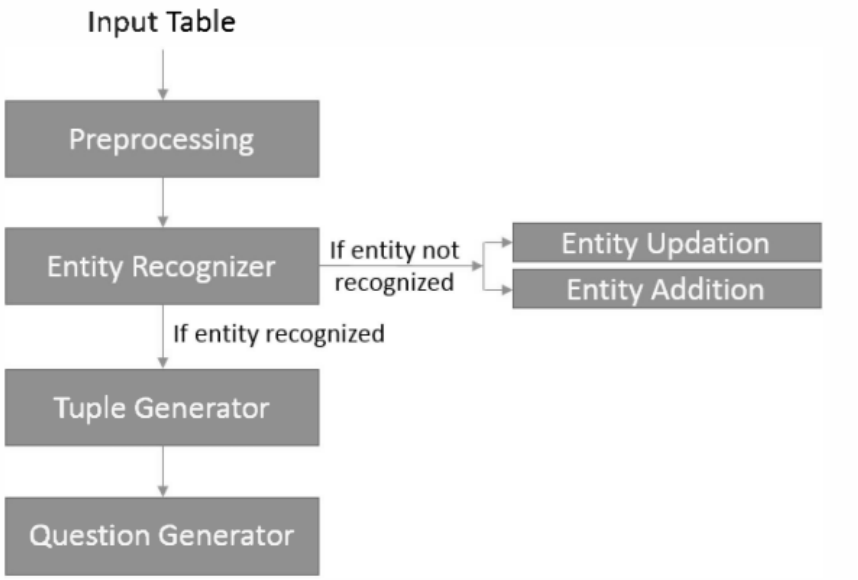


Figure 2.17 Designed system flow diagram for the automated question generator system (Shirudem Totala and Nikhar, 2015)

In conclusion, according to their research, this automatic question generations system did boost the compatibility and robustness of the questions without the needing of the brain effort from the question provider. The time taken to produce the same number of questions in a given time is obviously shorter than self-preparation method. In other word, the efficiency to generate the questions will be higher. The only issue faced here was just the grammatical error.

2.2.12 Semi automated and manual methods for counting cells expressing p75 receptor in endometriotic lesions

A total of 7 researchers (Agung, Xiaoyan, Jozsef, Rudolf, Anneliese, Ludwig and Beata, 2018) in a team conducted an experiment to compare the semi-automated way using a tool and manual way observed by different observers in counting the cell receptors in endometriotic lesions. The semi-automated cell analysis program tool that was used was the HistoQuest software while the manual way was to seek for 2 independent observers to calculate the cells quantitatively. The settings used for running both methods were the same, which consisted of exposure time and signal amplifications.

Based on the results gathered from this experiment, it has proven that semi-automated technique brought more advantages as compared to the manual observed technique. If the shape of nuclei is regular, the software can detect it very easily without any issue. However, if the shape of nuclei is irregular with colours overlapping, it leads to a failed result. For manual way, the observers were still able to detect it but not 100%, resulting in a poorer result. Nonetheless, the automatic method can save more time in detection and provide more precise as well as accurate result in a shorter period of time.

2.2.13 Research on Automated Testing Framework for Multi-platform Mobile Applications

Research was carried out by Da Zun, Tao Qi and Liping (2016) on automated testing framework by introducing a new framework for multi-platform mobile applications known as MATF (Multi-platform Automatic Testing Framework). As the name mentioned, it was an automatic method which utilised the keyword driven test technology. Hence, this did not require any manpower to brainstorm test scripts, indirectly lowering the price needed for testing. Other than this, it did save the effort and time spent to think of the scripts who must be corresponding to the test cases. Undeniably, this could help to reduce any false failure found from test scripts. In term of costs saving, this proposed framework was implemented to target different test scenarios or platforms, here it was targeted to iOS and Android platforms. Therefore, the testing costs were decreased, the test cycles were also decreased as well as boosting the rate of production. Figure 2.18 shows the summary of MATF framework.

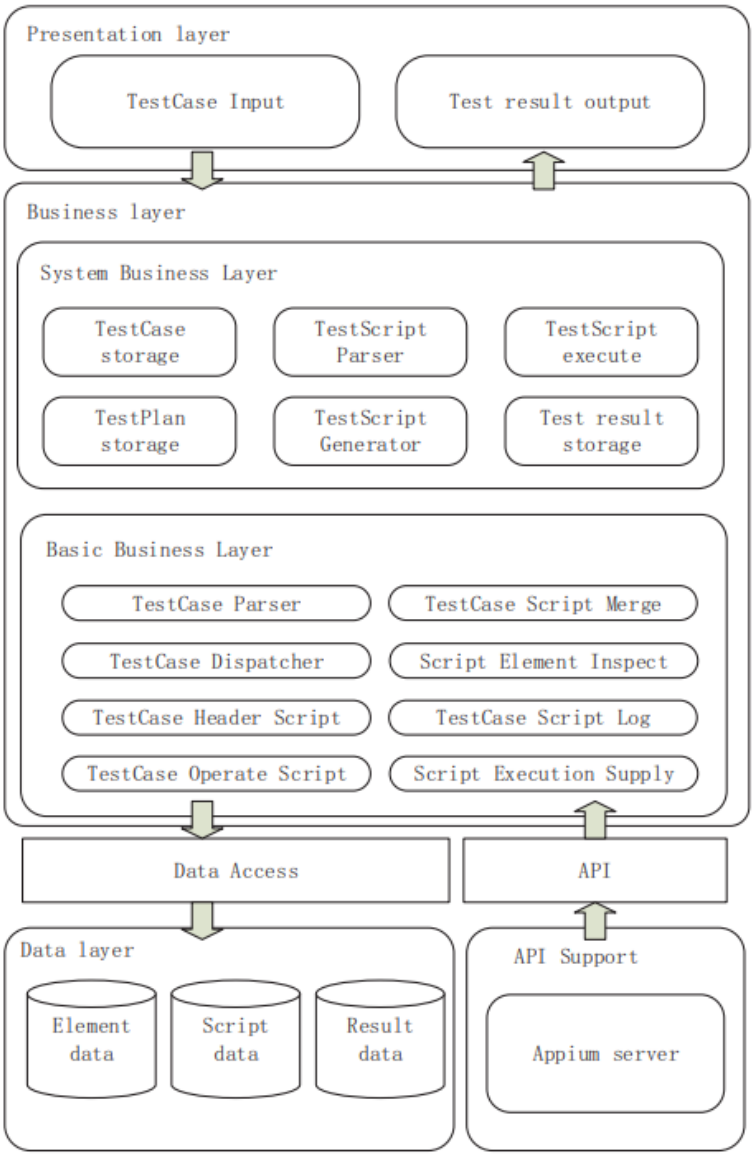


Figure 2.18 Summary of MATF framework (Da Zun, Tao Qi and Liping, 2016)

To sum up the information provided, this MATF framework was developed so that it would be applied in the different mobile platforms by only using this test script. Consequently, this had saved the time taken to write the test script, the effort needed to think of the possible steps and outcomes for the designed test script and the costs needed when being applied in the different fields as compared to the manual technique. In short, this had greatly increased the effectiveness by using the automated testing. However, with complex test in the mobile applications, this would lead to a failure.

2.2.14 Automated Exam Paper Marking System for Structured Questions and Block Diagrams

As a teacher or a lecturer, marking examination papers or assessments could be very time-consuming and effort exhaustion if the number of papers is a lot. Therefore, a research team including P. Jayawardena, Thiwanthi, Suriyaarachchi, Withana and C. Jayawardena (2018) conducted an experiment to automate the marking system for structured questions (typically essay questions) and block diagrams. Natural Language Processing (NLP) was utilised to check the essay questions. To target diagram type answers, Depth First Search (DFS) and Breadth First Search (BFS) will be used. For flow charts and logic circuits, they will be converted to Python programs for marking purpose. All of these structures will be embedded to the proposed system. Figure 2.19 and Figure 2.20 show the comparison of marks allocation between the proposed system and the examiner for block diagram and logic circuit questions respectively. Range of difference means the difference of score obtained which is marked by the examiner and the designed system.

Range of Difference (in marks)	Number of Answers
0 (same mark)	13
0.5	5
1 <mark <= 2	3
3 <mark <= 5	2
5 <mark <= 7	2
Mark >7	0

Figure 2.19 Marks allocation between the designed system and the examiner for block diagram questions (P. Jayawardena, Thiwanthi, Suriyaarachchi, Withana and C. Jayawardena, 2018)

Range of Difference (in marks)	Number of Answers
0 (same mark)	47
0 <mark <= 1	3
1 <mark <= 2	5
2 <mark <= 3	3
3 <mark <= 5	4
5 <mark <= 7	4
Mark >7	0

Figure 2.20 Marks allocation between the designed system and the examiner for logic circuit questions (P. Jayawardena, Thiwanthi, Suriyaarachchi, Withana and C. Jayawardena, 2018)

Based on Figure 2.19 and Figure 2.20, it is convinced that there was not much difference of the scores obtained with only minute difference between the automated system and the examiner (manual way) even though the accuracy was not 100%. Subsequently, the final results provided by the automated system are almost the same as the manual method but the time taken to mark all the samples will be lesser. Since the number of samples in this research is less, the effort consumption is not a lot.

2.2.15 IoT based Automated Examination Management System with Biometric Portal

Research was carried out by Tabassum, Ahsan, Chowdhury and Basu (2022). They designed an automated system for examination management such as schedules of the examination, seating as well as location allocations, in-charged invigilators and more. In other words, the main aim was to reduce the time taken for providing the information aforementioned manually, thus increasing the efficiency of this practice. This was a combination of the automated practice as well as the IoT (Internet of Things). The front-end consisted of NodeMCU Microcontroller which was assembled with a Fingerprint module for biometric data and a motorised script system. While for the back-end, a local host server was needed to be embbeded on Xampp using PHP and MySQL languages to save the biometric data of the students who would sit for the examination, hence producing an examination script for them. Figure 2.21 shows the flow chart of the proposed automated examination management system. Figure 2.22 shows the results obtained for fingerprints enrolment between the manual technique (based on the examiners) and the automated technique (using the proposed system).

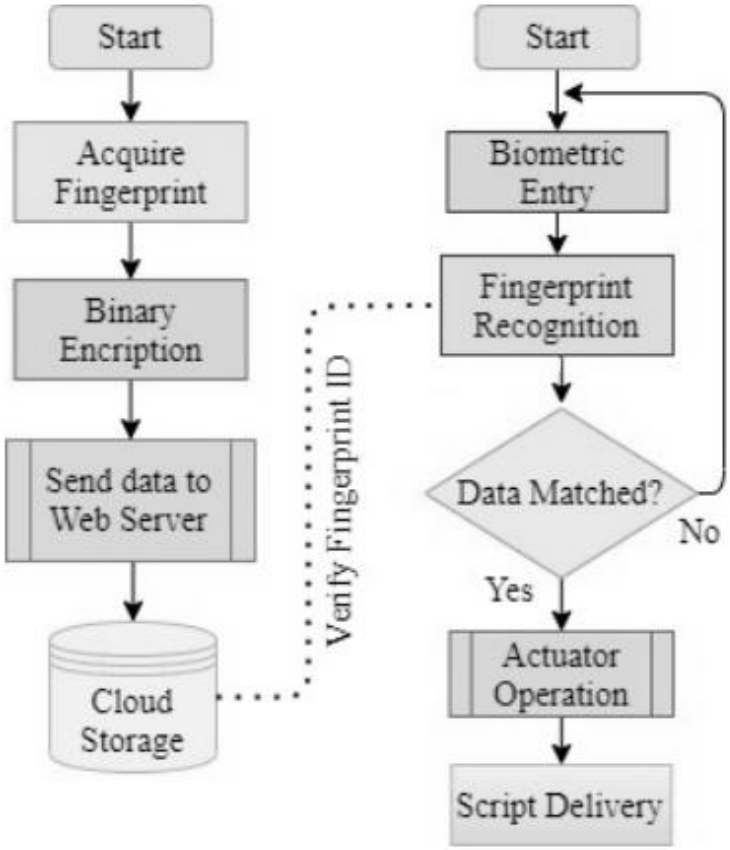


Figure 2.21 Flow chart of the designed automated examination management system (Tabassum, Ahsan, Chowdhury and Basu, 2022)

Method	Tests		
	Average time taken	***Inconvenience***	***No. of attempts***
Manual	10-12 minutes	3-4 times	15
Using our Device	2–3 minutes	Once	15

Figure 2.22 Results obtained for fingerprints enrolment between the manual technique (based on the examiners) and the automated technique (using the proposed system) (Tabassum, Ahsan, Chowdhury and Basu, 2022)

According to the results gathered as shown in Figure 2.22, it is proven that the time taken can be greatly decreased by 4 times if the automated system is utilised for enrolling the fingerprints of the examinees given the number of attempts for both techniques are the same. In term of reliability, the inconvenienced caused is just once for automated system as compared to the manual way. To conclude this, the proposed automated examination management system brings a lot of benefits for all examinations such as high accuracy, high efficiency, high robustness, high modularity and high effectiveness (time taken is shorter).

