JET TURBINE ENGINES:

LEGACY SOFTWARE MIGRATION

Myeza Sifiso

(201120715)

Bachelor of Science in Computer Science and Physics

Submitted in fulfillment of the requirements for the degree of

Honours in Computer Science

Department of Computer Science

Faculty of Science and Agriculture

University of Zululand

November 2017

Keywords

Comprehensive Migration Model - Jet Turbine Engines - Legacy Software Migration

Abstract

Programming languages have evolved rapidly. Developers are adopting the modern programming languages. Existing software or systems created using older programming languages are instantly obsolete and no longer serviceable. There are discrepant with the modern machines. Finding professional experts for the old programming languages, i.e., Pascal, FORTRAN, and Assembly Language is hard. Consequently, it has become challenging and expensive to maintain, update and upgrade legacy software. The South Africa Department of Defence developed a HoleFlow Jet Turbine Engine software in the 2000s. The HoleFlow software is a vital program for controlling the temperature in the Jet Turbine Engines. However, the Jet Turbine Engine HoleFlow software was written in the old version of Pascal programming language compiled to execute on old machines. Since this software operation is crucial, there is a need for maintaining and re-engineering it to support most modern engines. Therefore, the Jet Turbine Engine HoleFlow software need to be migrated and modernized. This research employs the existing migration approaches to propose a comprehensive migration model for migrating the Jet Turbine Engine HoleFlow software from Pascal to C++. This model employs the use of auto translational tool termed, Pascal-to-C/C++ translator that auto-translates Pascal to C++. Besides the evident benefits of having the program written in a current programming language. The proposed model succeeded in accomplishing the migration task of the Jet Turbine Engine HoleFlow Software written using old Pascal programming language to the targeted programming language, C++.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

OLSMM	Offline Legacy Software Migration Model
CT	Cold Turkey
CLS	Chicken Little Strategy
BM	Butterfly Methodology
CPU	Central Processing Unit

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature: _____

Date: _____

Acknowledgements

This research work would not have succeeded without the guidance and the contribution of specific valuable people.

Primarily I would like to offer the sincerest gratitude to the research supervisor Mr. P. Tarwireyi who has supported throughout the study period regardless of the type of the research I was conducting with patience and knowledge. Furthermore, the gratitude extended to Mr. J. Mhlongo for mentoring throughout this study without any complaints until the study completion.

Besides that, thanks to all the head members from the cluster, that includes Dr. P. Mudal, Mr. P. Tawaireyi, Mr. O. Kayode, and Mrs. N. Sibeko with their inputs and advice regarding the research work. Also, most profound gratitude extended to Mr. O. Kayode and Dr. P. Mudal for assisting in completing the research work in the absence of the primary supervisor.

Finally, thanks to everyone who responded to the survey used in the research for their input. They have shared a valuable insight into this study.

Chapter 1: Introduction

1.1 BACKGROUND

The Jet Turbine Engines are engines use to fly the aircraft at a minimal cost. They operate by sucking in air through the nominal head of the engine using turbofan. Inside the engine, exist a compressor that squeezes the sucked air and increases its pressure. The squeezed air mixes with fuel, and an electric spark burns the mixture at a high temperature inside the combustor. The oxidizing gas explodes through the nozzle located at the back of the turbine engine. Through the set-off, a burning gas that forcefully shoots backward of the turbine engine creates the thrust that causes the jet to move forward forcefully. Figures 1 and 2 shows the sample of Jet Turbine Engine and its components part respectively.

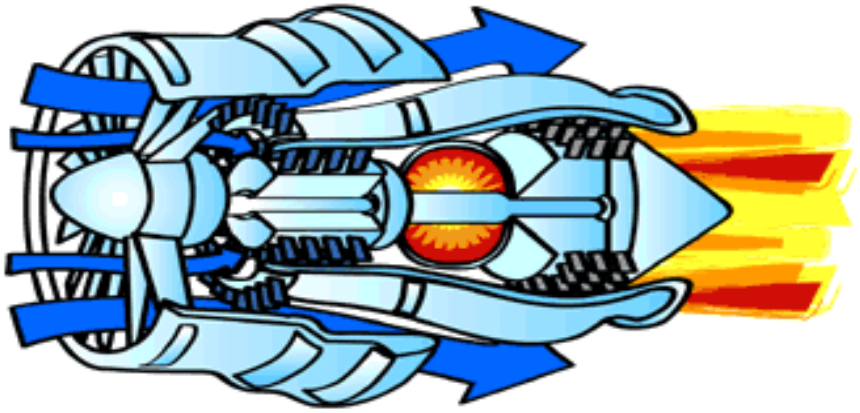

Figure 1: Turbine engine (Anon., 2017)

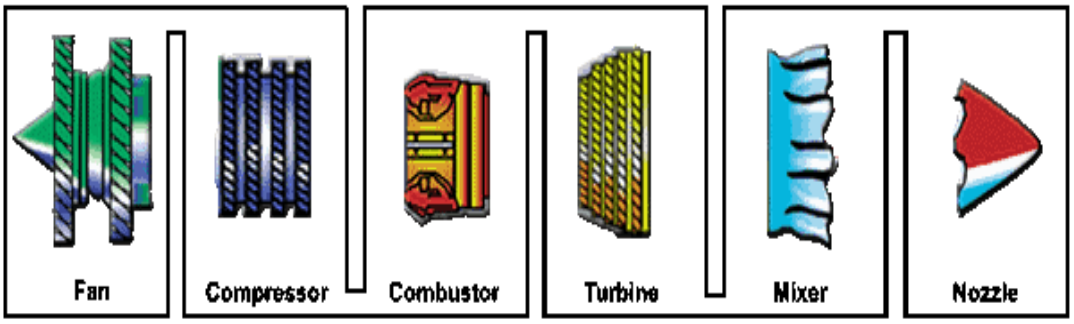

Figure 2: Parts of the turbine engine (Anon., 2017)

The jet turbine engines need to operate at specific maintained temperatures. As a result, a mission-critical computer program, termed the HoleFlow program, was developed in the 2000s by the South African Department of Defence to control the temperature inside the turbine engine combustor. This HoleFlow program was developed using the old Pascal programming language, which is now obsolete and no longer serviceable. Hence, this program needs to be migrated to be compatible with the current technologies. There are already existing migration approaches to migrate the legacy software to be modernized. However, these approaches are explanatory at a very high level with no specifications of the tools required to complete the migration task. There are no specific explanations on how to conduct each task. Therefore, this research proposes a comprehensive migration model that may be followed to migrate a Pascal program to the modernized C++ program.

When the Jet Turbine Engine HoleFlow Pascal program is migrated successfully to the modern programming language, C++, it will provide the benefits of secure compilation and execution on the modern machines. That will reduce the constraints associated with the flexibility of its design and adaptability. It will be easier to maintain, update and upgrade since there are more professionals available for the modern C++ programming language. In summation, this will reduce the cost of the HoleFlow program maintenance (Oest, 2008). Other companies or organizations that still operate legacy software written in Pascal programming language can quickly adopt and follow the procedure outlined by the proposed migration model in this research on how to migrate the Pascal program to C++.

1.2 DEFINITION OF TERMS

- **The Thrust** is the forwarding force, which pushes the jet engine and transitivity forward.
- **The fan** is the first component in the turbofan that sucks large quantity of air, and its blades usually are made of titanium. It functions by speeding up the air then split it into two parts whereby one part goes through the core of the engine (hotter air) and the other bypass the core of the engine (colder air).
- **The compressor** is the first component in the turbine engine made by fans with many blades attached to a shaft. Its functions to compress air that enters through smaller areas, which increase the air pressure. This squashed air enters the combustion chamber.
- **Combustor** this is where the air mixes with the fuel and then ignited. That causes burning air to a very high temperature approximately 3000 degrees Celsius.
- **Turbine** is where the high-energy airflow is coming out causing the turbine blades to spin which transitivity causes the blades in the compressor to spin since they are connected to the same shaft.
- **The nozzle** is the actual part that produces the thrust of the jet move.

1.3 PROBLEM STATEMENT

Technology has improved drastically, and programming languages have evolved rapidly. The companies or organizations are using modern programming languages, and they are operating on the modern machines. However, the Jet Turbine Engine HoleFlow Pascal program is a legacy software written in old Pascal programming language, which is incompatible with the modern machines. Therefore, it is imperative to migrate the HoleFlow Pascal program to the modern programming language, which is compatible with the modern machines. However, there is no comprehensive approach for migrating the legacy software. Therefore, this research envisages proposing a comprehensive migration model that migrates the software written in Pascal programming language to C++.

1.4 PURPOSES

1.4.1 Research Questions

This research addresses the following questions.

- What are the shortcomings of the existing migration models?
- How does the proposed comprehensive migration model overcome the shortcomings of the existing migration models?
- What is the effectiveness of the proposed migration model against the existing model?

1.4.2 Research Goal

The goal of this research is to propose the comprehensive migration model to migrate the software written in the Pascal programming language to C++.

1.4.3 Research Objectives

- To conduct a literature survey on the legacy software migration models.
- To propose a comprehensive legacy software migration model
- To evaluate the effectiveness of the proposed migration model and to compare it with the existing migration models.

1.5 RESEARCH SCOPE AND LIMITATIONS

The Jet Turbine Engine HoleFlow Pascal program was developed using a Windows operating system type that is a Microsoft Disk Operating System. It was compiled on that operating system and produced an application that is an executable file. Hence, the Jet Turbine Engine HoleFlow program is limited to only executing on the Windows operating systems. Furthermore, the Windows operating systems come in different system types, which include: 16-bit, 32-bit, and 64-bit operating systems. The Jet Turbine Engine HoleFlow Pascal program can only execute on the 32- or 16-bit operating system.

This lead to the limitation of the effectiveness and resource utilization evaluation of the Jet Turbine Engine HoleFlow program for the Pascal and C++ versions to be done on a 32-bit windows operating system for consistent comparison.

Again, the survey that was conducted based on the proposed comprehensive migration model only focused on the available people who are doing the similar work of migrating the legacy software to be modernized. The number of these people was limited to only three persons. That limited the number of people to participate in answering the survey questions.

1.6 MINI-DISSERTATION OUTLINE

The organization of the remaining chapters of the mini-dissertation is as follows:

- Chapter Two describes the literature review about the challenges and the needs for the legacy software migration including the various legacy software migration approaches against the proposed migration model.
- Chapter Three describes the methodology followed to accomplish the migration task of the legacy software.
- Chapter Four describes the qualitative evaluation results obtained from the literature and their analysis in the usage of the already existing approaches against the proposed legacy migration model. It also includes the survey questionnaire responses which was conducted to obtain a sight from different people about the proposed comprehensive migration model.
- Chapter Five provides the conclusions and future direction of the research.

Chapter 2: Literature Review

There has been a proposition of the different legacy software migration approaches and methodologies in the literature. (Oest, 2008) Reported that migrating legacy software can be a costly, time-consuming and a risky procedure that requires code changes, retesting, and even rectifying. Several factors make legacy software challenging to migrate, including compiler implementations, programming language nuances, runtime and hardware dependencies, and incompatible software code structure. (Bisbal, et al., 1997) Reported that legacy systems were developed using programming languages that are instantly obsolete and no longer serviceable. Therefore, they are challenging to extend. These legacy systems have grown hard and expensive to maintain, update and upgrade. That has forced the migration of software to newer programming languages, development information processing systems, compilers and operating systems. (Bisbal, et al., 1997) also exposed that there exists a gap in research about the comprehensive migration model of legacy software using migration approach guidelines.