2.2.16 Fully Automated Regression Tool for Post Silicon Validation

As time passes, the semiconductor industry invents the chips with the size as small as it could be and making the design to be more complex. As we know, it is not easy to validate the designed chip in a short period of time. In order to meet this criterion, a research team with 3 design engineers (Kansal, Sinha and Jaiswal, 2017) implemented an automated regression tool in the post silicon validation environment. Python scripting was selected in this case. This proposed tool required only a single click in order to kick start the automation aforementioned. This was aimed to fully replace the manual technique used nowadays and simultaneously save the time needed starting from code compilation, followed by test execution and lastly results consolidation. Figure 2.23 shows the flow chart of the designed automated tool while Figure 2.24 shows the output statuses for every test case run using the designed automated tool.

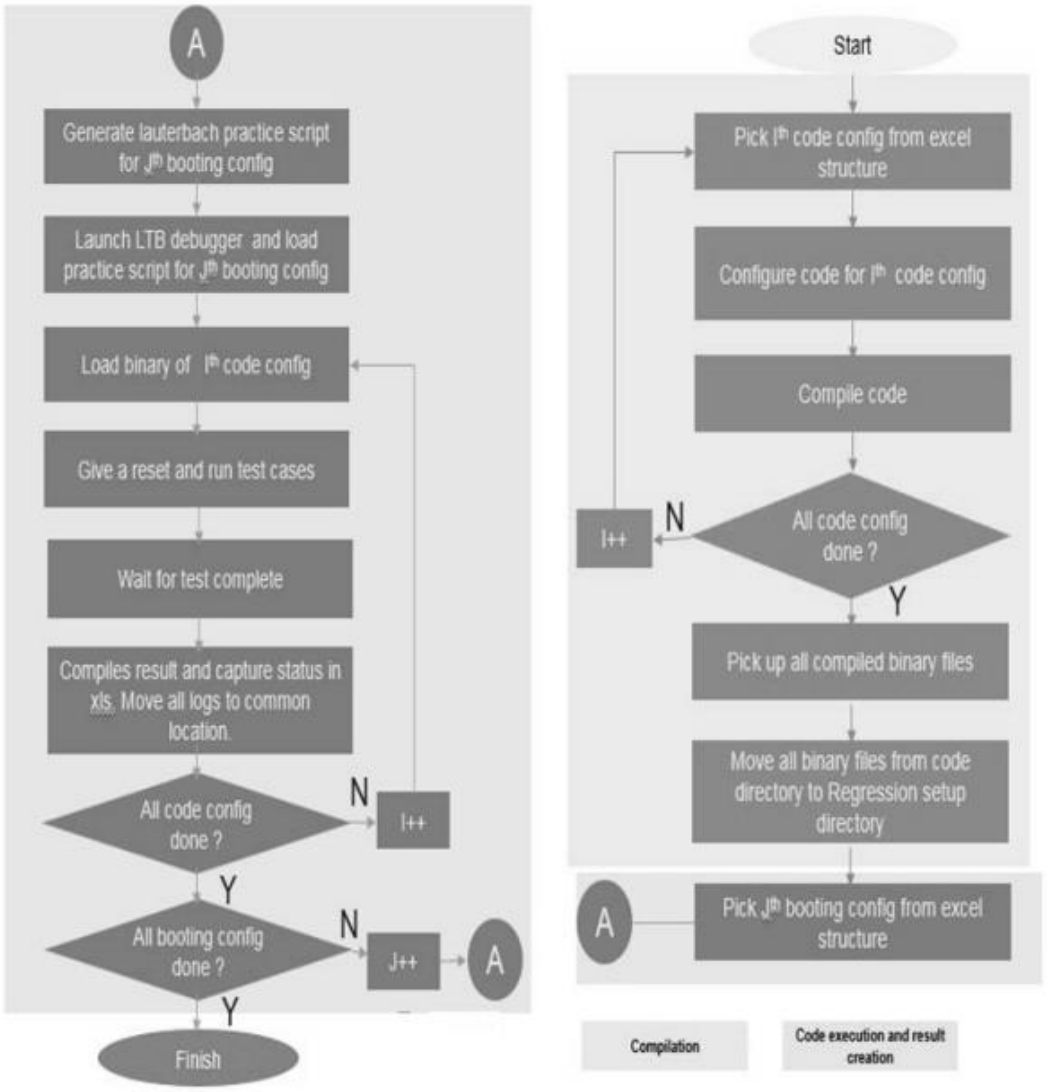


Figure 2.23 Flow chart of the designed automated tool (Kansal, Sinha and Jaiswal, 2017)

Configuration_1						
Test_name	Clocking_mode_status	Power_mode_status	Executing_core_status	Test_Case_status	Overall_status	Log_link
1_Test_Case_1	PASS	PASS	PASS	PASS	PASS	LINK_1
2_Test_Case_2	PASS	PASS	FAIL	PASS	FAIL	LINK_2
3_Test_Case_3	PASS	PASS	PASS	TIMEOUT	FAIL	LINK_3
4_Test_Case_4	FAIL	FAIL	FAIL	FAIL	FAIL	LINK_4

Figure 2.24 Output statuses for every test case run using the designed automated tool (Kansal, Sinha and Jaiswal, 2017)

According to Figure 2.24, if the test case running environment i.e. internal regulation mode is different from what it is supposed to be i.e. external regulation mode, the final output will be stated as FAIL based on configuration checks. TIMEOUT will be an extra configuration under the status if the regression hung at somewhere else in an unpredictable scenario.

As a summary, this proposed automated regression tool bring useful advantages over the manual technique. First of all, the automated way can the effort spent by the designer and shorten the time to figure out the corner cases as compared to the manual way. Next, it is flexible, meaning the written script can be used across other projects without any much modifications. Few clicks are only needed to modify the current configurations such as feature enabling or disabling with one click only. As a result, the automated technique becomes an easy way with detailed analyses provided including checks performed.

2.2.17 Automated Testing of the Medical Device

Roja and Sarala (2017) conducted an experiment to design the automated stress testing of the medical device. They used the LabVIEW software to implement the switches which functioned automatically of the medial device. In order to operate the designed automated medical device, Python was chosen for test scripting. To manually press the switches for stress testing, it would squander time and tedious. This was because it could not be confirmed that the force used to press the button would be consistent over a specific time. Furthermore, to rapidly toggle the button, it would cost a lot of effort and patience if it was done manually since constant speed was the minimum requirement for the stress testing. Inevitably, automated method was much recommended. Figure 2.25 shows the flow chart of the proposed automated system for stress testing in the medical device. Figure 2.26 shows the comparison between the manual and automated testing of the medical device in term of speed.

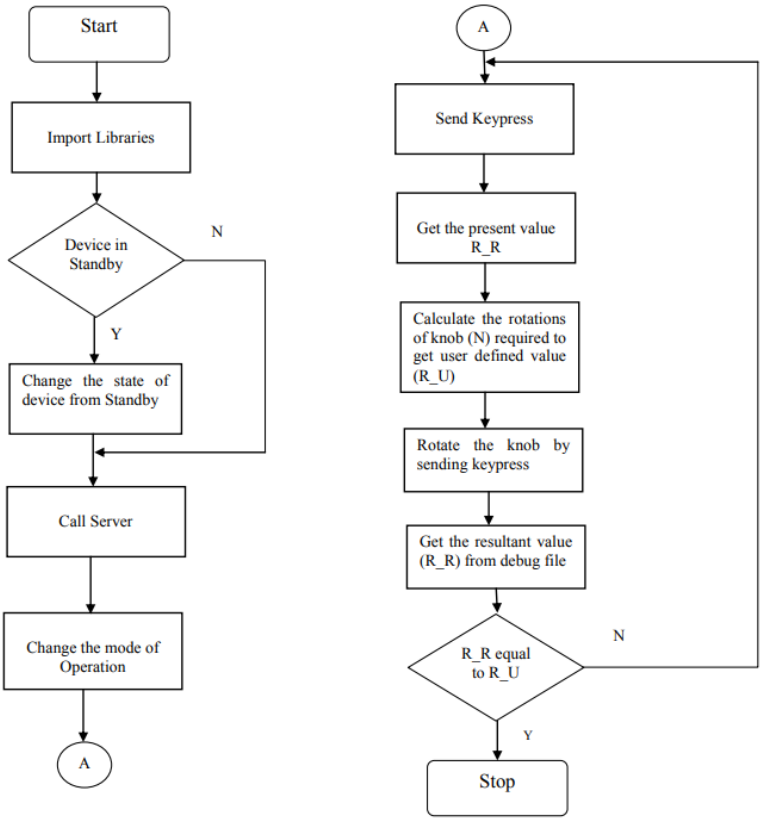


Figure 2.25 Flow chart of the proposed automated system for stress testing in the medical device (Roja and Sarala, 2017)

Number of Iterations	Manual testing speed (sec)	Automated testing speed(sec)
10	100	50
100	1000	500
1000	10000	5000
10000	100000	50000

Figure 2.26 Comparison between the manual and automated testing of the medical device in term of speed (Roja and Sarala, 2017)

Based on the results obtained from Figure 2.26, it is found out that the automated testing requires lesser speed (around 2 times lesser) than the manual testing speed given a constant number of iterations performed. In short, there are more pros over cons in automated testing. This includes high effectiveness (less time-consuming), high flexibility (can be applied for other medical testing with small modifications), lower cost, minimum erroneous data produced and high reliability. The only problem faced here is this requires knowledge and skills in implementing the test scripts such as medical feature and Python programming language.

2.2.18 Developing Reference Help Documents Automatically by Using Scripting Methods

Kulkarni and Madhavi (2016) developed Reference Help documents in an automatic way through the use of HTML scripting methods. This was aimed to reduce the effort consumption while scanning the Reference Help Guides (RHG) by the users. They performed a flow of tasks in order to implement the RHG. Firstly, extracting the required and related information from a source. Next, processing the data information gathered and formatting them by enhancing these with a test script, a Java script, such as sorting the information and a searching tool. Lastly, the users could select any format they wished to have through the execution of the scripts. The scripts that were chosen here is the Perl scripts. The summary of the flow is shown in Figure 2.27. Figure 2.28 shows the comparison of the manual method versus the automated method (or scripting method).

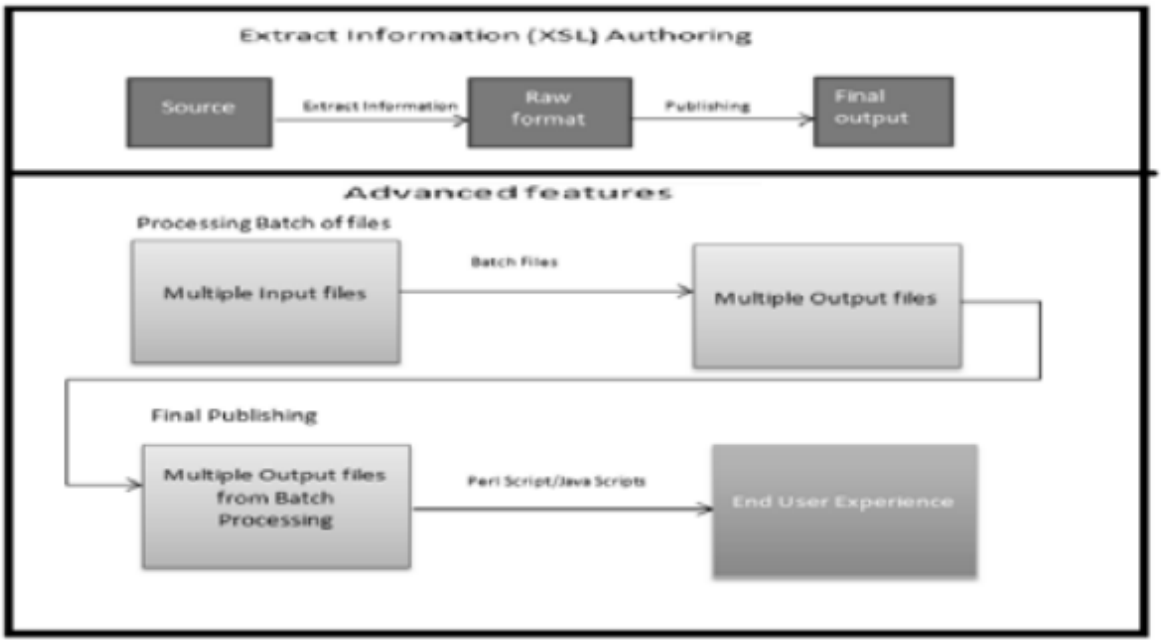


Figure 2.27 Summary of the proposed flow for scripting method (Kulkarni and Madhavi, 2016)

	Manual Effort	*Scripting Method*
Time	More time consuming	Less time
Accuracy	Prone to errors	Quality output
Resources	Multiple writers	One writer
Dependency	High	Low
Technical Reviews	Reviews mandatory	Not mandatory
User Experience	Difficult to enhance	Can be enhanced

Figure 2.28 The comparison of the manual method versus the automated method (Kulkarni and Madhavi, 2016)

In conclusion, based on the comparison as shown in Figure 2.28, it is obvious that scripting or automated method are more powerful than the manual method in developing the RHG. Other than this, the automated method is also applicable to most of the systems because it could save a lot of manpower, cost and time, leading to have a higher output produced, indirectly increasing the efficiency. Erroneous data could be decreased because the false failure could be due to the exhaustion in the human beings.

2.2.19 Automated Programming Assignment Marking Tool

A team of IT engineers (Vimalaraj, Thenuwara, Wijekoon, Sathurjan. Reyal. Kuruppu, Tharmaseelan, 2022) from Sri Lanka conducted research on automated programming assignment marking tool. Nowadays, the registrations of students to the programming course have increased from time to time. Indirectly, the number of programmes coded by students has also increased. Hence, the lecturer in-charge has to spend his/her time to evaluate the students' modules. The usual way would be running the script written by the students but this way was not ample in evaluating the marks for students. For much better efficiency, the lecturer would have to read the codes line-by-line. However, this manual evaluation might be very tedious, time-consuming as well as tiring, leading to an inaccurate and less precise marking. Therefore, this team proposed an automated technique to mark the assignments automatically in order to reduce the burden bear by the lecturers in Java script. Figure 2.29 shows the overall flow diagram of the automated assignment marking tool in which DB stands for database while Figure 2.30 shows the comparison scoring results between human being (invigilator) and grading model (proposed automated tool).

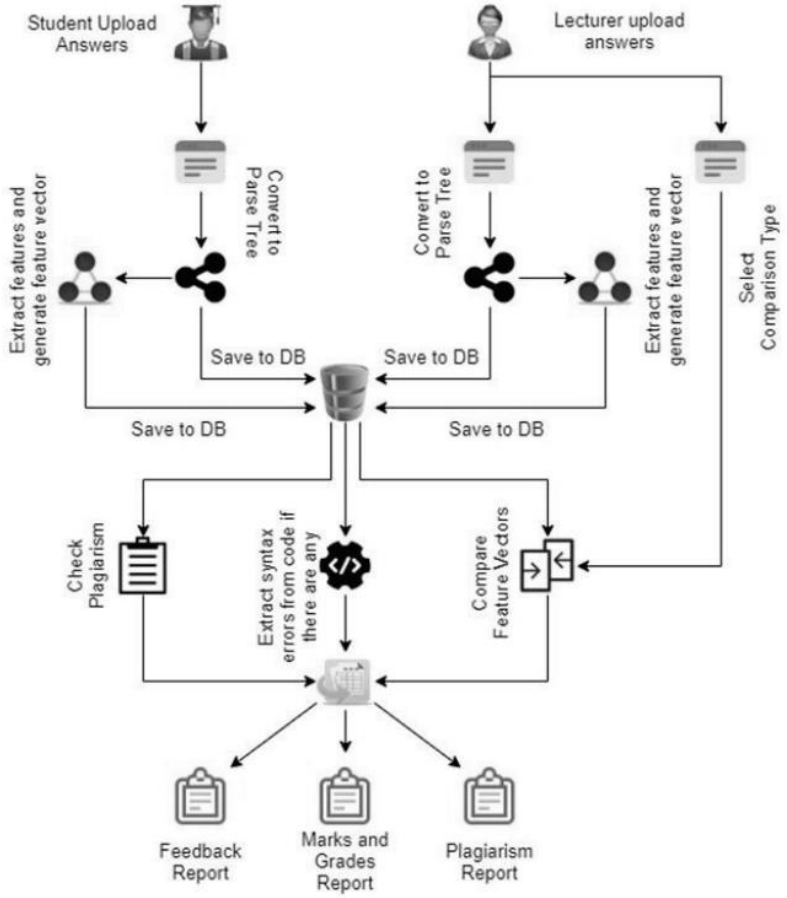


Figure 2.29 Overall flow diagram of the automated assignment marking tool (Vimalaraj, Thenuwara, Wijekoon, Sathurjan. Reyal. Kuruppu, Tharmaseelan, 2022)

		Marks assigned by human		Marks predicted by the grading model	
Question Number	Student Number	Exact Match	Functionality Match	Exact Match	Functionality Match
1	1	80	80	78	79
1	2	76	75	67	69
1	3	70	80	70	79
1	4	73	80	72	80
1	5	75	75	78	80
1	6	70	70	64	66
1	7	75	75	71	80
1	8	75	80	72	75
1	9	75	75	77	77
1	10	75	75	74	80

Figure 2.30 Comparison scoring results between human being (invigilator) and grading model (proposed automated tool) (Vimalaraj, Thenuwara, Wijekoon, Sathurjan. Reyal. Kuruppu, Tharmaseelan, 2022)

The summary of the flows of this implemented tool by the research team as shown in Figure 2.29 would be (1) Coding conversion to parse trees. (2) Features extraction and feature vectors generation, (3) Comparison between them from part (#2) and lastly (4) Marks allocation and plagiarism report generation. According to Figure 2.30, it can be concluded that the deviation between the marks allocated by the examiner and the designed tool is very low. This means that the automated tool has a very high accuracy for this feature. Other than this, this designed system prepares plagiarism level checks between all answer sheets, making this system to be more powerful. In other words, the lecturer does not need to spend the time and effort in marking and comparing all the programmed scripts.

2.2.20 Automated Mechanical Simulation System for Microelectronic Packaging

An experiment was conducted by Jianfei and Jianwei (2015). They constructed an automated mechanical simulation system for microelectronic packaging using HyperWorks and Tcl/Tk scripts known as AutoSim. This designed system allowed the users to key in required parameters in the GUI. After that, the system was able to ran some simulation testing automatically which consisted of geometry model implementation, meshing, properties of materials, loading applications and post-processes. Figure 2.31 shows the overview flow of the AutoSim tool.

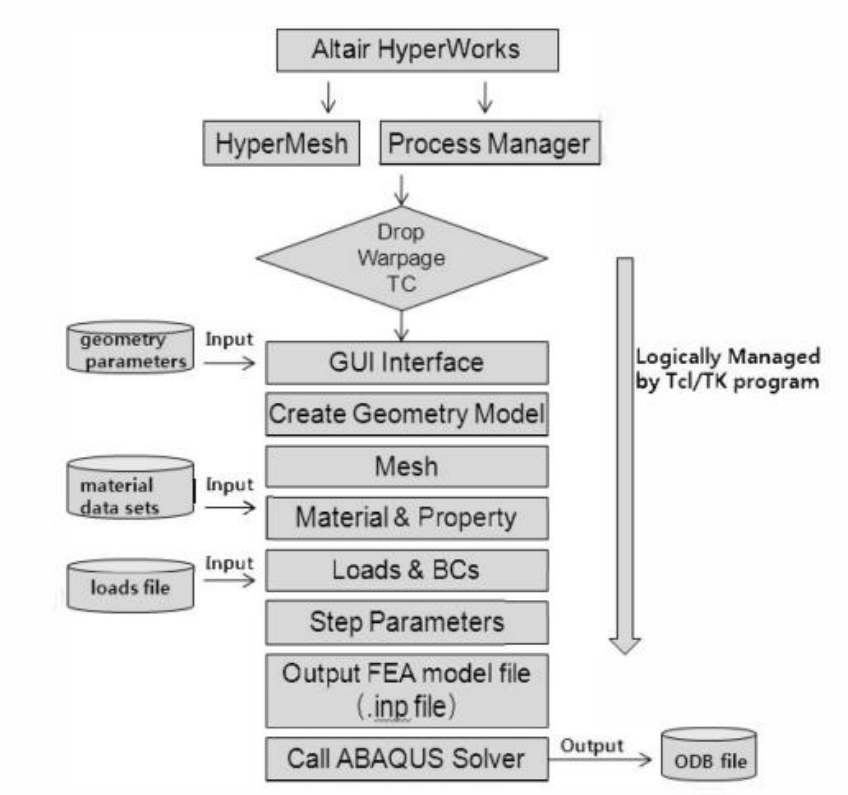


Figure 2.31 Overview flow of the AutoSim tool (Jianfei and Jianwei, 2015)

Figure 2.32 shows the comparison results between 3 simulated evaluation processes: Drop, TC (temperature cycling) and Warpage in the automated way and manual way in term of simulation time. The reduction time is also calculated between these 2 ways for easier comparison.

Simulation Process	Method	Create Model & Mesh	Load & BCs	Solve	Total Time Reduced
Drop	Manual	8-16hrs	0.5-1hrs	8-10hrs	50%~60%
	Auto	0.15-0.5hrs		8-10hrs	
TC	Manual	8-16hrs	0.5-1hrs	8-10hrs	50%~60%
	Auto	0.15-0.5hrs		8-10hrs	
Warpage	Manual	3-6hrs	0.5-1hrs	4-8hrs	40%~60%
	Auto	0.15-0.5hrs		4-8hrs	

Figure 2.32 Simulation time comparison between Drop, TC and Warpage in the manual and automated methods (Jianfei and Jianwei, 2015)

Based on Figure 2.32, it is shown that using the automated technique in any of the process will greatly decrease the time taken for microelectronic packaging by approximately half. In other words, this does increase the effectiveness and efficiency while doing the packaging process since the volume of the packaging will be at least thousand. In summary, the automated process will be undeniably recommended due to its fast-job process and cost-effective feature when dealing with a huge volume of packages.

2.2.21 NLP-based Automatic Answer Evaluation

Sinha, Yadav and Nerma (2022) conducted research on Natural Language Processing (NLP) based automatic answer evaluation. The purpose of conducting this was aimed to reduce the time needed and avoid the tedious process when marking the answer scripts by the teachers. As stated in the research, the marks allocation for an examinee not only came from the answer script, but it could have a small chance due to the current mood of the examiner as well as the relationship between the examiner and the examinee. To be worse, this could have led to an unavoidable mark deduction by a student and most of the time he or she did not have any idea why this score was obtained. Thus, this team came out this idea to convert this manual technique into more advanced way, that is an automated system. Figure 2.33 shows the designed automated system for the answer script evaluation.

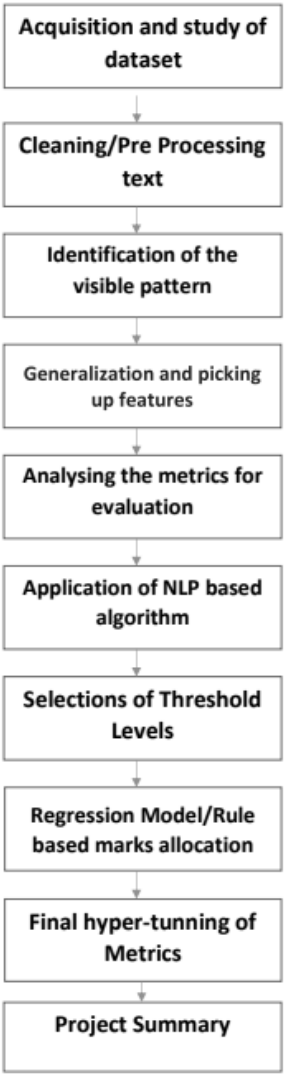


Figure 2.33 Designed automated system for the answer script evaluation (Sinha, Yadav and Nerma, 2022)

The summary of the flow would be firstly, all the handwritten or typed answers in the script will be scanned and extracted. Then, these answers will be used to compare between the correct answers which was initially saved in the database. While running the comparison flow, marks would be given at the same time. Lastly, the final score would be produced and a summary will be generated based on the keywords that were captured in the answers.

As a summary, this proposed automated system has proven that the scores obtained by each answer script similar to the one which is marked by the teacher. This means the automated system has a very high accuracy in doing this feature, in subjective question form. Furthermore, this designed system has also quickened the process of evaluating the answers, allowing the examiner to spend a lesser time to checks all the answer scripts.

2.2.22 AutoEval: An NLP Approach for Automatic Test Evaluation System

4 researchers from India (Agarwal, Kalia, Bahel and Thomas, 2021) developed an automated approach for automatic test evaluation system known as AutoEval. As a teacher or a lecturer, it is very difficult to evaluate the assessments if the number of students is large because he or she has to spend the time and effort, in short manpower, to mark the answer scripts and give the marks based on the marking scheme. Other than this, some external factors such as the examiner may be in a bad mood behaviour or there is a unique relation between the examiner or the examinee, resulting in the marks allocation to have an unfair result. Hence, this group of researchers proposed an automated technique system so that this would replace this repetitive manual evaluation method in the future. A common method known as MLP (Natural Processing Language) was utilised in this system. It not only could evaluate the answer scripts based on the theory but also check for grammar, syntactic analyses and similarities between the submitters. Figure 2.34 shows the overview of the block diagram of AutoEval.