Those approaches include Big Bang approach, also known as the Cold Turkey by (Brodie & Sronebraker, 1995), which is a re-developing of a legacy system from scratch using modern tools, architecture and running on new hardware configurations. Apart from a re-developing approach, others have used wrapping approach (Graham, 1995), which involves surrounding existing software to give a legacy system a newer and improved look or operations. The most famous example of wrapping is Screen Scraping that replaces character based front ends of a legacy system with a client based graphical user interface (Merlo, et al., 1995). The most efficient way of modernizing the legacy system is by migrating. It is evident in the study by (Ruhl & Gunn, 1991) that manual intervention for the migration was cumbersome, time-consuming and not complete, this led to the extension of the migration to estimated 24.7 staff months. On the other hand, the similar program was migrated using an automated approach that they proposed in the re-engineering methodology and only required ten staff months to complete. However, this approach was procedural therefore lacking agility. The three great approaches used to migrate

legacy codes are Col Turkey (Brodie & Sronebraker, 1995), Chicken Little Strategy (Brodie & Sronebraker, 1995) and a Butterfly Methodology (Wu, et al., 1997).

2.1 COLD TURKEY (BRODIE & SRONEBRAKER, 1995)

CT involves rewriting the legacy software from scratch to provide the target software using the current software techniques and hardware of the target environment. The use of this migration approach is too risky, expensive, time-consuming, prone to errors and even impossible to succeed if the complexity of the software is too high. Furthermore, technologies and business requirements are always changing, thus, at the end after a long migrating process; an organization might find itself with a redeveloped system based on obsolete technology that no longer meets its business needs.

2.2 CHICKEN LITTLE STRATEGY (BRODIE & SRONEBRAKER, 1995)

The CLS involves migrating the legacy software in place, by small incremental steps until the desired long-term objective achieved. This strategy lets the legacy software interoperate during migration using a mediating module known as the gateway, which is “a software module introduced between operation software components to mediate between them.”

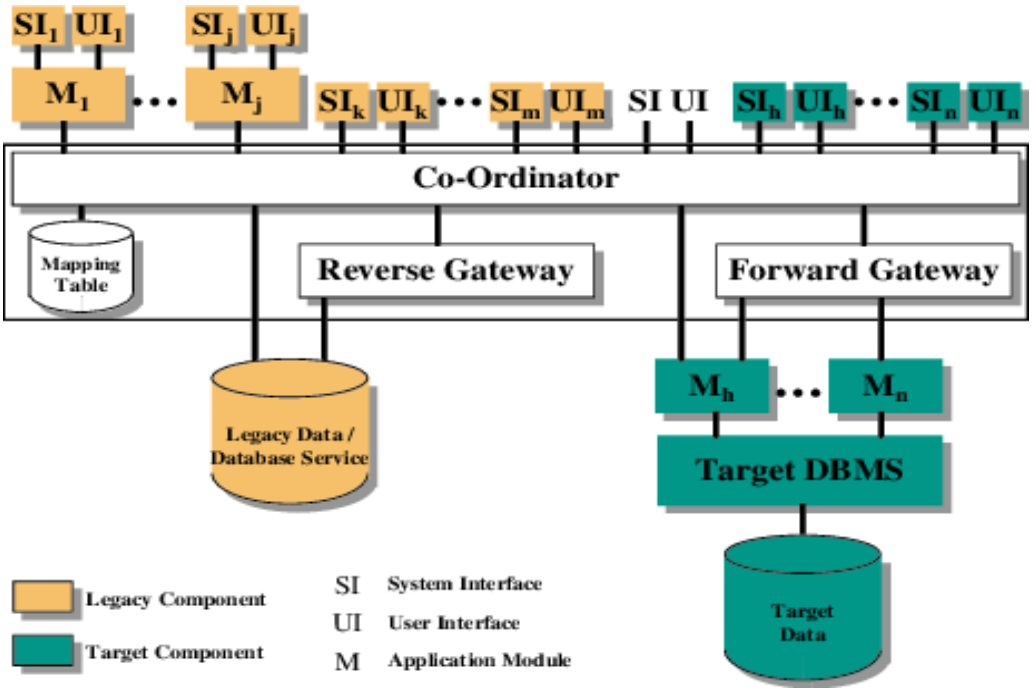

Figure 3: Chicken Little's general migration architecture (Brodie & Sronebraker, 1995)

Figure 3: represents the Chicken Little's general migration architecture. In this figure, there is a presence of forwarding and reverse gateways. These gateways, forward and reverse gateways, are used to enable the legacy applications to access the database environment on the target side of the migration process and enable target applications to access the legacy data management environment respectively. A gateway coordinator is used to maintain the data consistency. This coordinator introduces a complex technical problem since the maintenance of the consistency across the heterogeneous systems is difficult. The essential for the legacy and target systems to interoperate during the migration process via the gateways introduced also adds significantly to the complexity of an already complicated process of migration (Bisbal, et al., 1997).

CLS includes 11 steps process for cutting over from the legacy software to the target software. Each step is incremental:

1. Incrementally analyze the legacy software.
2. Incrementally decompose the legacy software structure.
3. Incrementally design the target interface.

4. Incrementally design the target application.
5. Incrementally design the target database.
6. Incrementally install the target environment.
7. Incrementally create and install necessary gateways.
8. Incrementally migrate the legacy database.
9. Incrementally migrate the legacy application.
10. Incrementally migrate the legacy interfaces.
11. Incrementally cutover to the target environment system.

2.3 BUTTERFLY METHODOLOGY (WU, ET AL., 1997)

The BM assumes that while the legacy system must stay operable throughout migration, it is not essential for the legacy and target systems to interoperate throughout the migration process. This assumption leads to the elimination of gateways, avoiding the enormous complications they involve during the migration process.

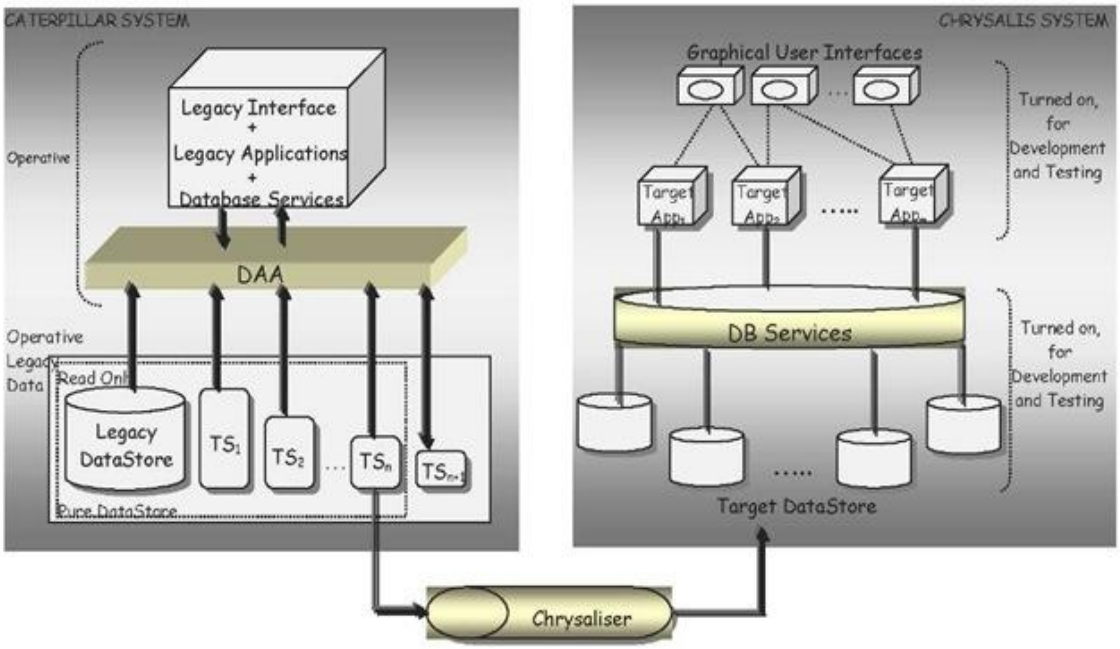

Figure 4: Butterfly Methodology's general migration architecture (Wu, et al., 1997)

Figure 4: shows the Butterfly Methodology's general migration architecture, which eliminates the gateways and gateway coordinator. This methodology reduces the complexity of the migration process by not interoperating through the migration process. It also avoids the challenges of the heterogeneity when migrating the legacy system.

However, the phases that are involved in this model are not explanatory in details. The phases are only outlining what is supposed to be done. Nevertheless, it does not outline how the phases are carried out. That forces the developers performing the migration task to figure out how to perform each phase by themselves. That increase the completion time of the migration task. Furthermore, this methodology also resembles the waterfall model as the CLS does. That causes this model to lack the agility of performing the migration task.

The Butterfly Methodology migration is processed in the following six phases:

1. Prepare for migration.
2. Understand the semantics of the legacy system and develop the target data schema(s).
3. Build up a Sample Datastore, based upon the Target Sample Data, in the target system.
4. Migrate all the components (except for data) of the legacy system to the target architecture.
5. Gradually migrate the legacy data into the target system and train users in the target system.
6. Cutover to the completed target system.

2.4 OFFLINE LEGACY SOFTWARE MIGRATION MODEL

All these researchers they have adopted different approaches to migrate legacy systems. (Brodie & Sronebraker, 1995), (Graham, 1995), (Merlo, et al., 1995), (Wu, et al., 1997) approaches assume that the legacy system operational environment already known, which is not always right. There are also explanatory at a very high level with no details of the specifics of how the migration process steps are carried out to produce the target software or system. There are missing the critical step to

analyze the targeted software against the legacy software to give the company or organization an insight into the performance gains or losses if there are any.

Building from the approximations of the mentioned researchers. The OLSMM migrates the legacy software by adopting the ideas of the already existing migration approaches. However, the OLSMM is Design Science Methodology based since it migrates only the legacy software written in Pascal programming language to C++.

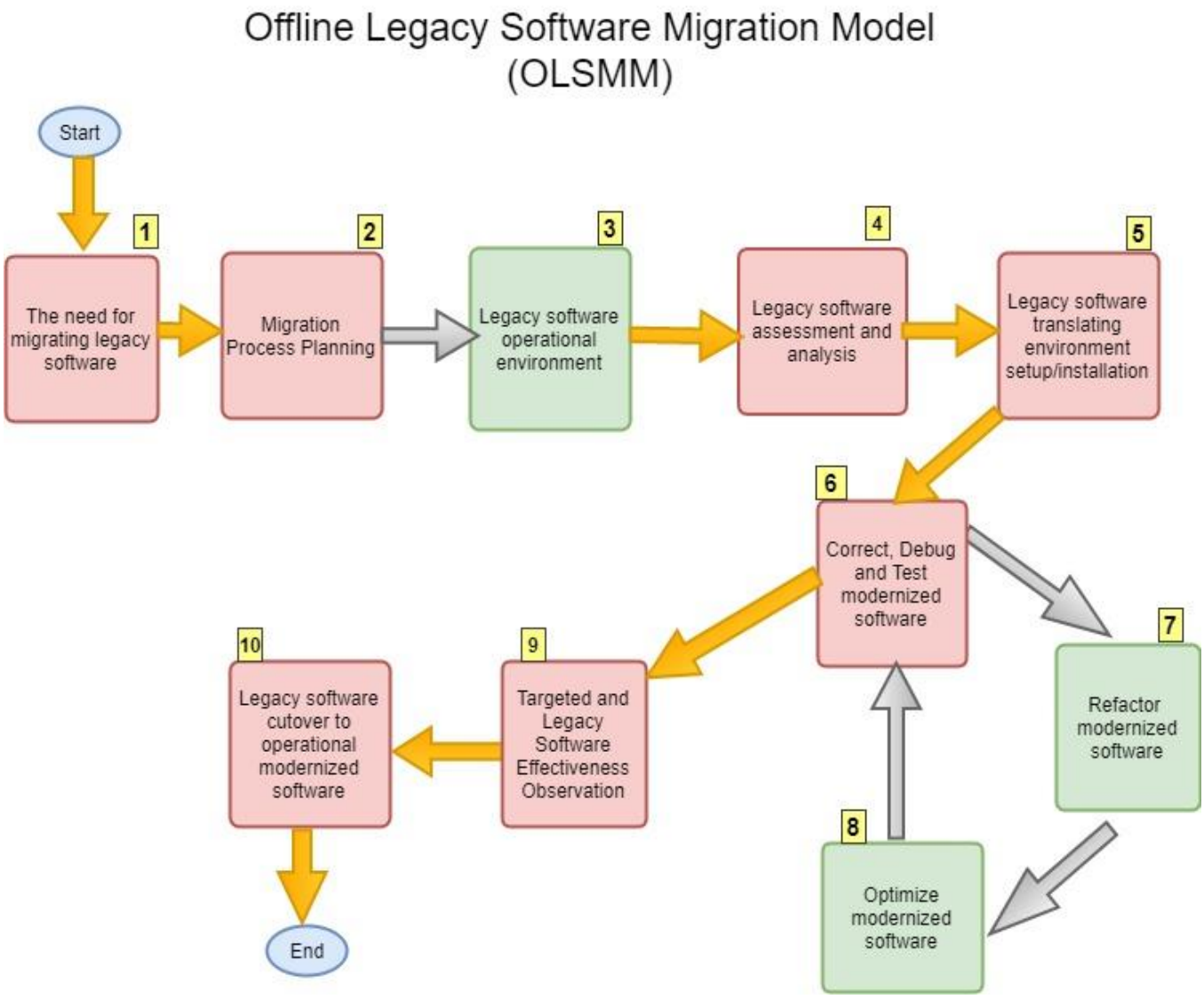

Figure 5: Offline Legacy Software Migration Model's flowchart

The OLSMM model consists of the following steps as they appear in figure number

1. The need for migrating legacy software.

2. Migration Process Planning.
3. Legacy software operational environment.
4. Legacy software assessment and analysis.
5. Install and Set up the target environment.
6. Correct, Debug and Test modernized software.
7. Refactor modernized software.
8. Optimize modernized software.
9. Targeted and legacy software effectiveness observation.
10. The legacy software cutover to modernized operational software.

2.5 SUMMARY AND IMPLICATIONS

Table 1: Comparison of the existing migration approaches against the proposed comprehensive migration model termed, Offline Legacy Software Migration Model

Steps	**CLS** (Brodie & Sronebraker, 1995)	**BM** (Wu, et al., 1997)	**OLSMM**
1	Analyse the legacy software.	Prepare for migration.	The need for Migrating Legacy Software.
2	Decompose the legacy software structure.	Understand the semantics of the legacy.	Migration Process Planning.
3	Design the target interface.	Build up a Sample Database, based upon the Target Sample Data, in the target	Legacy software operational environment.
4	Design the target application.	Gradually migrate all the components (except for data) of the legacy system to the target architecture.	Legacy software assessment and analysis.
5	Design the target database.	Incrementally migrate the legacy data into the target system and train users in the target system.	Install and or Set up the target environment.
6	Install the target environment.	Cutover to the completed target system.	Correct, Debug and Test modernized software.
7	Create and install		Refactor modernized

	necessary gateways.		software.
8	Migrate the legacy database.		Optimize modernized software.
9	Migrate the legacy applications.		Targeted and legacy software effectiveness observation.
10	Migrate the legacy interfaces.		The legacy software cutover to modernized operational software.
11	Cut over to the target environment system.		

Table 1: compares the already existing migration approaches against the proposed migration model, which developed following the design science methodology. The first migration approach mentioned termed, CT, involve redeveloping the legacy software from scratch with no outlined process. The Cold Turkey has the high probability of running the risk of migration failure. Hence, it was discarded at an early stage (Bisbal, et al., 1997).

The second migration approach namely, CLS, involves the interoperation migration process that performed incrementally. This approach uses gateway coordinator that introduces great complexity in the migration process. The process of the CLS takes a longer time to complete. The CLS runs a risk of the developed target software to be also obsolete, and that may cost the company such that it may make a loss. Through these disadvantages, the CLS is abandoned (Brodie & Sronebraker, 1995).

The third mentioned migration approach namely, BM, was developed as a solution to the CLS. It is considered the best of the other mentioned migration approaches. BM has solved the problems encountered by the CLS by eliminating the use of the gateway coordinator. The elimination of the gateway coordinator reduced the complexity of the migration process. This approach also solved the problem of heterogeneity when migrating since it process performed separately from the legacy and the targeted software or system. However, this approach skips the critical phase, or it assumes that the environment to operate the legacy system is known. That is not always the case. Also, this approach is for general migration of the legacy software

or system, which does not get into details as to how the specific software or system, including its migration environment set up, is performed (Wu, et al., 1997).

Through the limitations of these approaches, a migration model is proposed termed, OLSMM. This model is based on the design methodology. The migration model introduces the ignored step by all the other approaches. This step is the second step in the process of the proposed migration model. It involves the determination of the legacy software operational environment since it is not always known. Also, this proposed model introduces the use of the automated translational tool to auto-translate the legacy code to the target code. This tool significantly reduces the time of translating the legacy code to the targeted code. Instead of manually programming from scratch, which is time-consuming and is error prone and effortful, this model eliminates that process. It also adds the technique of effectiveness observation for both the legacy and the modernized software that analyses the metrics of interest, i.e., CPU usage, Memory usage, and the Compilation time. The flowchart of the proposed comprehensive migration model is outlined in figure 5, and it shows the process to complete the migration task.

Chapter 3: Research Methodology

This chapter describes the design adopted by this research to achieve the aim and objectives stated in section 1.4 of Chapter 1. Section 0 discusses the methodology used in the study, the stages by which the methodology was implemented.

3.1 EXPERIMENTAL METHODOLOGY

The migration of the Jet Turbine Engine HoleFlow program from Pascal to C++ was completed following the proposed model termed, OLSMM. The details of the migration process are explained below.

3.1.1 The need for migrating legacy software

The legacy software, HoleFlow program, has been developed using old Pascal programming language that has come to instance obsolete. It is operational on the old machines that are no longer manufactured. The professional experts on this old Pascal programming language are scarce. For these reasons, the legacy software, HoleFlow program, is complicated to maintain, improve and expand. That is also due to the lack of its understanding because of the unavailable legacy software documentation. Furthermore, integrating legacy software, HoleFlow program, to newer systems is difficult because the new business software uses completely different technologies that are incompatible with the legacy software. All these reasons raised the need for the legacy software migration. Hence, why the migration of the Jet Turbine Engine HoleFlow Pascal program migration was necessary.

3.1.2 Migration Process Planning

This phase of the migration process involves planning on how the migration process would be carried out. The first thing to be considered is the migrating of legacy software benefits and risks. These benefits and risks help decide whether to proceed with the legacy software migration or not. When it is decided to proceed, the

identification of the tools needed to accomplish the migration of the legacy software to the target software is needed. Some of the tools are open source, and others are not. Hence, the identification of tool's cost is required to estimate what it will cost to complete the migration process. The other component to establish is the software specification requirements. That includes primarily the review of the input the legacy software requires to operate. Once the inputs it uses established, it can also be used for the target software. After confidently knowing the inputs required then the application user interfaces that a legacy software operates on is analyzed for preparing to build the user interface for the target software. From that legacy software user interfaces, design the modernized user interfaces for the target software. With the tools identified including their cost and the design of the target system is known, then the legacy software is translated using the automated tool. When it is now translated, it is tested if it works as it supposes to, else, it is corrected for correct functioning. All these steps are performed following a migration-working plan with the milestones. After all the milestones are accomplished, then the legacy software is replaced with the modernized software.

3.1.3 Legacy Software Operational Environment

The Jet Turbine Engine HoleFlow Pascal program had to be recompiled to satisfy that it is error free and correctly functional as it is before migrating it. Since the Jet Turbine Engine HoleFlow program was written using an old programing language, which is Pascal, and compiled in the Microsoft Disk Operating System, it was not able to compile in newer versions of Windows operating systems. Through the brief description that it was developed using Turbo Pascal and compiled using the Turbo Pascal Compiler and the fact that it was first released in the year 2000s, it was figured that is operational in the old operating systems. The use of the functions that call DOS libraries raised an idea that it is operational on the Microsoft Disk Operating System. Since this system is ancient and could not be found to test the Jet Turbine Engine HoleFlow program if it is working correctly or not, a tool termed Dos-Box, of which is the virtualization of the Microsoft Disk Operating System was used to compile the Jet Turbine Engine HoleFlow Pascal program. The compiler used to compile the Jet Turbine Engine HoleFlow Pascal program was Turbo Pascal Compiler version 7.0. To be able to use these tools, firstly, the Dos-Box version 0.74

was installed. The trick for this installed Dos-Box to operate without any restrictions is to choose the Dos-Box installation directory to be the Desktop. When the Dos-Box installation done successfully, the folder of the Turbo Pascal Compiler version 7.0 was inserted inside the Dos-Box folder to link the two tools. After that, that Turbo Pascal Compiler directory needed to be mounted in the Dos-Box using commands. The figures below show the directory of the installed Dos-Box with the Turbo Pascal Compiler folder in it and the process of mounting the Turbo Pascal Compiler to the Dos-Box using commands.