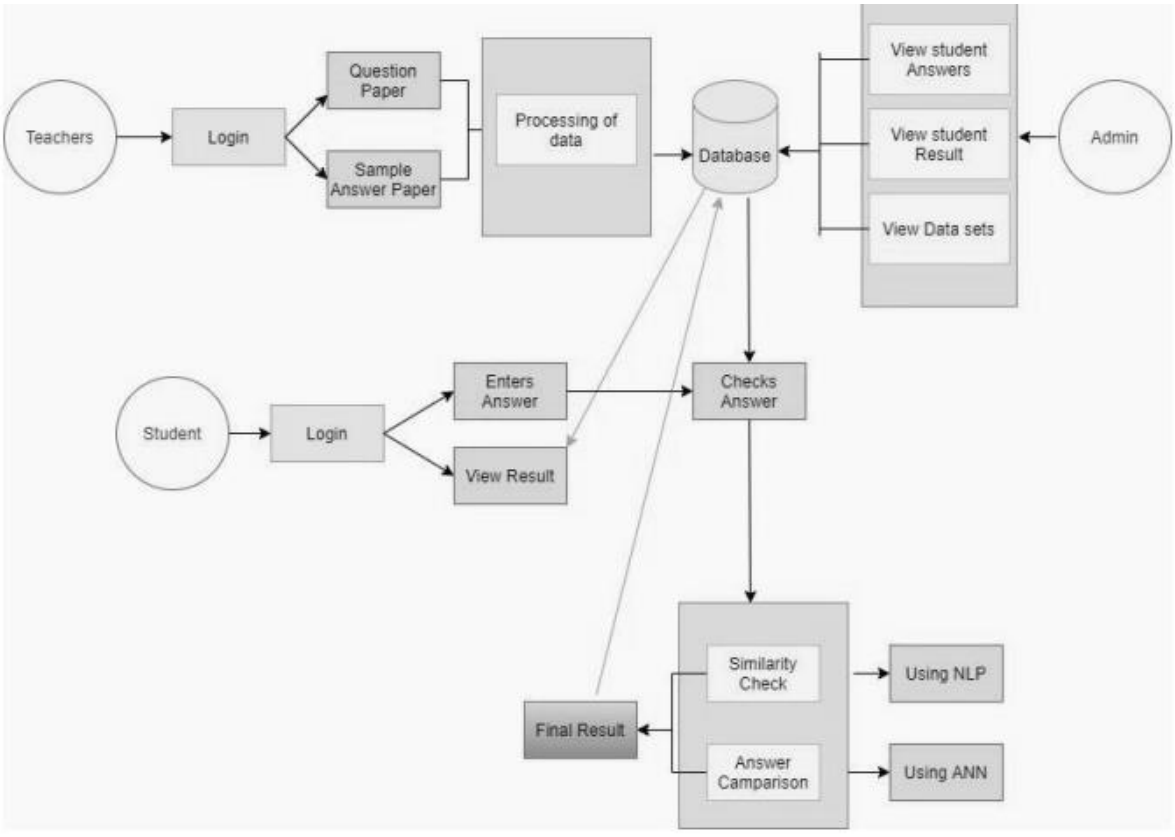


Figure 2.34 Overview of the block diagram of AutoEval (Agarwal, Kalia, Bahel and Thomas, 2021)

In order to run the AutoEval, here is the steps. Firstly, the students have to submit their answers by uploading to the system. Next, a comparison will be made between the uploaded answer and the answer provided by the teacher saved in the database. Evaluation of the answers starts next to check the similarities and hence the score allocation. Lastly, the final mark can be viewed at the last page.

According to this research, a basic output was prepared between the manual assessment by the teacher and the assessment evaluated by the system (also known as automated way). The manual technique took about 1 minute while the designed automated technique needed only 15 seconds respectively to respond a response. In other words, 3 times of the times was saved by using the proposed automated system, hence increasing the overall effectiveness and efficiency in evaluating the assessment. Subsequently, the manpower can be obviously saved because using the manual technique, the teacher needs some time to think before evaluating the response. Maintenance cost of this proposed system is minimum as it is only single time investment and the overall cost is cheaper since a system can function at all time while a human being cannot.

2.2.23 A New Marking Technique in Semi-Automated Assessment

Across most of the researches, almost half of them discussed about the automated marking method in evaluating the assessment using Python tool. For this research, the researchers (Buyrukoglu, Batmaz and Lock, 2017) did the same field. They introduced a semi-automated approach in the assessment marking system for program codes which was aimed to save the time and effort spent by the marker. The reason behind to implement a semi-automated technique over manual technique and fully-automated technique was because this will provide detailed feedbacks while the examiner was evaluating the assessment rather than providing a general comment based on the assessment. In short, the examiner was able to reuse the previous comments for the similar works, thus decreasing the time needed and reducing the workload. The first stage was the segmentation process. This was aimed to collect the program codes. The next step was the codifying process. As the name mentioned, this stage was to normalise the code by following the general programming rules. This was also to increase the similarity of the various codes gathered. After that, it was the grouping process in which comparing all the codes collected based on the similarities. This was of utmost importance process because this was a preparation stage for the marking stage so that the feedback can be provided in a

consistent way. The last stage was the marking stage. It was obvious to be an assessment evaluation as well as providing comments for the assessment. By and large, this proposed semi-automated marking system was satisfied by 75% of the participants. For the feedback system, it was 17% higher. This has proven that not all system has to be fully-automated, it can be just a semi-automated but still depending on the field specified. Figure 2.35 shows the overview of the proposed semi-automated marking system.

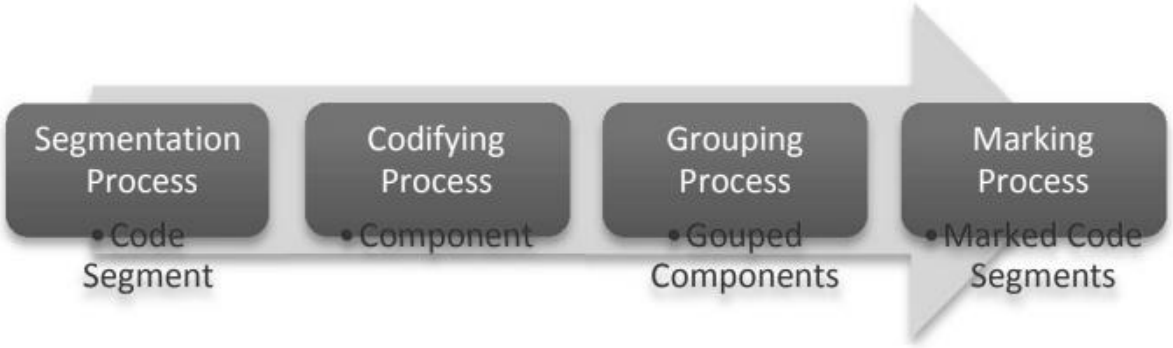


Figure 2.35 Overview of the proposed semi-automated marking system (Buyrukoglu, Batmaz and Lock, 2017)

As a summary, the existed detailed feedback system can greatly help to reduce the workload faced by the examiner, indirectly saving his or her time in evaluating the assessment and increasing overall efficiency of marking the assessment. However, some issues that are found here consists of sequence of marking the assessment between the manual and automated technique, challenges in grouping process due to variables used different and reusable comments which may not be correct even for the similar codes.

2.2.24 Automation of Reflectarrays in HFSS Using Visual Basic Scripting

A researcher named Tariq (2018) conducted research to automate the reflectarrays in HFSS (high frequency structure simulator) using Visual Basic (VB) scripting method. To satisfy the demands by the clients in the aspects of greater volume and coverage, high gain reflectarrays are required. However, for huge reflectarrays, approximately thousands of elements are needed. It is almost impossible to manually locate all the elements due to the dimension of each element is not exactly the same with each other in the simulation process for prediction prior to fabrication stage. In order to curb this issue, Tariq implemented this automated system using VB script technique. Figure 2.36 show the steps for the automation of reflectarrays in HFSS.

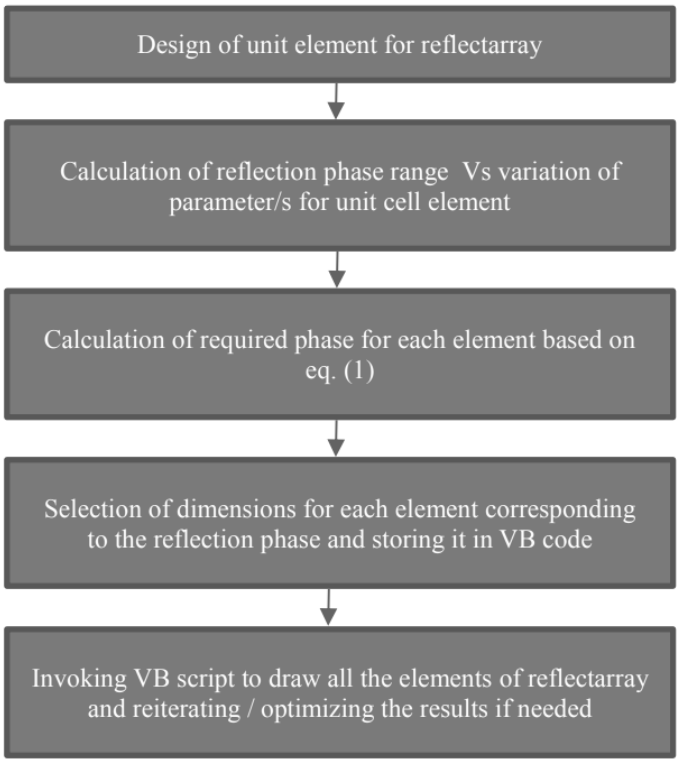


Figure 2.36 Steps for the automation of reflectarrays in HFSS (Tariq, 2018)

In summary, the designed automated system has greatly eliminated the tedious work needed to draw the elements manually in order to obtain a realised gain of reflectarrays for application usages. Other than this, the VB script is flexible to most of the software with few modifications needed but the minimum requirement for that software is it has to recognise the script.

2.2.25 Automated Assessment of Multi-Step Answers for Mathematical Word Problems

3 researchers (Kadupitiya, Ranathunga and Dias, 2016) from University of Moratuwa, Sri Lanka carried out research in implementing a system that would evaluate the mathematic field questions automatically. This designed system was aimed to boost the efficiency in grading the answers submitted in XML form by the students especially in shortening the time and effort needed while evaluating the answers. Other than this, this proposed system has a unique feature whereby if the answer marked is wrong, this system can locate the place and provide the guidelines on the ways to answer that question correctly. In other words, the students were able to check on the steps in answering the past year papers. For this system, it comprises 4 modules namely schema file handling module, expression validation module, unit validation module and lastly grading module. Figure 2.37 shows the flow chart of the proposed automated system. Figure 2.38 and Figure 2.39 show the grading method used between manual and automatic techniques to evaluate question 01 and question 02 in graphical form.

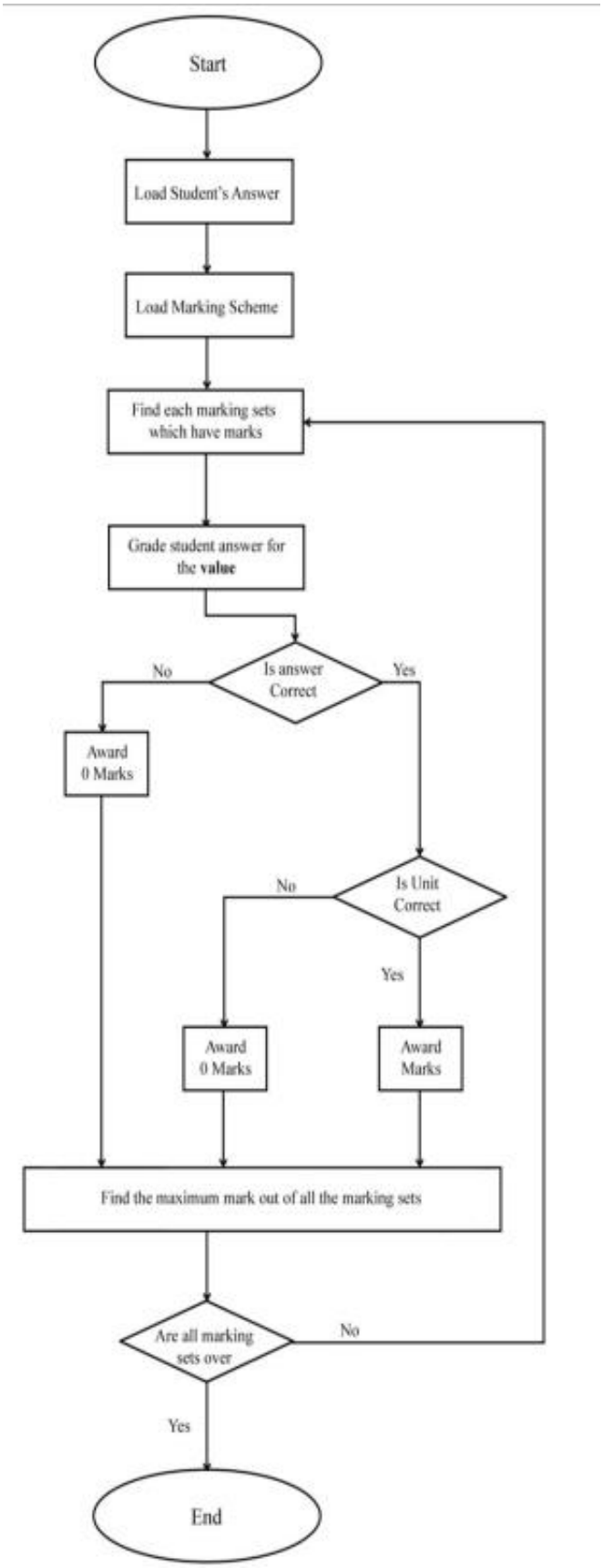


Figure 2.37 Flow chart of the automated mathematical questions grading system (Kadupitiya, Ranathunga and Dias, 2016)