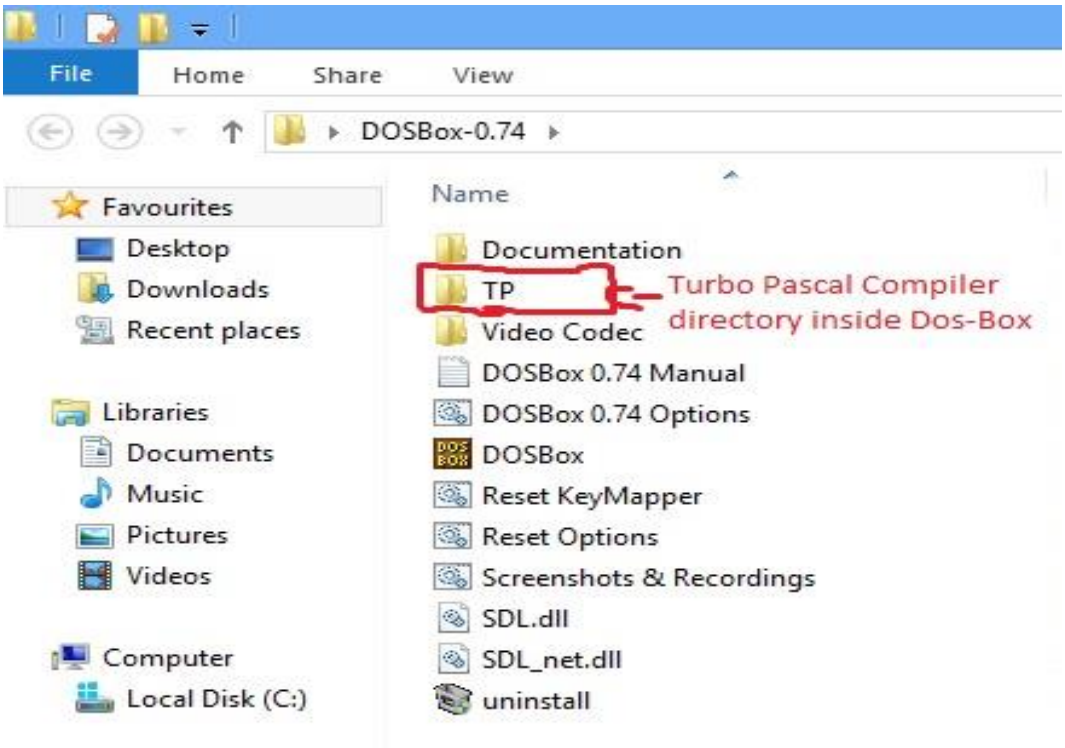

Figure 6: Turbo Pascal Compiler folder inside Dos-Box directory

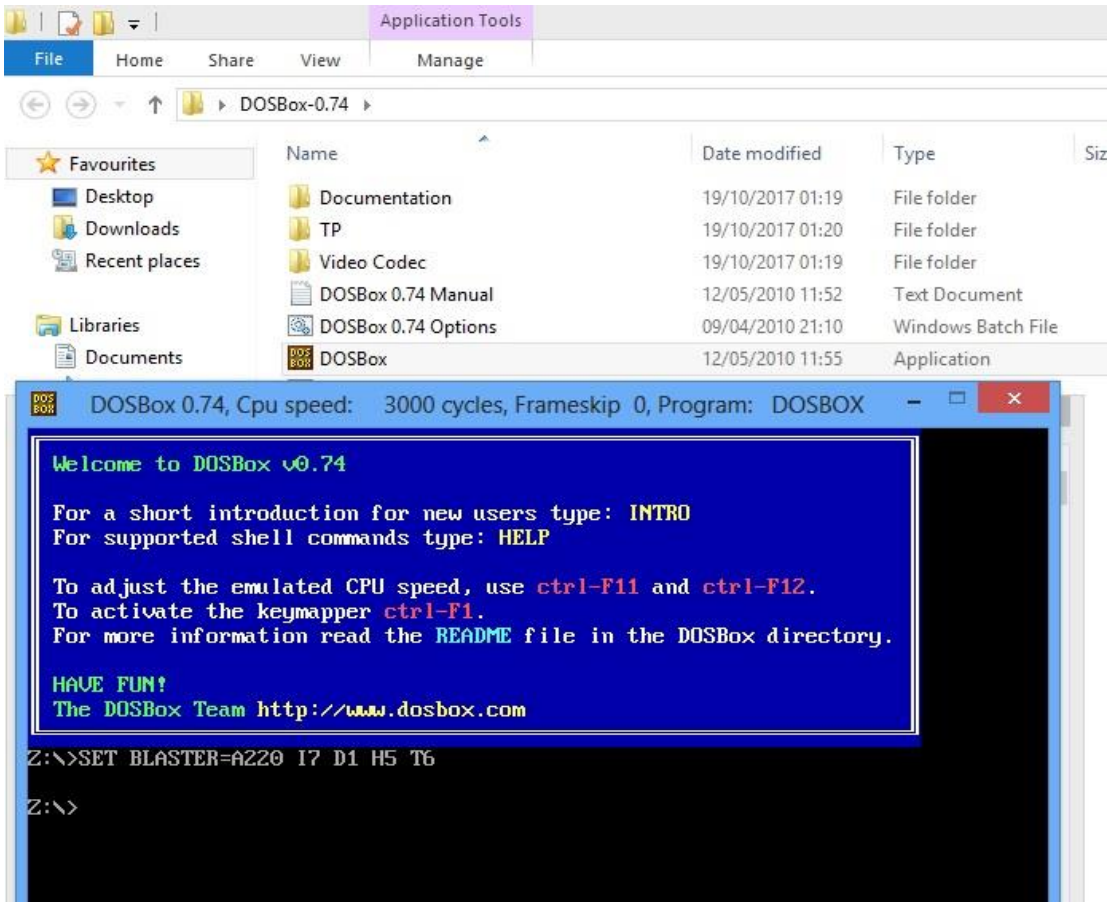

Figure 7: Dos-Box, which virtualizes the Microsoft Disk Operating System

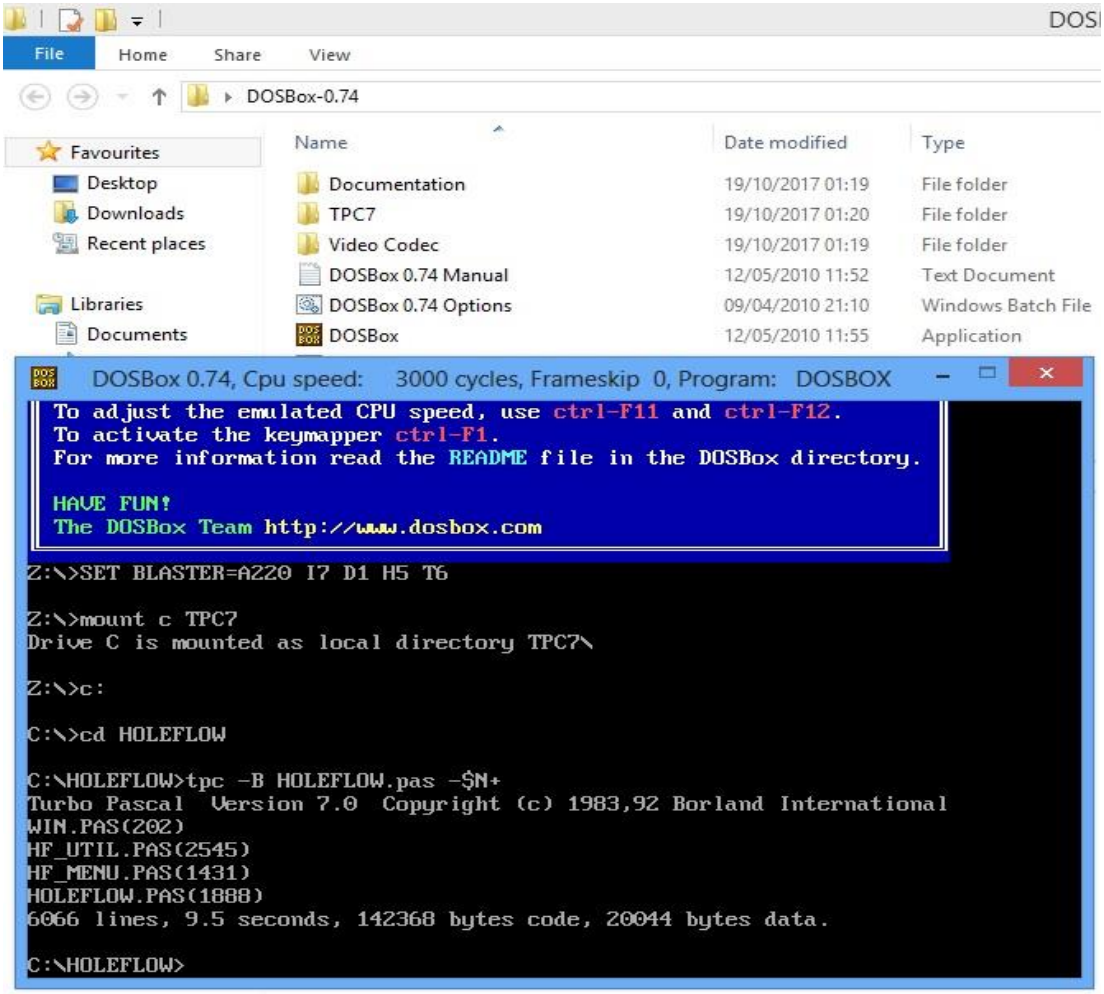

Figure 8: Pascal HoleFlow program compilation commands

In figure 8: there are commands followed to compile the Jet Turbine Engine HoleFlow Pascal program. Below are the commands explanations:

- Z:\>mount c TPC7

  This command mounts the driver C as the local directory TPC7. That tells that the drive C is the TPC7 directory.

- Z:\>c:

  Changes the drive from drive Z to drive C. Note that the drive C is the TPC7 directory.

- C:\>cd HOLEFLOW

  Inside the TPC7 directory, there is HOLEFLOW directory, which contains the Pascal files to be compiled. This command changes the directory to the HOLEFLOW directory.

- C:\HOLEFLOW>tpc –B HOLEFLOW.pas -$N+

  This command is for compiling the Pascal program. The first part [tpc] is specifying that a Turbo Pascal Compiler to be used. The second part [-B] is an option that tells the compiler to build. The third part specifies the filename of the compilable file. The third part [-$N+] enables the compiler to compile with the “8087” mode.

- C:\HOLEFLOW>tpc
  This command shows more options on how to use the Turbo Pascal Compiler.
  After the compilation, it executed on the 32-bit Windows operating system. It did not execute due to the breakneck speed this type of operating system has. Since the Jet Turbine Engine HoleFlow Pascal program was compiled in the Microsoft Dos Operating System that has the speed less than 200, it cannot handle the speed of a Pentium, which is 200 or higher.
  This problem was solved using the patch application termed PATCHCRT.

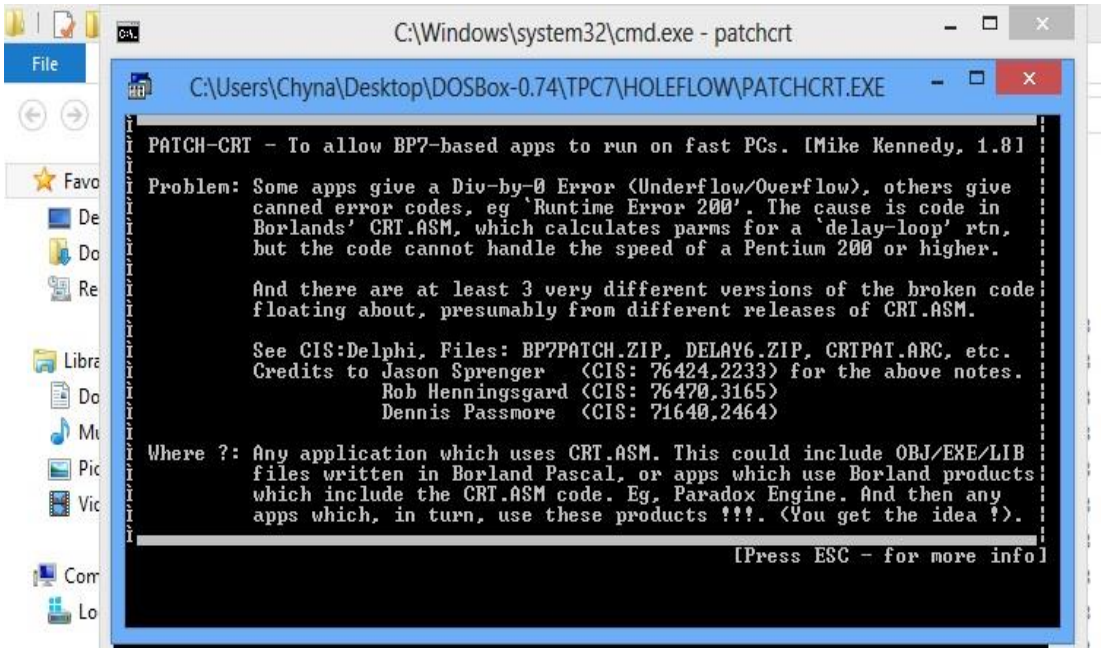

Figure 9: The Patch-CRT application

In figure 9: there is an explanation of the application speed problem with different Windows operating systems. This application should be in the same directory with the application to patch. Press ESC until it prompts for the filename of the application to patch. The full name of the application including it extension

[HOLEFLOW.exe] was entered, and the Jet Turbine Engine HoleFlow Pascal program patched successfully. It was then able to execute, at most, in the 32-bit Windows operating system.

3.1.4 Legacy Software Assessment and Analysis

Before the actual migration of the legacy software, its understanding of how it works was established. In ordinary cases, this is done by reading the software documentation that details the software requirements, architectural design, functionalities, and non-functionalities. Then executing the software to observe if it does what it is outlined in the document. However, the Jet Turbine Engine HoleFlow Pascal program came in with the document that is not detailing its operations. The document shows the calculations, and input and results in files.

This lead to actually executing the Jet Turbine Engine HoleFlow application for a better understanding of how the software operates. As the application was executed, all the details of its transactions from one state to another were noted, and the code was analyzed for functions fragments or code statements that produce the output of the application execution. With that state of the art, the acquired knowledge as to how the Jet Turbine Engine HoleFlow Pascal program operates was carried over to perform the legacy software migration from Pascal to C++.

3.1.5 Install and Setup Migration Environment

Now the Jet Turbine Engine HoleFlow Pascal program can be migrated with the confidence that it is functional.

The Jet Turbine Engine HoleFlow Pascal program migration was achieved by translating the Pascal code to C++ code using an automated tool in a 32-bits Windows 8 operating system. The automated tool used termed the Pascal-to-C/C++ (ptoc358). This tool requires no installation. What is required is that the Pascal files to be translated need to reside where the following files reside:

- An executable Pascal to C/C++ file: ptoc.exe
- The configuration Pascal to C/C++ file: ptoc.cfg

- The Turbo Pascal to C/C++ file: tptoc.pas
- The Array C++ header file: array.h
- The Pascal Library C++ header file: paslib.h
- The Pascal to C/C++, C++ header file: ptoc.h
- The Set C++ header file: set.h

All these files are located inside the folder of the ptoc358 translator.

The Jet Turbine Engine HoleFlow Pascal program consisted of four Turbo Pascal files that needed translation. These files include:

- WIN.pas
- HF_UTIL.pas
- HF_MENU.pas
- HOLEFLOWV2.pas

These files were put in the same folder where the translator files reside to perform the automated translation. The translational was achieved using the command prompt. The command is as follows:

- (Directory of Folder)>ptoc –turbo [name of the translatable file].pas

For this command to work must be in the directory where all the required files are located. The first part in the command [ptoc] instruct that a Pascal-to-C/C++ tool is to be used. The second part [-turbo] specifies that the Pascal file is a Turbo Pascal file. Then the last part specifies the Pascal translatable file name.

By default, the translator produces the equivalent C++ file of the Pascal file and the C++ header file necessary for that translated file. The name of the translated file by default will take the names of the original file.

Below are the figures 10 and 11 that show the relevant information explained above:

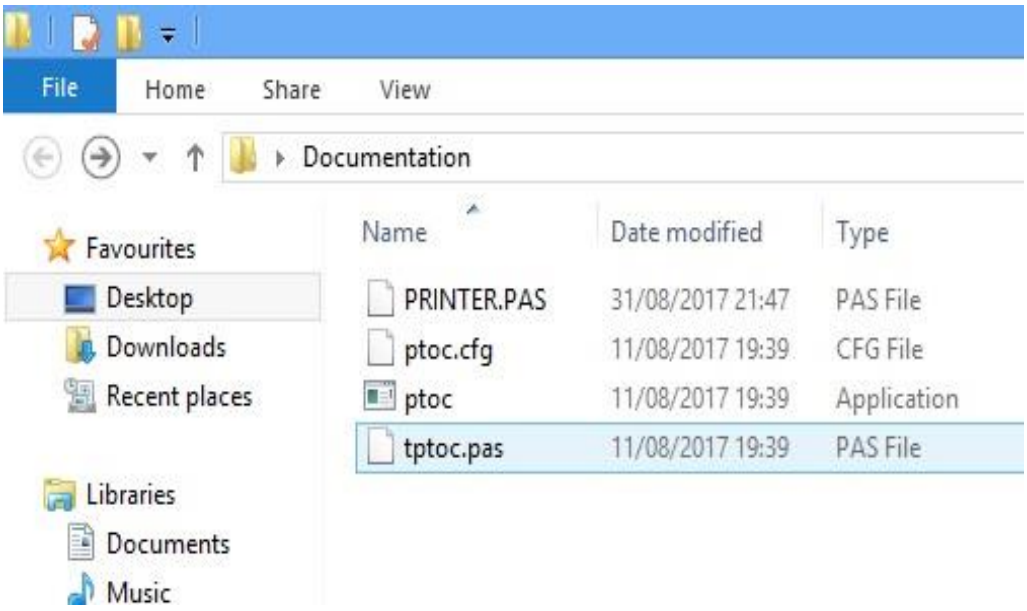

Figure 10: Translator directory consisting translator files and file to be translated

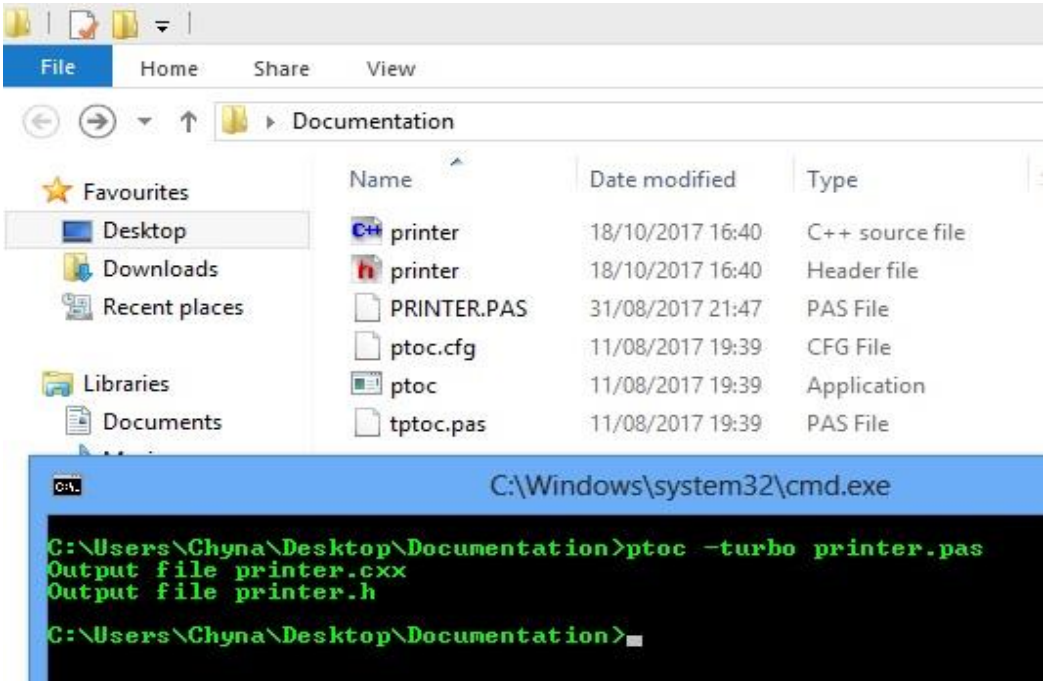

Figure 11: Translator directory with the translated files (printer.c++ and printer.h)

The figures 10 and 11: show the original Pascal printer file and the translated C++ printer files translated using the described command above respectively. With the translated files that are now in the C++ version, there were used to build the C++ console project. The C++ console project was created using the Code::Blocks version 16.01. Code::Blocks is the open source, cross-platform Integrated Development

Environment (IDE). The project was then compiled using the GNU GCC Compiler version 6.3.0.

3.1.6 Correct, Debug and Test Modernized Software

Using the created project, when the project compilation was done, it produced errors. These errors were libraries related errors. The libraries that were needed included the CRT, DOS, and the PRINTER Pascal libraries. These libraries were found in the old Pascal archives. There were then also translated using the Pascal-to-C/C++ tool and added to the C++ console project. The project was recompiled. When compilation was done, it produced less number of errors. The newly errors involved incompatibility of the translated C++ against the compiler used. These errors were eliminated by debugging the code and eliminating them according to the compiler specifications using the trail an error catch. This process was done until there were no errors produced. The program was attempted for the execution, and it crashed. It turned out that the code had logical errors. Hence needed to be refactored manually.

3.1.7 Refactor Modernized Software

After the logical errors were analyzed and understood the functions that were causing those errors were noted down with their description of what there are supposed to perform. Through the search, these functions were redeveloped and tested if they function in the way there are supposed to. The way there were tested was by putting in the input data into the function, let it process and receive the output. If the output corresponded to the expected output, then that function was considered functionally correct. After it was tested and approved, it was integrated into the Jet Turbine Engine C++ program.

This technique was performed with all the rest of the other functions until there were done. The project was compiled. After it compilation, it was executed successfully. Even though the program was running, it was not optimal according to its structure, the declaration of variables, the use of functions and the use of loops.

3.1.8 Optimize Modernized Software

The translated code has the structure that is not optimal since it was directly translated using the auto translational tool. Hence, the need for optimizing the code was necessary. The declared variables that spare the memory but not used are removed to reduce the memory usage that is unnecessary. The use of arrays with the memory allocation not used was rewritten to use the minimum memory allocation for the reduction of the memory usage. The loops, i.e., while, do while and a for a loop was reconstructed to loop when it is necessary and was managed to break or return when the condition was met for reducing the central process unit usage. The lines of codes that the translated code came with were very large due to the unnecessary declarations, splitting of statements that can be combined. Hence, these unused variable declarations and functions were removed, and the statements that can be combined into a single statement were combined. That optimized the modernized program.

3.1.9 Targeted and Legacy Software Effectiveness Observation

To proof conclude that the modernized software gained or degraded any performance, a resource utilization observation was performed. The metrics that focused on include central process unit usage, memory usage, compilation time, and the thread count.

3.1.9.1 Data Collection Setup

The application termed, Perfmon, of which is the Performance monitoring tool preinstalled in the Windows operating systems was used to collect the resource utilization data. The data that was collected using Perfmon tool includes the memory usage and the central processing unit usage. The compilation time was collected when the program was compiled. The data that was obtained was used to plot the line graphs for each HoleFlow application based on its resource utilization of relevant. Then, the resource utilization of the Jet Turbine Engine HoleFlow Pascal and C++ program versions were compared against each other. The average values were used to achieve the comparison of the resource utilization at large.

3.1.9.2 Resource Utilization Measurements and Analysis

The central process unit usage, and the memory usage data for the Jet Turbine Engine HoleFlow program was collected at the intervals of 60 seconds for 5 minutes. The average of these intervals in 5 minutes was collected. That data was used to plot the graphs. Since the Jet Turbine Engine HoleFlow program has a menu that consists of different operations, those menus were executed while collecting the data for each. That data was used to plot the line graph, which revealed what menu is consuming much of the systems resource. Then the average of the executed menu options for central process unit usage, memory usage, and thread count was used collectively for the Jet Turbine Engine HoleFlow program. That was done for both the C++ and the Pascal versions of the Jet Turbine HoleFlow program. Then the averages got from both the C++ and Pascal versions of the Jet Turbine Engine HoleFlow program was used to plot the column graph. The column graph was plotted to show the comparison between the two versions.

The compilation time data was also collected at the time of compiling the Jet Turbine Engine HoleFlow program five times. Then the line graph showing the attempts against the time it took to compile was plotted. The averages of the compilation times were calculated using the Excel software. Those averages were then used to plot the column graph for the compilation time comparison for both the C++ and Pascal versions of the Jet Turbine Engine HoleFlow program.

Following are the data tables and the respective graphs of the central process unit usage, memory usage, thread count and compilation time respectively.