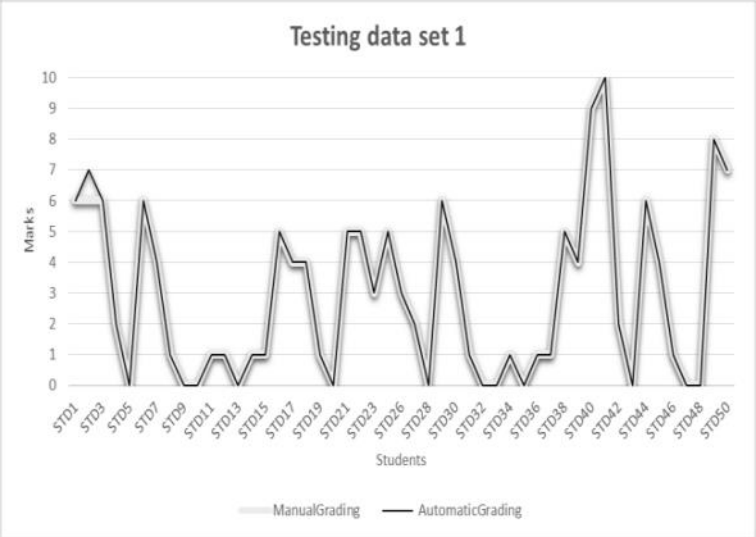


Figure 2.38 Grading technique used to evaluate question 01 (Kadupitiya, Ranathunga and Dias, 2016)

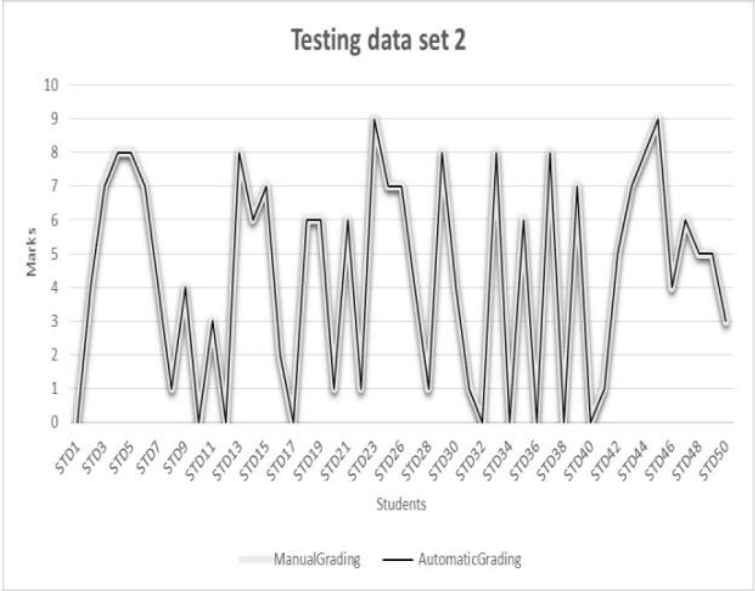


Figure 2.39 Grading technique used to evaluate question 02 (Kadupitiya, Ranathunga and Dias, 2016)

Based on two figures above, it can be seen that only 1 student is marked differently between manual and automated way. This means this designed automated system has a very high accuracy in grading the mathematical question automatically. As a conclusion, an automated system is having a higher accuracy as well as efficiency than the manual way. In this research, it can reduce manpower because one system works more than a teacher that can provide due to time and effort limitation.

In the next section, a tabulated summary will be presented on types of fields, techniques and programming languages used by every prior work.

2.3 Summary of prior works

Fields	Authors (numberings are based on Chapter 2.2)																								Total
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
Testing	✓	✓		✓	✓	✓		✓		✓			✓			✓	✓			✓				✓		12
Process			✓																							1
Papers Evaluation							✓		✓					✓					✓		✓	✓	✓		✓	8
Questions setup											✓															1
Medical												✓														1
Fingerprint															✓											1
Documentation																		✓								1
Types	**Techniques described**																								**Total**
Automated	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	24
Manual				✓		✓	✓			✓				✓	✓		✓	✓	✓	✓			✓		✓	12
Semi-automated																							✓			1
Types	**Programming Language usage**																								**Total**
Python	✓	✓				✓	✓		✓							✓	✓						✓			8
UNIX				✓																						1
Java										✓									✓							2
NLP											✓			✓							✓	✓				4
HistoQuest												✓														1
HTML																		✓								1
HyperWorks																				✓						1
Visual Basic																								✓		1
XML																									✓	1
Combination			✓					✓							✓											3
Others					✓								✓													2

Figure 2.40 Tabulated summary on types of fields, techniques and programming language used by every prior work

Legends:

NLP – Natural Programming Language

This section is to discuss the summary of all the prior works that are presented in the previous section 2.2 and Figure 2.40. According to all 25 researches, there are a total of 12 researches performing a comparison between automated technique (usually implemented in a system or a tool) and manual technique (usually manpower like human being) in their experiments. The rest of the researches, which a number of 13, directly utilise the automated way or semi-automated way over the manual way while conducting the experiment. From this result, this has proven that the automated method brings a lot of benefits while performing specific tasks. The advantages are shown below in point form:

i. Save more time and effort because the manual way is usually in the form of manpower which leads to tiredness after a long period of time while working, hence increasing the overall efficiency.
ii. Higher accuracy system due to the existence of the database system. Direct comparison is performed between the prepared ones and the submitted ones, therefore increasing overall effectiveness. For the manual method, he or she may accidentally miss out some small parts of the work.
iii. Higher flexibility because the automated method usually consists of a script and then it is implemented in a system or a tool (software). The written script is not unique to a software but just a few modifications are needed then the modified script can work well in other software. Since it can work in a number of software, thus the cost is lower comparatively.
iv. Higher reliability because the automated process performs consistently over a period of time. In the manual way, like discussed in the prior works, some teachers may be in a bad mood or have a special relationship with the student, the marks allocated to that student could be not tally as expected, leading to have a lower reliability.

In term of field, there is also a huge coverage because the automated system is not only applicable to factory usage such as test stage and processes but also for the study field due to its valuable benefits. In term of programming language, different researchers used different kind of them. This is due to its flexibility of the scripting. To wrap up, the automated technique does more benefits over the manual technique as aforementioned. Therefore, in this project, an automated flow named as Open Debug (OD) flow for the Periodic System Management Interrupt (PSMI) will be implemented in Python when a failure is captured by the OD will be presented in Chapter 3 (Methodology).

CHAPTER 3 METHODOLOGY

3.1 Overall project flow

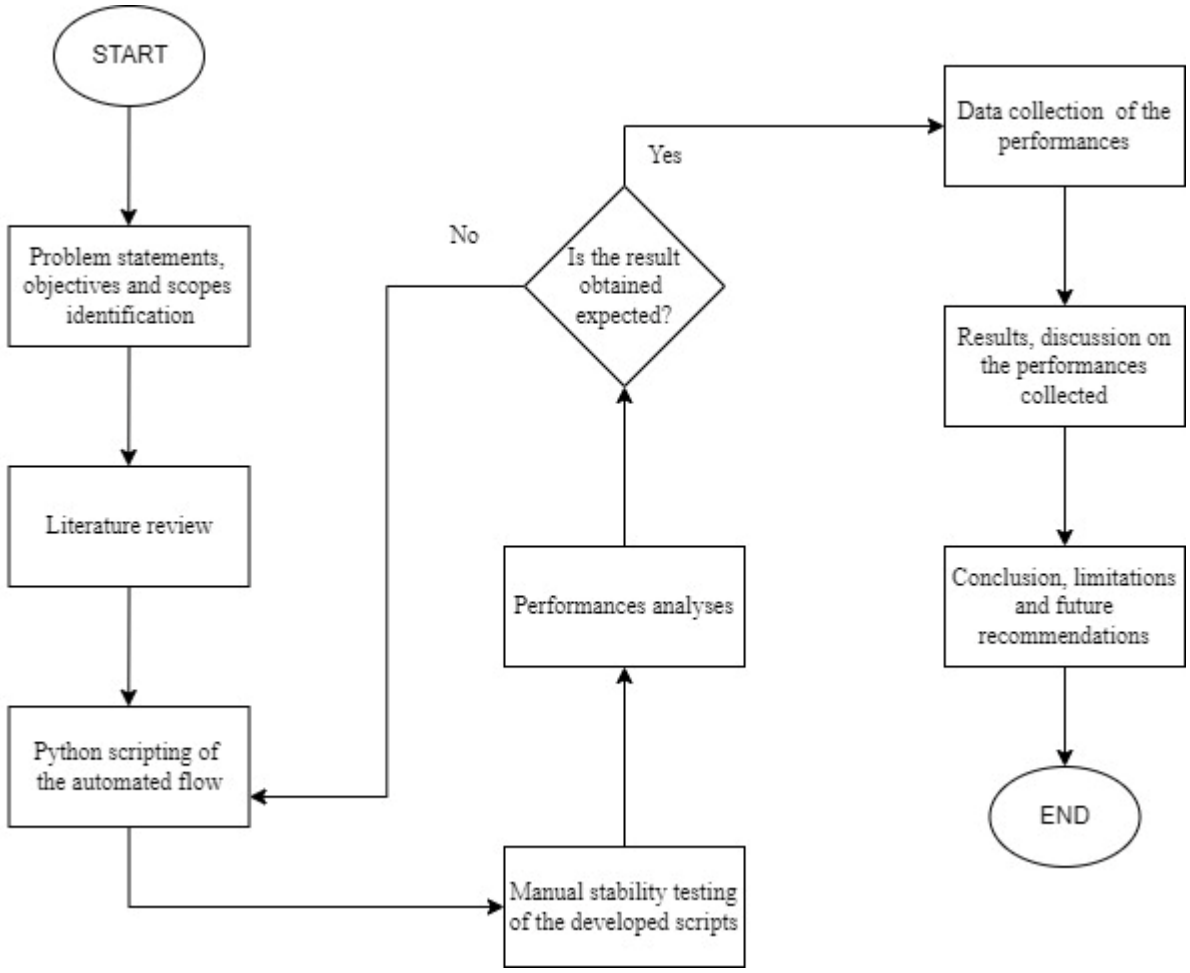


Figure 3.1 Flow chart of the overall project

Figure 3.1 shows the flow chart of the overall project flow. The problem statements, objectives and scopes have been discussed in Chapter 1 while the literature review has been presented in Chapter 2. After the literature review, the designing part starts here. Python scripting consists 2 parts: the PSMI capture flow and the failing signatures of failures captured. The steps of designing these scripts will be discussed at the later sections. After designing the script, a manual stability testing will be run in order to confirm the stability and analyse the performances (automated PSMI trace capture) with the data gathered. Next, results and discussion as well as the conclusion sections will be presented after the Chapter of Methodology. In the next section, the design of the automated flow of the PSMI capture in the OD flow will be presented.