3.1.9.3 Central Process Unit Utilization

Table 2: C++ and Pascal HoleFlow programs CPU usage data

Jet Turbine Engine HoleFlow Program CPU Usage for C++ and Pascal versions by Menu Options Operational		
Menu Options	**C++ CPU USAGE (byte)**	**Pascal CPU Usage (bytes)**
A-N	0.0104	80.9395
O	0.410808	80.7101
P	0.421197	80.628
R	0.202804	95.0697
S	0.005197	66.1612

T	2.199658	78.9395
Average	0.541677333	80.408

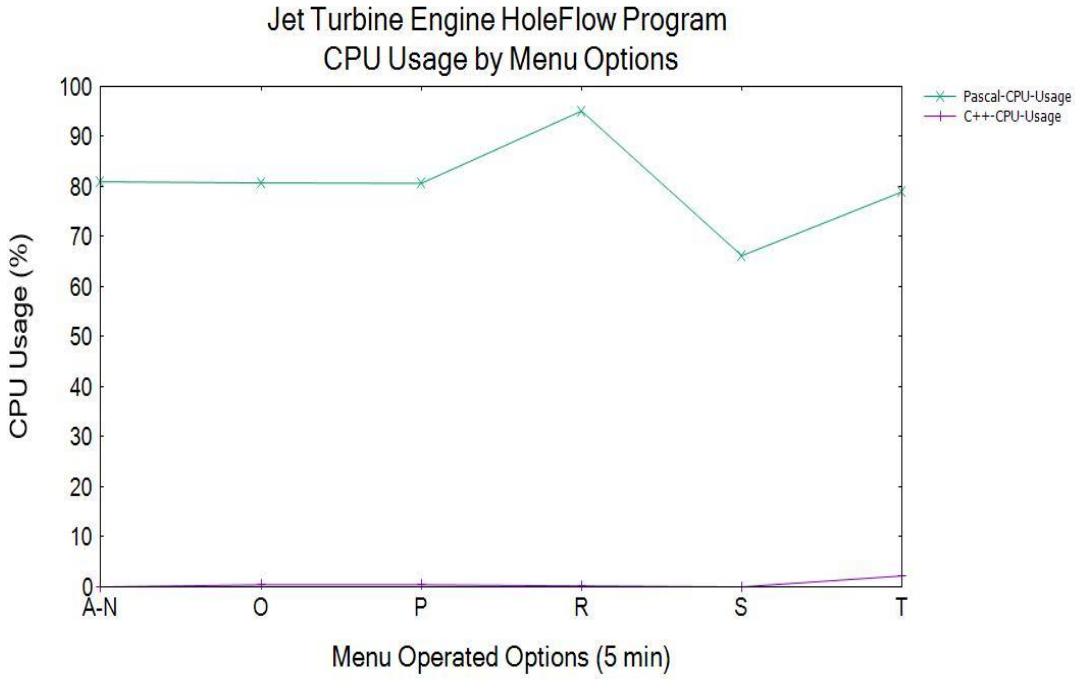

Figure 12: C++ and Pascal HoleFlow programs CPU usage line graph

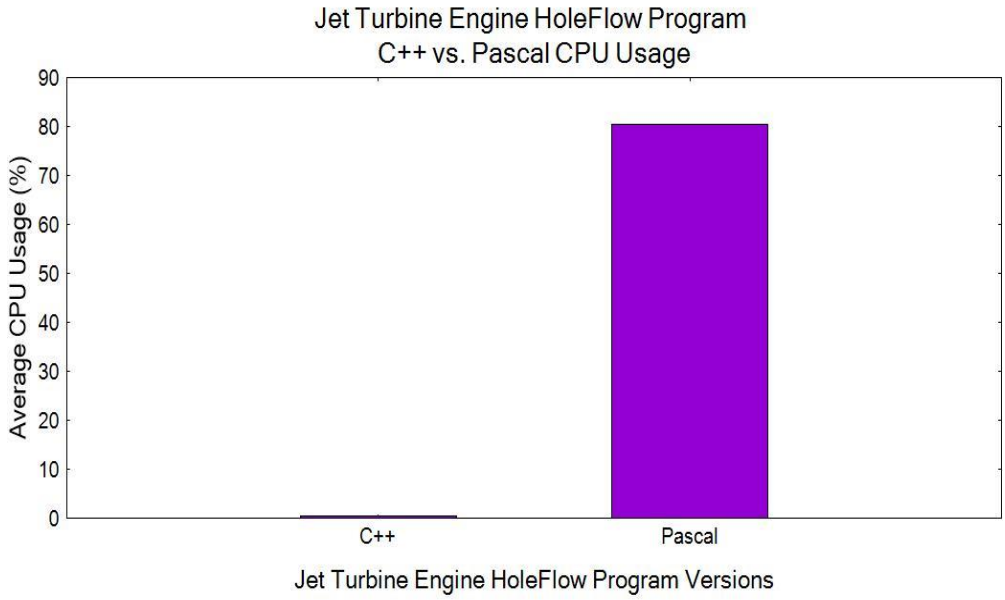

Figure 13: C++ and Pascal HoleFlow programs Average CPU usage column graph

Figure 13, shows that the CPU usage of the Jet Turbine Engine HoleFlow program for the C++ version is (0.5416%) and it less than that of the Pascal version, which is (80.4080%). That tells us that C++ program effectiveness on the CPU usage is better than that of the Pascal CPU usage. The difference or the gap between the two versions CPU usage is

$$CPU\ Usage\ Difference = Pascal\ version - C++version$$
$$= |80.4080 - 0.5416|$$
$$= 76.8664\ \%$$

The $CPU\ Usage\ Difference$ shows that the Pascal version of the Jet Turbine Engine HoleFlow program utilizes the CPU more than the C++ version with the $76.8664\ \%$ percentage. The reason being is that the Pascal program, when executed, it always listening for the key press to instruct the application on what to perform according to the menu it have. That listening action makes it always consuming the cpu while the C++ application does not listen for the key press but instructed by entering the option and press enter to execute that particular option in the menu which makes it utilize cpu only when it instructed to do so. Furthermore, these results corresponded with results by (Pereira, et al., 2017) on their study.

3.1.9.4 Memory Utilization

Table 3: C++ and Pascal HoleFlow programs Memory usage data

Jet Turbine Engine HoleFlow Program Memory Usage for C++ and Pascal versions of Menu Options Operational		
Menu Options	**C++ Memory USAGE (byte)**	**Pascal Memory USAGE (byte)**
A-N	8425472	1826816
O	8425472	1826816
P	8425472	1826816
R	8425472	1826816
S	8425472	1826816
T	8425472	1826816
Average	8425472	1826816

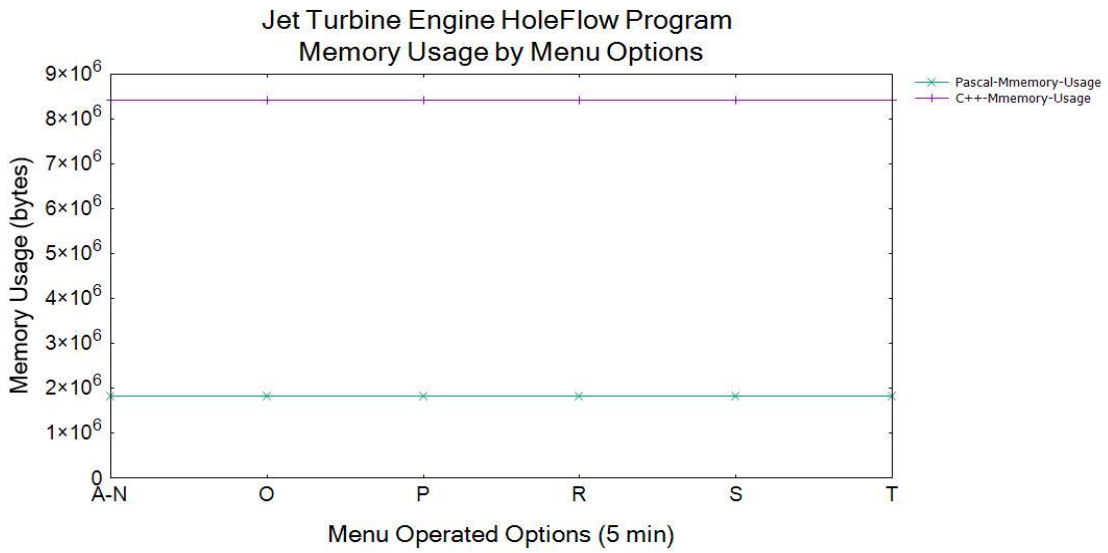

Figure 14: C++ and Pascal HoleFlow programs Memory usage line graph

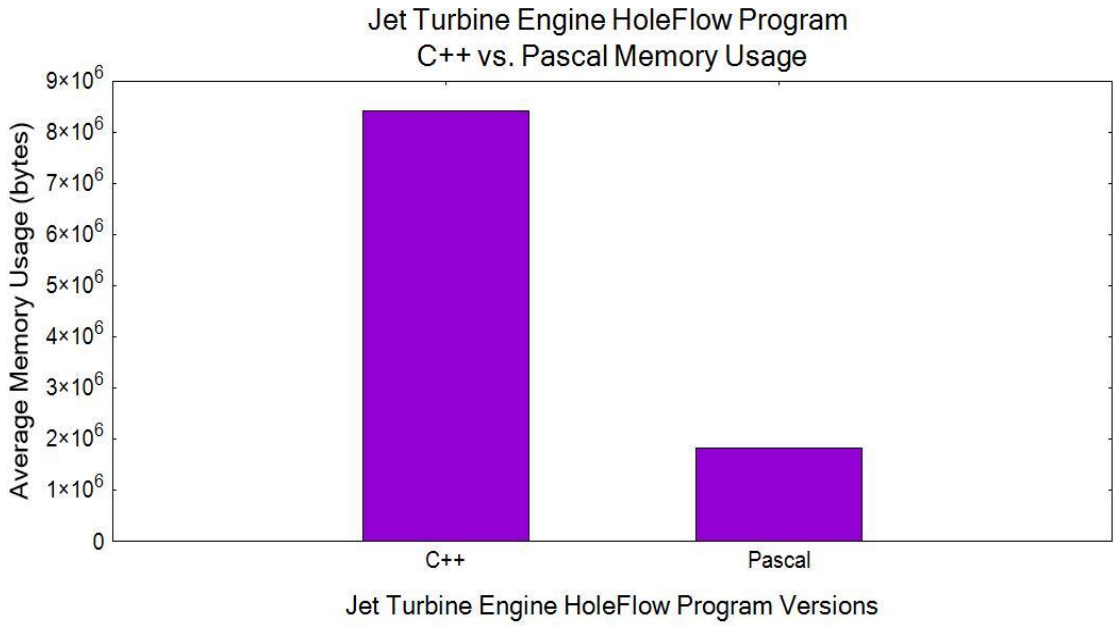

Figure 15: C++ and Pascal HoleFlow programs Average Memory usage column graph

Figure 15, reveals that the memory usage of the Jet Turbine Engine HoleFlow program for the C++ version is (8425472 bytes) while that of the Pascal version is (1826816 bytes). That shows that the Pascal program effectiveness on memory usage is better than that of the C++ memory usage. The difference or the gap between the two versions memory usage is:

$$Memory\ Usage\ Difference = \ Pascal\ version - C++version$$
$$= |1826816 - 8425472|$$
$$= 6598656\ bytes$$

The $Memory\ Usage\ Difference$ reveals that the C++ version of the Jet Turbine Engine HoleFlow program utilizes the memory more than the Pascal version with the value $6598656\ bytes$ of memory. The reason being is that the C++ program uses libraries that are not static and the dynamic linking overheads. The C++ HoleFlow program have seven libraries to link and it consist header files including the C++ files that contributes more to the memory consumption when the compiled application is executed. These results confirm with the results by (Pereira, et al., 2017).

3.1.9.5 Compilation Time

Table 4: C++ and Pascal HoleFlow programs Compilation Time data

Jet Turbine Engine HoleFlow Program Compilation Time for C++ and Pascal versions		
Compilation Attempts	**C++ Compilation Time (sec)**	**Pascal Compilation Time (sec)**
T1	4	9.5
T2	2	9.5
T3	3	9.5
T4	2	9.5
T5	1	9.5
Average	2.4	9.5

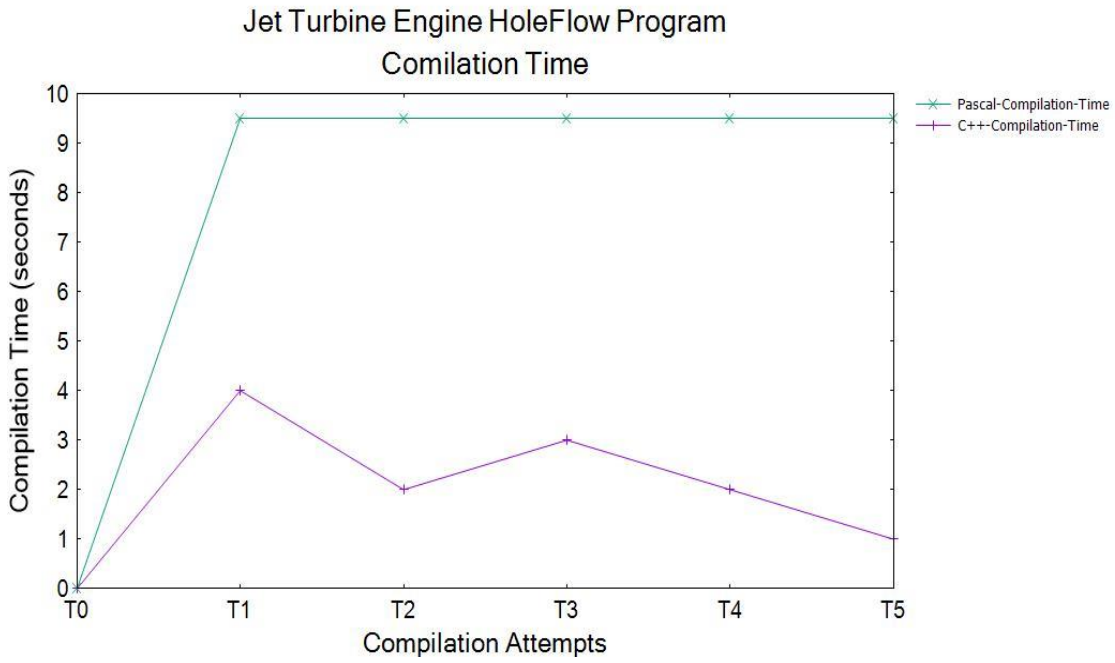

Figure 16: C++ and Pascal HoleFlow programs Compilation Time line graph

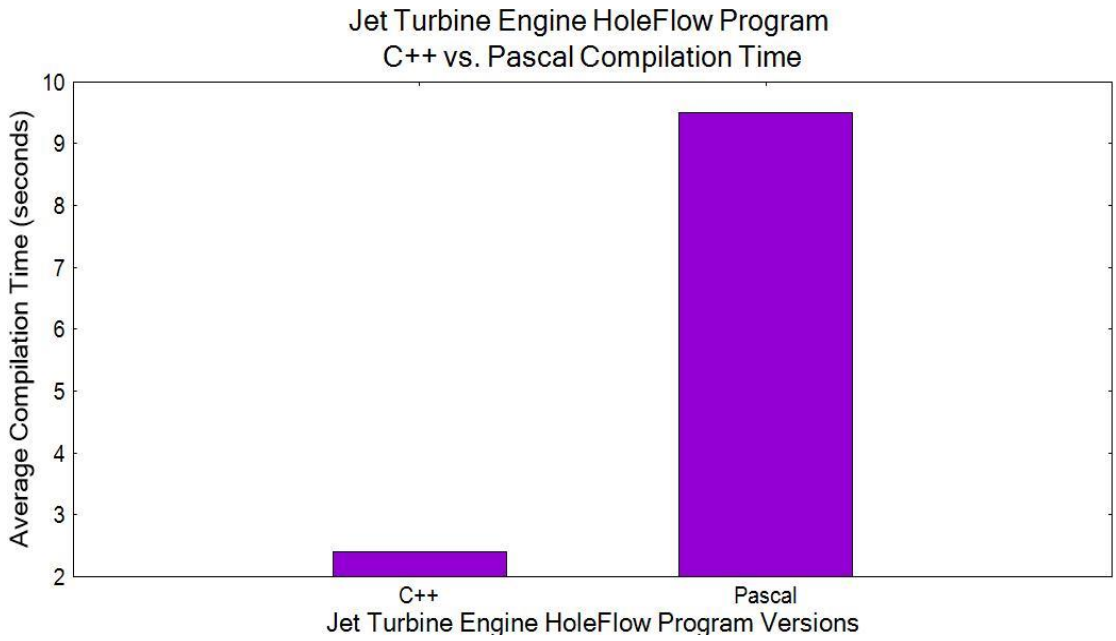

Figure 17: C++ and Pascal HoleFlow programs Average Compilation Time column graph

Figure 17, shows that the compilation time of the Jet Turbine Engine HoleFlow program for the C++ version is (2.4 sec) and that of the Pascal version is (905 sec). That shows that the C++ program effectiveness compiling time is better than that of the Pascal compilation time. The difference or the gap between the two versions compilation time is:

$$Compilation\ Time\ Difference = Pascal\ version - C++version$$
$$= |9.5 - 2.4|$$
$$= 7.1\ sec$$

The $Compilation\ Time\ Difference$ reveals that the Pascal version of the Jet Turbine Engine HoleFlow program compilation time takes longer than that of the C++ version with the value $7.1\ sec$ of the compilation time. The reason being why the Pascal version took long compilation time it because it was compiled on the Dos-Box of which is a tool for virtualizing the Disk Operating System that is the old Windows Operating System. That system is very slow in processing the compilation time such that the programs compiled using it cannot execute on the machines. Hence why the legacy software compilation is high. In order for the application to execute on the 32-bit Windows operating System, it had to be patched to fix it speed problem. (Anon., 2017)

3.1.10 Legacy Software Cutover to Operational Modernized Software

Since the C++ version of the Jet Turbine Engine HoleFlow program was able to execute, it was then operated while comparing it against the Pascal version of the Jet Turbine Engine HoleFlow program. The comparison was made by executing each operation in both versions with the same input. The output in both versions was compared to confirm if there are the same or not. If the output were the same, then that operation was considered to be functionally correct. The same technique was performed with the rest of the operations until there are all done. If the output of the specific operation was different from one another, then that operation was fixed until its output matches the output of the legacy software.

Chapter 4: Results and Analysis

Table 6 outlines the qualitative evaluation based on the existing legacy migration models against the Offline Legacy Software Migration Model. The evaluation of the existing migration approaches is adapted from the literature.

Table 5: Qualitative evaluation of the Offline Legacy Software Migration Model

Measurements	**CT** (Brodie & Sronebraker, 1995)	**CLS** (Brodie & Sronebraker, 1995)	**BM** (Wu, et al., 1997)	**OLSMM**
Complexity	CT does not consist of any steps to follow. It has no planning before the migration process. It is complicated to perform and has a high possibility of failure. (Bisbal, et al., 1997)	CLS uses gateway coordinator to perform migration interoperable. This gateway increases complexity due to the technology heterogeneity. (Wu, et al., 1997)	BM has phases to perform the migration. It does not include gateway coordinator as it performed separately from the legacy system that reduces it complexity. (Wu, et al., 1997)	This proposed migration model is built on the BM idea following a design science methodology in migrating the Jet Turbine Engine HoleFlow program. Since it provide steps that have detailed specifics on how to carry out the steps, that reduces its migration task complexity for the person who is performing migration task of the legacy software written in Pascal to C++ programming language.
Completion time	Due to the lack of the plan for the migration process. The time of completion for the migration task is not determined with certainty. Furthermore. Since legacy system migration is a complicated process, its completion time is long (year/s). (Bisbal, et al., 1997)	Because this approach migration process is performed interoperable and it migrates components incrementally. It takes a long time to complete probably years, because of the complexity of heterogeneity raised by the interoperability. (Brodie &	This methodology solves the problem of complexity that appears on the CT, and the CLS by formulating a flexible migration process phases and eliminating the use of gateways by letting the migration task performed separately. That condensed the time it takes to	The model explains each step with the detailed specifics on how to perform the migration. It provide the resource used to perform the migration. The person who is performing the migration task will not be required to research about the tools and methods used to migrate but simply use what is detailed in the document.

		Sronebraker, 1995)	complete the migration task. (Wu, et al., 1997)	
Clarity	With no steps to follow. With no plan to provide the guidance and milestones for completing the migration task. This approach has no clarity at all, as to how the migration should be carried out. (Bisbal, et al., 1997)	This approach includes steps to be followed to complete the migration task. However, it does not provide the details on how each step is supposed to be carried out. Therefore, it also lacks the clarity. (Brodie & Sronebraker, 1995)	In this methodology, each phase is detailed of what is supposed to be done, but how it was done, that is left of the developer who is performing the migration to figure out. Thus, its clarity is not sufficient as to how the migration phases are carried out. (Wu, et al., 1997)	This model is processed in steps. Each step is precisely explained how to carry out each task from the start until the end of the migration process with an example of migrating the HoleFlow program which is legacy software. That gives a clear understanding how the migration task is carried out.
Targeted and Legacy Software Effectiveness Observation	This approach does not show the effectiveness observation phase of any sort. (Bisbal, et al., 1997)	This approach does not include any effectiveness observation step of any sort. (Bisbal, et al., 1997)	This approach does not include any effectiveness observation step of any sort. (Wu, et al., 1997)	This model introduces the technique to observe the resource utilizations of both the legacy and the modernized software. This observation give details about how much the modernized does and the legacy software consumes computer resources. This may give guidance to the company operating that software about the consumption of computer resources.
Agility	Without the process to be followed, there would be no agility. (Bisbal, et al., 1997)	With the step-by-step migration process followed that is done incrementally, this method resembles the waterfall methodology. The waterfall methodology lack the agility. (Bisbal, et al., 1997)	This methodology forces the developer to move from one phase to another with the completion of the previous phase. Therefore, it lacks the agility in performing the migration task. (Bisbal, et al., 1997)	The way this model was developed, it was through experimental. During the process of developing, it conformed to the agile methodology. It involves the back and forth of migrating the legacy software, so that when there are changes or discoveries with the legacy software can

				quickly change to meat that discovery. Again as the languages advance this also helps to adapt quickly to the advancement of the programming language, compilers, and operating systems.

Table 7 outlines the OLSMM survey including the responses. The way the survey was conducted was based on the people who are doing the similar work of migrating legacy software. These people were limited to the quantity of three persons. Hence, the survey sample consists of only three persons. The reason why the survey is limited only to these people it because they can relate by looking at what models they have used and the OLSMM and be able to the question without the need actually to use the OLSMM.