3.2 Design of the automated flow of the PSMI capture in the OD flow

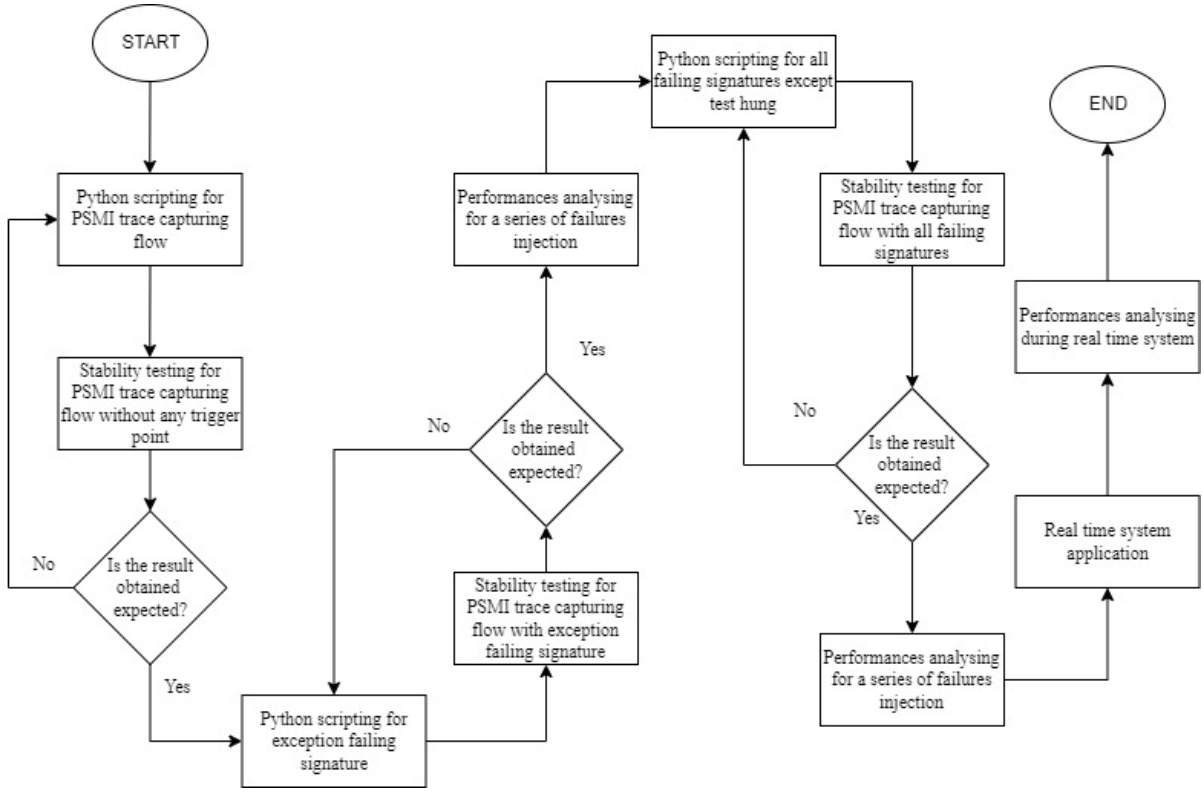


Figure 3.2 Flow chart of the automated flow of the PSMI capture in the OD flow

Figure 3.2 shows the design of the automated flow of the PSMI capture in the OD flow. For this proposed project, the scripts will be written in Python. Firstly, the overall PSMI trace capturing flow will be automated and designed using Python and is applied into OD flow. It is then followed by a manual testing without any failure injection in order to test for stability. Once it is working, the next step will be designing a script with a trigger point (a failing signature). This is then being tested for stability manually. After confirming the flow, a series of failures will be input to test for stability of the designed script. The failing signatures includes exception, memory mismatch, VM Exits Error and register mismatch will be developed into a script and will be tested manually as well. Data collection will be further gathered simultaneously.

The next section will discuss the design of the trigger point.

3.3 Design of trigger point

The communication between the failure and the PSMI is known as the trigger point. A trigger point is used to stop the PSMI at a particular point of the instruction given a particular PSMI interval (user-defined). There are many trigger points which can be used to capture a failure. However, not every trigger point suits every failure. Sometimes it can be used, but more effort is needed especially when performing trial-and-error methods, leading to time-consuming.

Therefore, in order to make use of the methods of capturing the PSMI, a universal method is introduced. This method is a memory trigger method, which is described as capturing a particular address, known as the error block address, with a specific data depending on the types of failing signatures. The failing signatures that are used to be tested are exception, memory mismatch, VM Exit Error and register mismatch. Those failing signatures are denoted by a specific value in hex form as shown in Table 3.1 below.

Table 3.1 Type of failing signature and its specific value notation

Failing signatures	Specific Value in hex form
Exception	0x08
VM Exit Error	0x09
Memory mismatch	0x02
Register mismatch	0x0A

For the error block address, it is always having an offset of 0x24 with a range of 0x100 bytes. The lowest byte 0x24 is denoted as the internal loop, which could be different for every run and every test. In other word, it is a randomised value. For the range from 0x25 to 0x28, which are 4-byte in size, it is a constant value once the failing signature is determined. Hence, the designed automated system can use this fixed and most significant hex value (offset 0x28) to capture the PSMI as shown in Table 3.1. As a summary, once the full error block address for the aforementioned failing signature is determined, the universal method can be used. The example is shown below:

- Address: 0x____28 (Error block address which excludes internal loop)
- Data: 0x08 or 0x8 (if the failing signature is an exception)

In the next section, the Python Scripting for the PSMI capture flow in the Open Debug (OD) flow will be presented.

3.4 Python Scripting for the PSMI capture flow in the OD flow

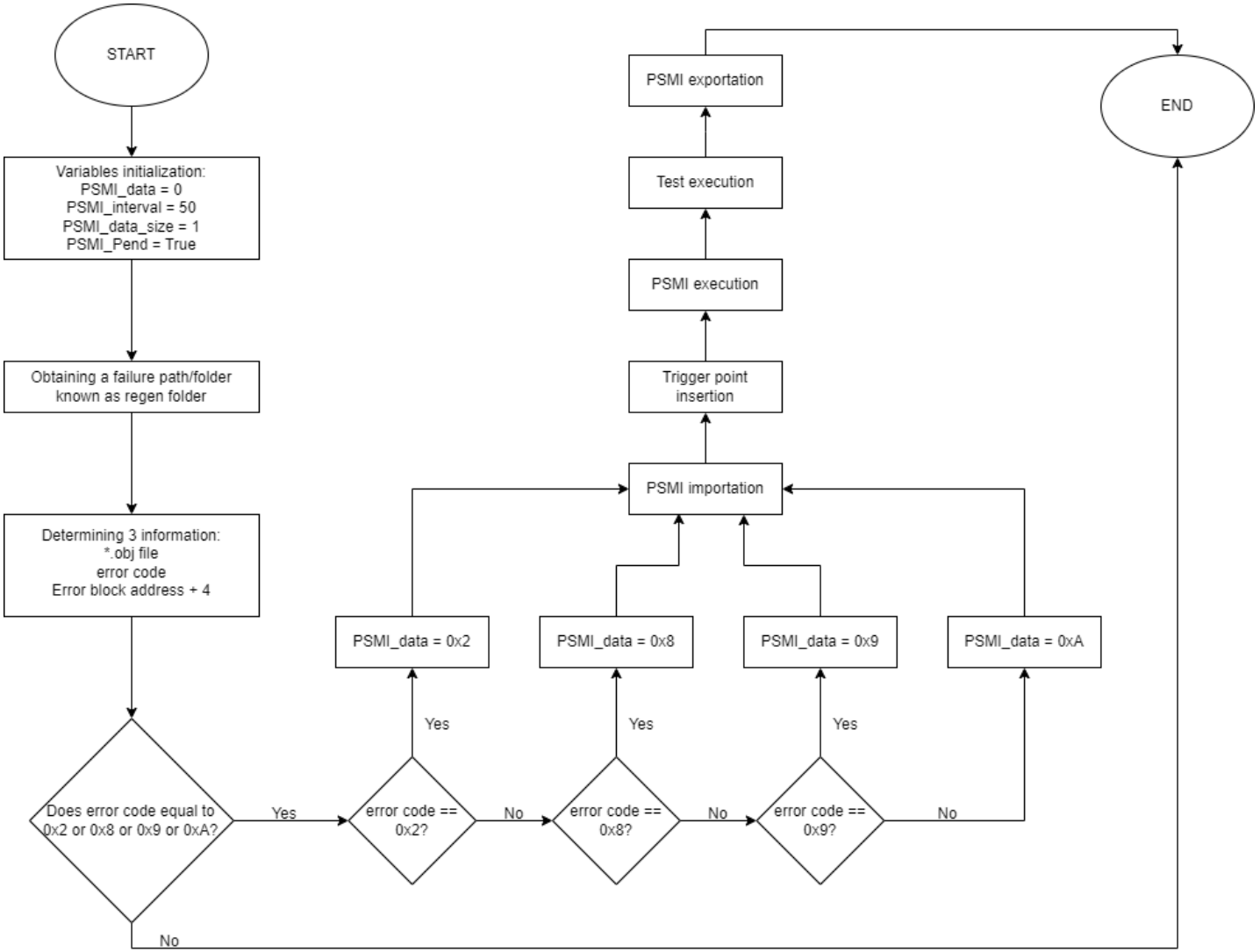


Figure 3.3 Flow chart of the Python Scripting for the PSMI capture flow

Figure 3.3 shows the flow chart of the Python Scripting for the PSMI capture in the Open Debug (OD) flow. The scripting method used here is Python only. Each of the step stated in the flow chart shown in Figure 3.3 will be discussed individually. In order to run this script, a normal regression is kicked started. Once a failure is detected, the flow chart in Figure 3.3 will be triggered.

1) Variables initialization

   Similar to most of the projects, some variables will be initialized including some libraries importation. Here, there are 4 variables being used:

   - PSMI_data which is used for trigger point usage during PSMI capture flow.
   - PSMI_interval, a user-defined time period, which is used for PSMI to capture a number of cycles of the trace.
   - PSMI_data_size which is used for trigger point usage during PSMI capture flow.
   - PSMI_Pend, a bool variable, which is also used for PSMI capture flow.

2) Obtaining the failure path named as regen folder

   Once a failure is detected, a regen folder will be regenerated automatically in the steps prior to PSMI capture flow in the OD flow. This is necessary because 3 important information have to be prepared and gathered before going to the next step. For this, the method here to obtain the path is to use the pre-defined libraries that are used before PSMI capture flow in the OD flow.

3) Determining *.obj file, error code and error block address

   - Files grepping

     In order to obtain these 3 information, 3 files in the regen folder are needed to be found. Hence, “glob” library is utilized here. For *.obj file, it is used to run the test during PSMI capture flow in the OD flow.

   - Information grepping from a file

     For error code and error block address, the data are found inside the specific files. Therefore, “re.search” technique is performed here and return the values obtained as variables. A note to be taken here is the error block address gathered will be added by ‘4’ as discussed in Chapter 3.3.

4) PSMI_data decision based on error code

   After collecting the information, the flow will go into a decision path to check whether the error code read is 0x2, 0x8, 0x9 or 0xA. If it is none of them, the PSMI capture flow in the OD flow will end directly due to undefined error code specified in the flow.

5) PSMI capture flow

There are 5 stages of PSMI capture flow All stages use the built-in Python commands. The first 3 stages are very simple and direct. The fourth stage is to perform the test execution while the last stage is to save the information in the common path. Each stage is discussed respectively in point form as shown below:

i) PSMI Importation

The first stage of the PSMI capture is known as the PSMI importation. This stage requires the user to import all the essential files needed during PSMI capture which comprise types of the project that is running, the communication between the project and the user and the fixes to the bug found previously. Once the user prompts the code, the system will enter a halted state, allowing the user to key in the trigger point which is needed in the second stage.

ii) Trigger point insertion

The second stage of the PSMI capture is the user has to command the trigger point which is to stop the PSMI at a particular point of the instruction given a user-defined particular PSMI interval. This stage has already been discussed in the previous section 3.3.

iii) PSMI execution

The third stage of the PSMI capture is the PSMI execution. This includes the user-defined PSMI interval to be commanded simultaneously. Once this is commanded, the PSMI starts running counting from zero and the system will go back to the operation mode. At this period, it is time to run the test.

iv) Test execution

The fourth stage of the PSMI capture is to run the test. The PSMI will stop the test whenever it detects the trigger point that is set previously in stage 2 or it is called as “trigger”. Once the PSMI capture flow gets a “trigger”, the test will enter halted mode again.

v) PSMI exportation

The last stage of the PSMI capture is to export the captured trace to a common path which is used for debug purpose. At this period, the PSMI capture process is completed successfully.

The full Python coding will be attached at the Appendix B/Full Python coding for the PSMI capture flow in the OD flow.

The next chapter, Chapter 4, will present the Results and Discussion.

CHAPTER 4 RESULTS AND DISCUSSION

In this chapter, the manual method time estimation for the PSMI (Periodic System Management Interrupt) capture process will be collected and discussed. This is to compare with the automated PSMI capture flow which has been implemented in the OD (Open Debug) flow. In the next section 4.1, the manual method time estimation for the PSMI capture will be discussed.

4.1 Manual method time estimation for PSMI capture and data collections

In this section, the survey for the manual time estimation for the PSMI capture process will be collected across 4 different projects. Those gathered information is collected from the colleagues whose have performed at least one manual PSMI capture. There is a total of 7 steps, starting from a time period when the debugger starts the initial process to understand the failure's behaviour until the completion of the PSMI capture, as mentioned in Chapter 1.4 (vi). Each step will be explained briefly with valid assumptions made in point form as shown below:

1) Understanding a failure.
   - Assumption: A fresh failure is assigned to a debugger.
2) PSMI trigger type selection.
   - Assumption: Only 4 kind of failing signatures which are Exception, VM Exit Error, Memory mismatch and register mismatch.
3) Select and lock a system.
   - Assumption: The selected system is undergoing operations and there is no ongoing process in the OD flow.
4) Reboot the selected system to initial configuration.
   - Assumption: The debugger is able to boot the system successfully without facing any environment errors.
5) Testing selected PSMI trigger type.
   - Assumption: The debugger is able to get a "trigger" after the testing stage.
6) Reboot the selected system once again.
   - Assumption: The debugger is able to boot the system successfully without facing any environment errors.
7) Starting the process of PSMI capture.
   - Assumption: Including all the steps mentioned in Chapter 3.4. The debugger is also able to complete the PSMI capture process successfully.