Table 6: Offline Legacy Software Migration Model Survey

Question	**Yes**	**No**	**N/A**	**Remarks**
1. Does the model achieve what it intends to achieve?	3	0	0	• If its intention is providing migration guidelines that can lead to better success ratio in software migration then yes.
2. Are the model steps clear detailed?	3	0	0	
3. Is the model efficient to use?	3	0	0	• The model seems efficient, but steps seven and eight should be merged.
4. Does the model display a smooth flow of the steps?	3	0	0	
5. Is the model likely to satisfy any scenario of applicability?	3	0	0	
6. Are you satisfied with this model?	3	0	0	
7. Would you recommend this model to others?	3	0	0	• Since the model is used and it worked, it should be sufficient to recommend the model. • Well, it is a detailed model should apply to both small and large-scale projects.

Chapter 5: Conclusions, Summary, Recommendation and Future Work

5.1 CONCLUSIONS

What are the shortcomings of the existing migration approaches?

From the literature, it outlined that existing approaches have different shortcomings. Firstly, the CT has no process to follow. It clarity on how the migration process is done is not outlined at all. It has a high possibility of migration task failure. Secondly, the CLS has a shortcoming of high complexity as it employs the use of the gateways to maintain the data consistency throughout the migration process. That cause the significant increases in the completion time of the migration process. It also suffers from the heterogeneity problems since it migration process is performed interoperable of the legacy and the target software. Also, it resembles the waterfall methodology, which causes it not to be agile. On the other hand, the BM lacks clarity since it does not outline the specific details of how each phase is conducted precisely. Furthermore, it also resembles the waterfall methodology, which lacks the agility in of the migration process.

How does the proposed comprehensive migration model overcome the shortcomings of the existing migration models?

The OLSMM employs the design science methodology. OLSMM focusses on migrating the legacy software written in Pascal programming language to C++. To overcome the shortcoming that includes the complexity, and more prolonged completion time the OLSMM eliminates the use of the gateways, which exist in CLS (Brodie & Sronebraker, 1995). The BM does not outline clearly the specific details of the migrations phases, and it resembles the waterfall methodology. Hence this methodology lacks the agility (Wu, et al., 1997). To overcome the shortcomings concerning the clarity and agility in the BM, OLSMM explains each step of it migration process in specific details. It also includes the snapshots to give the graphical representation of the start and the end process of each step when it is necessary. By doing that, the developer who is performing the migration task has a clear understanding and an easy way to follow the process of completing the

migration task. Also, the OLSMM introduces the step of observing the effectiveness of the modernized software against the legacy software. The function of this step is to compare the performance gains, or lost, if any, for the modernized software. That will give guidance to the companies or organizations about what needs to be updated in the future concerning the modernized software.

What is the effectiveness of the proposed comprehensive migration model against the existing model?

The effectiveness of the OLSMM was evaluated by conducting the qualitative study and Survey. Also, the OLSMM was put into practice by migrating HoleFlow program. That was done to validate the OLSMM differences against the CT, CLS, and BM. The qualitative study shows the OLSMM is more efficient compared to CT, CLS, and BM because it provides more detailed steps to be followed to perform the legacy software migration. The survey conducted supplements the outcomes of the qualitative study. All the responses to the questions that were included in the questionnaire to conduct the survey were in favor of the OLSMM based on the responses collected. Therefor our model is different from the other three models based on qualitative study and the feedback that was provided by the participants from the survey.

5.2 SUMMARY

This study has addressed the issue of the gap in the comprehensive legacy software migrating model by posing the following questions:

What are the shortcomings of the existing migration approaches?

The existing migration approaches CT, CLS, and BM consist of several shortcomings that include the complexity, agility, completion time, clarity, and targeted and legacy effectiveness observation. Once the shortcomings of the existing migration approaches are established, and the causes of these shortcomings understood, the proposition of the comprehensive migration model is developed in overcoming these shortcomings.

How does the proposed comprehensive migration model overcome the shortcomings of the existing migration models?

The OLSMM have overcome the shortcomings of the existing legacy migration approaches CT, CLT, and BM by providing the precise, detailed steps for the model. Each step is outlined with clarity including the tools necessary for completing the migration process. Also, it includes the snapshot to provide the graphical view to observe how the step is carried out.

What is the effectiveness of the proposed comprehensive migration model against the existing approaches?

The effectiveness evaluation of the proposed comprehensive migration model, OLSMM, is based on a qualitative study that was done, and it was supplemented with the survey. The qualitative study reveals that the OLSMM performs differently regarding complexity, completion time, clarity, effectiveness evaluation of the modernized software, and agility since it is a well-detailed model with the precise explanatory of what is required and how to each step is carried out to complete the migration task.

5.3 RECOMMENDATION

The OLSMM successfully migrated the Jet Turbine Engine HoleFlow program written in Pascal programming language to C++. Therefore, this model can be recommended to the companies or organizations that are still operating the legacy software written in Pascal programming language to migrate their software to C++ so that their software can be modernized.

5.4 FUTURE WORK

The Jet Turbine HoleFlow program is a console-based application. Therefore, the future work of this study intends to advance the HoleFlow application to be a Graphical User Interface based application.

Bibliography

- Anon., 2017. *Engines.* [Online]
  Available at: https://www.grc.nasa.gov/www/k-12/UEET/StudentSite/engines.html
  [Accessed 15 June 2017].

- Anon., 2017. *Migration of Legacy Information Systems.* [Online]
  Available at: http://what-when-how.com/information-science-and-technology/migration-of-legacy-information-systems/
  [Accessed 20 September 2017].

- Anon., 2017. *MIT School of Engineering | » How does a jet engine work?.* [Online]
  Available at: https://engineering.mit.edu/engage/ask-an-engineer/how-does-a-jet-engine-work/
  [Accessed 5 June 2017].

- Anon., 2017. *Turbo/Borland Pascal Patches for CRT problems on fast PC's.* [Online]
  Available at: http://www.ipnet6.org/tppatch.html
  [Accessed 5 September 2017].

- Bisbal, J. et al., 1997. A survey of research into legacy system migration. *Technique report.*

- Brodie, M. L. & Sronebraker, M., 1995. *Legacy Information Systems Migration: Gateways, Interfaces, and the Incremental Approach.* San Francisco: Morgan Kaufmann Inc.

- David, A., 2013. Source-to-Source Translation and Software Engineering. *Journal of Software Engineering and Applications,* 6(4A), pp. 30-40.

- Graham, I., 1995. *Migrating to Object Technology.* 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

- Merlo, E. et al., 1995. Reengineering User Interfaces. *IEEE Software,* 12(01), pp. 64-73.

- Oest, O. N., 2008. *Legacy Software Migration.* [Online]
  Available at: http://mil-embedded.com/pdfs/LegacySWMigration.Jun08.pdf
  [Accessed 13 August 2017].

- Pereira, R. et al., 2017. *Energy Efficiency across Programming Languages.* New York, SIGPLAN International.

- Qui, L., 1999. *Programming Language Translation.* [Online] Available at: https://ecommons.cornell.edu/handle/1813/7400 [Accessed 6 August 2017].

- Ruhl, M. K. & Gunn, M. T., 1991. *Software Reengineering: A Case Study and Lessons Learned.* Gaithersburg, Special Publication (NIST SP).

- Wirth, N., 1971. *The programming language Pascal.* [Online] Available at: https://link.springer.com/article/10.1007%2FBF00264291?LI=true [Accessed 4 August 2017].

- Wu, B. et al., 1997. *The butterfly methodology: A gateway-free approach for migrating legacy information systems.* In Engineering of Complex Computer Systems, 1997. Proceedings., Third IEEE International Conference on (pp. 200-205), IEEE.

Appendix A

Offline Legacy Software Migration Model Questionnaire

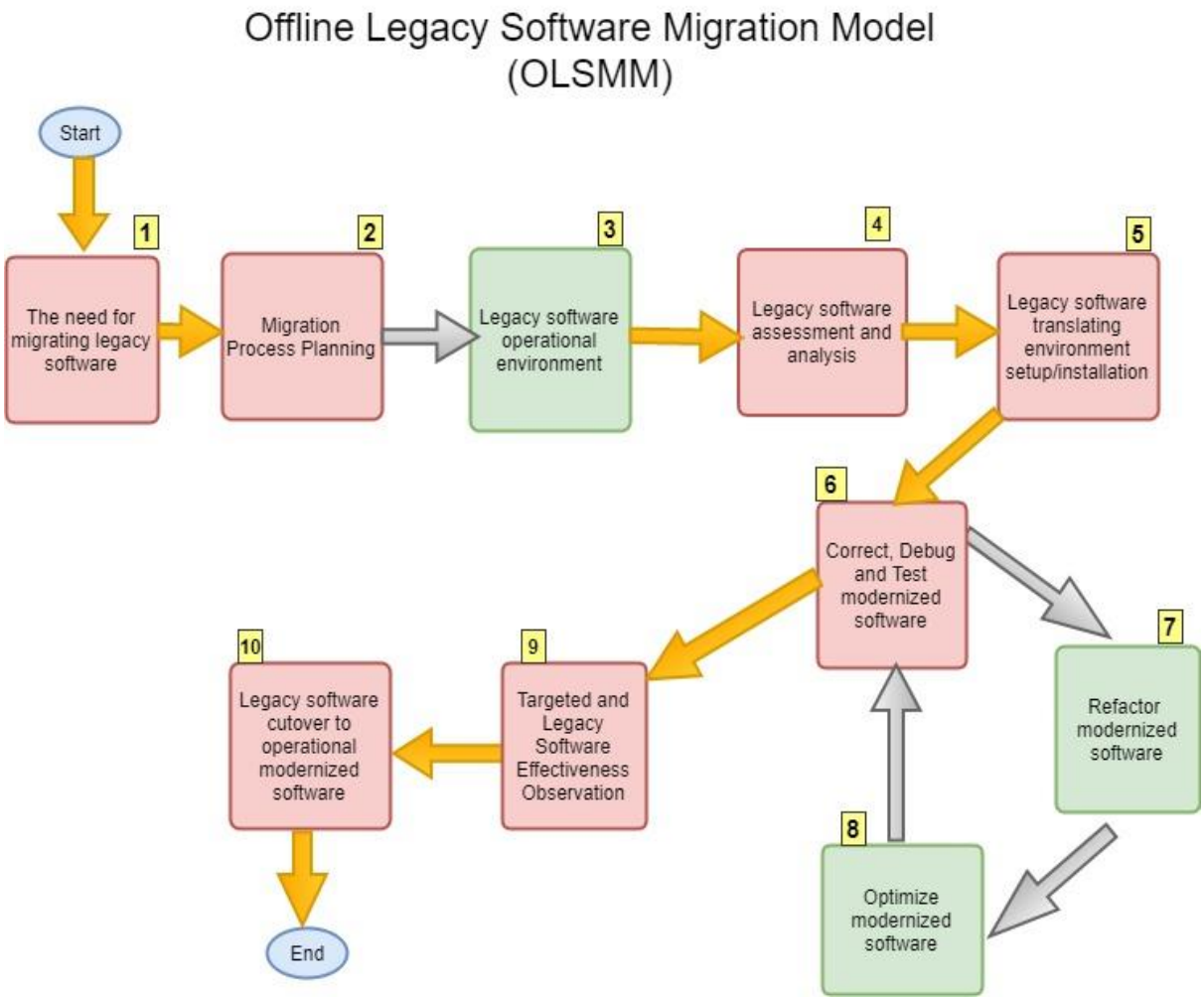

Figure 18: Proposed Migration Model

Table 7: Survey questions about the proposed migration model

Question	**Yes**	**No**	**N/A**	**Remarks**
8. Does the model achieve what it intends to achieve?				
9. Are the model steps clear detailed?				
10. Is the model efficient to use?				
11. Does the model display a				

	smooth flow of the steps?				
12.	Is the model likely to satisfy any scenario of applicability?				
13.	Is this model satisfying?				
14.	Can give recommendations for this model to others?				