The survey for the manual time estimation for the PSMI capture process is summarised in Figure 4.1 with an average value computed based on every step per project in an ideal case. There are 2 assumptions being made. In order to make the calculations more easily, all the time estimated in every step will be standardised in MINUTES form. Next, the minimum time required for each step will be set as 1 minute. Every one column means capturing the PSMI trace for 1 failure only.

Across 4 different projects, it is observed that with the increasing number of cores, the time required to capture the PSMI trace manually is increased with the exception between Project A and Project B due to the maturity of the PSMI capture process. This is because when the number of cores increases, the system will need to operate more cores including understanding the failure (step 1) and rebooting the system (steps 4 and 6). Other than that, the time taken will be consumed more if there is any unpredictable environment issue occurring or the failure is having a special characteristic, making the PSMI trace fail to capture. Both steps 2 (PSMI trigger type selection) and 3 (Select and lock a system) may not consume much time since the debugger can direct decide what they need to use. For step 5 (testing process), this consumes lesser time as the importation of the PSMI is not needed here, leaving with only first 3 steps are needed (based on Chapter 3.4). Lastly, for step 7 (PSMI capture), as aforementioned after Project A, is almost consistent across the other 3 projects due to maturity issue.

Projects	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Total time (minutes)
Project A number of cores = x	20	1	1	5	15	5	20	67
	15	1	2	5	15	5	30	73
Average	**17.5**	**1.0**	**1.5**	**5.0**	**15.0**	**5.0**	**25.0**	**70.0**
Project B number of cores = 2x	30	1	1	10	5	10	10	67
	20	1	1	5	5	5	15	52
	20	1	1	8	10	8	20	68
	20	1	2	8	8	8	10	57
	20	1	2	7	10	7	15	62
Average	**22.0**	**1.0**	**1.4**	**7.6**	**7.6**	**7.6**	**14.0**	**61.2**
Project C number of cores = 3x	30	1	1	13	5	13	10	73
	25	1	2	12	10	12	15	77
	25	1	2	13	10	13	15	79
Average	**26.7**	**1.0**	**1.7**	**12.7**	**8.3**	**12.7**	**13.3**	**76.3**
Project D number of cores = 12x	40	1	1	23	5	23	5	98
	35	1	2	20	10	20	15	103
	35	1	1	22	5	22	10	96
Average	**36.7**	**1.0**	**1.3**	**21.7**	**6.7**	**21.7**	**10.0**	**99.0**

Figure 4.1 Manual time estimation for PSMI capture process across 4 different projects

In order to compare the manual method time estimation for the PSMI capture process with the time computation of the automated PSMI capture flow which is developed in the OD flow, the discussion on the automated method time computation will be presented in the next section.

4.2 Automated method time computation for PSMI capture and data collections

The automated time period for PSMI captured starts when the OD detects a failure and triggers the PSMI capture flow until the completion of the PSMI capture, as mentioned in Chapter 1.4 (v) as well as in Chapter 3.4. In order to allow the developed script to be run, it is required to add the written script into a folder named as "flows.ini" with some defined functional variables as shown in Table 4.1. This is a built-in folder to run any written script in the Open Debug (OD) flow.

Table 4.1 Defined functional variables in "flows.ini" for PSMI capture process

Defined functional variables	Data
flow	/<path to the python script/…/PSMI_final.py
timeout	1800
enabled	1

Based on Table 4.1, there are 4 defined functional variables being used for PSMI capture process. The variable "flow" is to allow the OD to run the stated python file, in this case is PSMI_final.py, when all the conditions are met. The variable "timeout" is to set a maximum time allocated for this flow in the OD, in this case it is 1800s (or 30 minutes). This is to make sure the OD will not hang forever in this flow and quit this flow after reaching the timeout value. This may happen due to environment issues or special tests. Lastly, the variable "enabled" is an on/'1' or off/'0' bit. If the user wants the OD to run the particular script, the variable "enabled" has to be set to '1', else this script will be ignored by the OD.

Before computing the automated method time required for the PSMI capture, there is one aspect to be taken into consideration, that is the test execution in the PSMI capture process. For every test in any project, there is a minimum time required for the test to execute known as "heartbeat". Here, it is set as 300,000ms (or 5 minutes). This is to avoid any false failure which requires a longer period of time to pass.

Projects	Analyses Time Collections (hour, minute, second)			Average Analysis Time (hour, minute, second)
Project A Number of cores = x	00:29:45	00:32:02	00:31:31	**00:31:06**
Project B Number of cores = 2x	00:13:15	00:12:19	00:16:56	**00:14:10**
Project D Number of cores = 12x	00:15:33	00:14:59	00:16:02	**00:15:31**

Figure 4.2 3 analyses time and average analysis time for projects A, C and D respectively

Projects	Analyses Time Collections (hour, minute, second)					
Project C Number of cores = 3x	00:13:03	00:13:25	00:15:12	00:15:42	00:15:11	00:14:12
	00:15:53	00:14:31	00:14:11	00:14:25	00:14:59	00:14:06
	00:15:48	00:13:34	00:14:52	00:13:05	00:13:35	00:14:09
	00:15:11	00:14:22	00:15:19	00:14:19	00:14:29	00:13:45
	00:14:01	00:14:25	00:14:27	00:13:40	00:14:32	00:14:51
	00:14:49	00:15:34	00:15:44	00:13:59	00:13:12	00:13:57
	00:15:25	00:14:59	00:13:40	00:15:51	00:14:53	00:13:07
	00:13:52	00:13:43	00:14:02	00:13:35	00:15:40	00:14:23
	00:14:32	00:14:05	-	-	-	-
Average Analysis Time (hour, minute, second)	**00:14:27**					

Figure 4.3 50 analyses time and average analysis time for project B

Figure 4.2 shows the 3 analyses time (hour, minute, second) and average analysis time for projects A, B and D respectively. The Python coding shown in Appendix B is not compatible in these 3 projects. According to Figure 4.2, only 3 analyses time are collected for projects A, B and D because there are limited systems for these 3 projects to test for the written script for the automated PSMI capture. For project A. this project is an old project and the current PSMI capture for this project is not mature. For projects B and D, both are on-going projects which are prioritized to achieve the goals. Hence, only 3 captures are done for projects A, B and D respectively during testing process. Figure 4.3 shows the 50 analyses time (hour, minute, second) and average analysis time for project B. The Python coding in Appendix B is basically for Project B only. Based on Figure 4.3, more data are captured in project B comparatively is because of the project systems person in-charge ownership. In the next section, the manual method and the automated method to capture PSMI will be compared and discussed in term of time.

4.3 Comparison between Manual method and automated method for PSMI capture in term of time

For a better view and comparison, the manual method time estimation for PSMI capture and the average analysis time of the automated method time computation for PSMI capture (or the Python Scripting for the PSMI capture in the Open Debug (OD) flow) for projects A, B, C and D will be tabulated in Table 4.2.

Table 4.2 Average manual method time estimation and average automated method time computation for PSMI capture process

Projects	Average manual (M) method time estimation (hour, minute, second)	Average automated (A) method time computation (hour, minute, second)	% Improvement = (|M-A|/M)*100% (%)
Project A	01:10:00	00:31:06	55.57%
Project B	01:01:12	00:14:10	76.85%
Project C	01:16:18	00:14:27	81.06%
Project D	01:39:00	00:15:31	84.33%

Table 4.2 shows the average manual method time estimation, average automated method time computation for PSMI capture process as well as the percentage improvement. The term "percentage improvement" here means how much time is saved when performing PSMI capture using automated method or Python scripting over manual method. Undeniably, based on Table 4.2, it is proven that using automated method to capture PSMI can save more than 55% to 84% time as compared to manual method. This is because automated method can help to eliminate the steps 1 to 6 as discussed in Chapter 4.1. With this, the debugger is able to save a lot of time to search for system to use as well as avoiding any environment issue. With the help of the PSMI process in the OD, once the failure is detected, the PSMI can be captured directly without needing the effort from debuggers to triage and proceed the following steps.

In the next Chapter, Conclusion, limitations and future recommendations will be presented.

CHAPTER 5 CONCLUSION, LIMITATIONS AND FUTURE RECOMMENDATIONS

In this chapter, 3 sections will be discussed in the sequence namely conclusion, limitations and future recommendations of the proposed and developed automated flow of the PSMI capture in the OD flow will be discussed. In the next section, the conclusion will be presented first.

5.1 Conclusion

As a conclusion, in this era of globalisation, the automated systems are widely used in various kind of fields such as engineering, learning, examination and more around the world. This trend has become an inevitable aspect in the life of human beings. Even though manual methods still exist, but nowadays, semi-automated or even fully-automated systems are gradually replacing the traditional methods known as the manual method or manpower due to its fast-process, high reliability, high effectiveness and high efficiency features.

As discussed in the Literature review, many researchers utilised automated tools or systems in their researches in the specific fields. The performances of the automated systems showed that they had many advantages over the manual techniques. This has also proven that the automated flow has been fully embedded in the root of the life.

In this project, in order to design the proposed automated system known as the automated PSMI flow in the Open Debug (OD), Python programming language is utilised. The failing signatures that are going to be tested in this developed flow are the exception, memory mismatch, VM Exit Errors and register mismatch. The failure of each of the failing signature is inserted in this flow to check for stability. The trigger point design basically is using a memory trigger, meaning the PSMI will be stopped (and this is known as a “trigger”) once it detects the pre-selected address, in this case is the error block with offset 0x28, and the pre-selected data. In this case, depending on the types of the error code (memory mismatch is 0x2, exception is 0x8, VM Exit error is 0x9 and register mismatch is 0xA). After understanding the design phase of the trigger point, the Python Scripting for the PSMI capture flow in the OD flow is designed on the next step, a total of 5 steps. First, it will be the common step known as variables initialization. After that, it is necessary to obtain the failure path named as regen folder. Inside the regen folder, by using Python coding, *.obj file, error code and error block address are required to be determined before undergoing PSMI capture flow. Next, it is the PSMI_data decision based on error code, basically this is the step comes from the design of the trigger

point. Last but not least, the final step is the PSMI capture flow which comprises 5 steps – PSMI importation, trigger point insertion, PSMI execution, test execution and PSMI exportation.

In this proposed technique, 2 methods are carried out separately namely manual method and automated method for PSMI capture. In order to compare both methods, the time consumed between them are taken into considerations. For manual method time estimation for PSMI capture, it is defined as the manual time period for PSMI captured starts when the debugger starts the initial process to understand the failure's behaviour until the completion of the PSMI capture. To collect data for this method, a survey is done from different debuggers that have completed the PSMI capture at least once in any of the 4 projects. Based on Figure 4.1 and Table 4.2, it is observed that at least one hour is needed to capture PSMI for any manual method in any project. For automated method time computation for PSMI capture, it is defined as the automated time period for PSMI captured starts when the OD starts the PSMI capture flow until the completion of the PSMI capture. In order to gather data for this method, few attempts haven been done in projects A, B and D (due to limited systems) while 50 attempts have been done in project C. According to Figure 4.2, Figure 4.2 and Table 4.2, it can be seen that at the maximum of half an hour is needed to complete the PSMI capture in automated technique. By comparing both approaches, undeniably, for about 55% to 84% time has been saved if utilising automated approach over manual approach. In other words, the automated technique consumes lesser time and efforts as compared to the manual technique in PSMI capture.

In the next 2 sections, the limitations and the future recommendations of the proposed and developed automated flow of the PSMI capture in the OD flow will be presented.

5.2 Limitations

For this proposed project, there are some limitations and they will be discussed respectively in point form as shown below.

1) Limited error codes detection and project specific

   As mentioned in previous chapters, for this automated PSMI capture in the OD flow, it will only detect 4 kinds of the failing signatures known as exception, memory mismatch, VM Exit Errors and register mismatch. Other failing signatures will be ignored. The command in Appendix B is only applicable for project B. Modifications are needed for other projects.
2) Capturing multiple failures with the same root-caused issue

   For this project, the Python scripting for the PSMI capture flow in the OD flow will be always triggered when a failure is detected. Since the failing signature could be different or the same, after being root-caused, they are the same, leading to waste time and effort.
3) Affect regression run rate

   For the Python scripting for the PSMI capture flow in the OD flow, when it is triggered, the regression will be forced to stopped because the system cannot run multiple tests simultaneously. Since in the PSMI capture flow, it is required to run the failure, indirectly affecting the regression run rate.

5.3 Future Recommendations

In order to solve the aforementioned limitation found in this proposed project, some recommendations are stated here.

1) Enhance the current script to be more flexible by understanding how failures are reported and allowing the script to be applied to all projects which support PSMI. The method suggested here for first limitation is to use port out 0xf8, a built-in code for most of the failing signatures.
2) Enhance the current script by connecting it with a new feature known as Quark signature which can help to filter the failure at the surface level. This can help to avoid capturing PSMI trace of the failures with the same root-caused issue.
3) Develop a batch file to include the current script such as PSMI.bat. This is to allow the debuggers to run the PSMI capture automatically without affecting the regression run rate.

REFERENCES

1. Intel Wiki – OD, *Open Debug* [online]. Available at: https://wiki.ith.intel.com/display/Hexa3/OpenDebug. [Accessed at: 13 October 2022]

2. Intel Wiki – PSMI, *Periodic System Management Interrupt* [online]. Available at: https://wiki.ith.intel.com/display/DebugEncyclopedia/PSMI. [Accessed at: 13 October 2022]

3. C. Jiang, B. Liu, Y. Yin and C. Liu, "Study on real-time test script in automated test equipment," *2009 8th International Conference on Reliability, Maintainability and Safety*, Chengdu, China, 2009, pp. 738-742, doi: 10.1109/ICRMS.2009.5270090.

4. V. Radhakrishna, V. SravanKiran and K. Ravikiran, "Automating ETL process with scripting technology," *2012 Nirma University International Conference on Engineering (NUiCONE)*, Ahmedabad, India, 2012, pp. 1-4, doi: 10.1109/NUICONE.2012.6493217.

5. A. Shirude, S. Totala, S. Nikhar, V. Attar and J. Ramanand, "Automated Question Generation tool for structured data," *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Kochi, India, 2015, pp. 1546-1551, doi: 10.1109/ICACCI.2015.7275833.

6. B. Kovacevic, M. Kovacevic, D. Stefanovic and M. Loncarevic, "Verification of Set-Top box media player functionality using automated test system," *2015 IEEE 1st International Workshop on Consumer Electronics (CE WS)*, Novi Sad, Serbia, 2015, pp. 68-71, doi: 10.1109/CEWS.2015.7867158.

7. 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.

8. J. Long and J. Zhou, "Automated mechanical simulation system for microelectronic packaging," *2015 16th International Conference on Electronic Packaging Technology (ICEPT)*, Changsha, China, 2015, pp. 51-55, doi: 10.1109/ICEPT.2015.7236543.

9. A. Kulkarni and Madhavi E, "Developing Reference Help documents automatically by using scripting methods," *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, Bengaluru, India, 2016, pp. 428-429, doi: 10.1109/CSITSS.2016.7779399.

10. D. Zun, T. Qi and L. Chen, "Research on automated testing framework for multi-platform mobile applications," *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Beijing, China, 2016, pp. 82-87, doi: 10.1109/CCIS.2016.7790229.

11. J. C. S. Kadupitiya, S. Ranathunga and G. Dias, "Automated assessment of multi-step answers for mathematical word problems," *2016 Sixteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, Negombo, Sri Lanka, 2016, pp. 66-71, doi: 10.1109/ICTER.2016.7829900.

12. Y. Stefinko, A. Piskozub and R. Banakh, "Manual and automated penetration testing. Benefits and drawbacks. Modern tendency," *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, Lviv, Ukraine, 2016, pp. 488-491, doi: 10.1109/TCSET.2016.7452095.

13. A. Kansal, R. Sinha and R. Jaiswal, "Fully automated regression tool for post silicon validation," *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Delhi, India, 2017, pp. 1-4, doi: 10.1109/ICCCNT.2017.8203924.

14. G. P. Roja and S. M. Sarala, "Automated testing of the medical device," *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, India, 2017, pp. 217-220, doi: 10.1109/RTEICT.2017.8256589.

15. S. Buyrukoglu, F. Batmaz and R. Lock, "A new marking technique in semi-automated assessment," *2017 12th International Conference on Computer Science and Education (ICCSE)*, Houston, TX, USA, 2017, pp. 545-550, doi: 10.1109/ICCSE.2017.8085551.

16. A. Dewanto et al., "Semi-Automated and Manual Methods for Counting Cells Expressing P75 Receptor in Endometriotic Lesions," *2018 4th International Conference on Science and Technology (ICST)*, Yogyakarta, Indonesia, 2018, pp. 1-6, doi: 10.1109/ICSTC.2018.8528285.

17. R. R. A. M. P. Jayawardena, G. A. D. Thiwanthi, P. S. Suriyaarachchi, K. I. Withana and C. Jayawardena, "Automated Exam Paper Marking System for Structured Questions and Block Diagrams," *2018 IEEE International Conference on Information and Automation*

for Sustainability (ICIAfS), Colombo, Sri Lanka, 2018, pp. 1-5, doi: 10.1109/ICIAFS.2018.8913351.

18. S. Tariq, "Automation of reflectarrays in HFSS using visual basic scripting," *2018 Texas Symposium on Wireless and Microwave Circuits and Systems (WMCS)*, Waco, TX, USA, 2018, pp. 1-4, doi: 10.1109/WMCaS.2018.8400640.

19. I. Dobles, A. Martínez and C. Quesada-López, "Comparing the effort and effectiveness of automated and manual tests," *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, Coimbra, Portugal, 2019, pp. 1-6, doi: 10.23919/CISTI.2019.8760848.

20. P. Dinesh Divsekar, R. Porob and P. S. Kuwelkar, "Automated Regression Testing and Data Analytics using Python," *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, Bangalore, India, 2019, pp. 95-99, doi: 10.1109/RTEICT46194.2019.9016774

21. R. Walker, "Pattern System Design: An Approach to Automating the Design of Automated Test Equipment," *2019 IEEE AUTOTESTCON*, National Harbor, MD, USA, 2019, pp. 1-4, doi: 10.1109/AUTOTESTCON43700.2019.8961047.

22. M. Agarwal, R. Kalia, V. Bahel and A. Thomas, "AutoEval: A NLP Approach for Automatic Test Evaluation System," *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)*, Kuala Lumpur, Malaysia, 2021, pp. 1-6, doi: 10.1109/GUCON50781.2021.9573769.

23. H. Vimalaraj et al., "Automated Programming Assignment Marking Tool," *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, Mumbai, India, 2022, pp. 1-8, doi: 10.1109/I2CT54291.2022.9824339.

24. K. S. Koushik, B. U. Sourav Chengappa and R. P. Chendan, "Automated Marks Entry Processing in Handwritten Answer Scripts using Character Recognition Techniques," *2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC)*, Coimbatore, India, 2022, pp. 728-733, doi: 10.1109/ICESC54411.2022.9885493.

25. N. Tabassum, M. S. Ahsan, I. Chowdhury and U. Basu, "IoT based Automated Examination Management System with Biometric Portal," *2022 International Conference*

on Innovations in Science, Engineering and Technology (ICISET), Chittagong, Bangladesh, 2022, pp. 52-55, doi: 10.1109/ICISET54810.2022.9775923.

26. R. A. Rizvee, M. F. Arefin, M. R. Khan, M. N. Islam and K. F. Rabbi, "An Automated System to Calculate Marks from Answer Scripts," *2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Vancouver, BC, Canada, 2022, pp. 0048-0054, doi: 10.1109/IEMCON56893.2022.9946626.

27. S. K. Sinha, S. Yadav and B. Verma, "NLP-based Automatic Answer Evaluation," *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, 2022, pp. 807-811, doi: 10.1109/ICCMC53470.2022.9754052.

APPENDICES

Appendix A Work Schedule

Gantt Chart of Research Project		First Trimester (Weeks)												Second Trimester (Weeks)											
No.	Aspects	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Preparation of Thesis																									
1	Introduction																								
2	Literature Review																								
3	Research Methodology																								
4	Design Methodology																								
5	Results and Discussions																								
6	Conclusion																								
Design Flow																									
7	Python scripting for PSMI trace capturing flow																								
8	Stability testing for PSMI trace capturing flow without any trigger point																								
9	Python scripting for exception failing signature																								
10	Stability testing for PSMI trace capturing flow with a trigger point on exception failing signature																								
11	Performances analysing for a series of failures injection																								
12	Python scripting for all failing signatures except test hung (universal method)																								
13	Stability testing for PSMI trace capturing flow with a trigger point for all failing signatures except test hung																								
14	Performances analysing for a series of failures injection																								
15	Apply into real time system																								
16	Performances analysing during real time system																								

Appendix B Full Python coding for the PSMI capture flow in the OD flow

```python

import os
import os.path
import re
import sys
import glob
from evtar.lotus.open_debug.base.framework.core import action


#*****************************************************************************************************************************************************
# Initialisation
#*****************************************************************************************************************************************************

# Initialise variables
#----------------------

PSMI_data = 0
PSMI_interval = 50 # user-defined
PSMI_data_size = 1
PSMI_Pend = True

# To get a failure path/folder
#------------------------------
failure_path = action.test.get_regenerate_folder()


# To get *.obj file for test execution
#--------------------------------------
obj_file = 'VER_*.obj'
obj_path = glob.glob(failure_path + "\\*" + obj_file)


# To read error code
#--------------------
parsed_error_file = '_ParsedErrorInfoLog.txt'
parsed_error_path = glob.glob(failure_path + "\\*" + parsed_error_file)

def get_error_code(parsed_error_path):

    if os.path.exists(parsed_error_path):
        with open(failure_path, 'r') as error_code_file:
            for line in error_code_file:
                if 'ERROR_CODE' in line:  # ERROR_CODE                       : 0x09
                    result = re.search(':\s*(?P<error>[0-9a-fA-Fx]+)', line)
                    break

    error_code_subf = int(result.group('error'), 16)
    return error_code_subf
```

```python

# To read error block address
#-----------------------------
cfg_file = 'VER_*.cfg'
cfg_path = glob.glob(failure_path + "\\*" + cfg_file)

def get_error_block_address(cfg_path):

    if os.path.exists(cfg_path):
        with open(failure_path, 'r') as error_block_address_file:
            for line in error_block_address_file:
                if 'StartAddress3' in line:  # StartAddress3 = 0x74024
                    result = re.search('=\s*(?P<address>[0-9a-fA-Fx]+)', line)
                    break

    error_block_address_subf = int(result.group('address'), 16)
    return hex(error_block_address_subf)
```

```
#*****************************************************************************************************************************************************
# Main function
#*****************************************************************************************************************************************************

def run_flow():
    action.log("Entered PSMI:run_flow()")
    error_code_main = get_error_code() # decimal
    error_block_address_main = get_error_block_address() + 4 # hexadecimal and address+4

    if (error_code_main == 2 or error_code_main == 8 or error_code_main == 9 or error_code_main == 10):

        if (error_code_main == 2): # memory mismatch
            action.log("Proceeding to capture PSMI on Memory Mismatch.")
            PSMI_data = 2
            return hex(PSMI_data)

        if (error_code_main == 8): # exception error
            action.log("Proceeding to capture PSMI on Exception Error.")
            PSMI_data = 8
            return hex(PSMI_data)

        if (error_code_main == 9): # VM Exit error
            action.log("Proceeding to capture PSMI on VM Exit Error.")
            PSMI_data = 9
            return hex(PSMI_data)

        if (error_code_main == 10): # register mismatch or 0x0A
            action.log("Proceeding to capture PSMI on Register Mismatch.")
            PSMI_data = 10
            return hex(PSMI_data)

        #*************************************************************************************************************************************************
        # PSMI importation
        #*************************************************************************************************************************************************

        sys.path.append(r"C:\pythonsv\grandridge") # By project
        from psmi.atom_psmi_for_grr import psmi # run in pythonSV

        #*************************************************************************************************************************************************
        # Trigger point insertion
        #*************************************************************************************************************************************************

        psmi.cmt_domain.trigger.on_memory_address(addr=error_block_address_main, data=PSMI_data, size=PSMI_data_size)
```

```
        #*************************************************************************************************************************************************
        # PSMI execution
        #*************************************************************************************************************************************************

        psmi.go(PSMI_interval, pend=PSMI_Pend)

        #*************************************************************************************************************************************************
        # Print messages
        #*************************************************************************************************************************************************

        message_error_code = f"The error code of this failure is 0x{hex(error_code_main)}."
        action.log(message_error_code)

        message_error_block_address = f"The error block address of this failure is 0x{error_block_address_main}."
        action.log(message_error_block_address)

        message_PSMI_memory_trigger = f"The command of the memory trigger is: psmi.cmt_domain.trigger.on_memory_address(addr=0x{error_block_address_main}, data=0x{PSMI_data}, size={PSMI_data_size})."
        action.log(message_PSMI_memory_trigger)

        message_PSMI_execution = f"PSMI is executed with PSMI interval of {PSMI_interval}ms."
        action.log(message_PSMI_execution)

        message_test_execution = f"Failure is running now, waiting for the trigger."
        action.log(message_test_execution)

        #*************************************************************************************************************************************************
        # Test execution
        #*************************************************************************************************************************************************

        from pysvtools.psmi.common.memory import xmonApi
        sys.path.append(r"C:\Program Files\Intel\XMon\scripts\remote")
        from testrunner_client import TestrunnerClient
        tc = TestrunnerClient()
        x1 = xmonApi(tc)
        x1._exec_xmon_cmd("run -init fake -cli -skip_mem_validation -no-verify {obj_path}")

        # save the info in a temporary folder
        psmi.export()

    # PSMI capture is skipped for other failing signatures
    else:
        action.log("Skipping PSMI capture due to other failing signatures.")

# Call the main function
if __name__ == "__run_flow__":
    run_flow()
```