Complexity Analysis of Simulink Models to improve the Quality of Outsourcing in an Automotive Company

Jeevan Prabhu
August 2010
ABSTRACT

Usage of software in automobiles is increasing rapidly and since software development is not the core business of automobile industry, outsourcing the development activity of the software is gaining more importance. Quality is the main concern when any product is outsourced. Quality of the product can be judged by various attributes like its functionality, testability, maintainability etc. All these aspects are directly or indirectly depend on the implementation activity performed by the supplier. Hence the outsourcer has to have some knowledge over the suppliers’ activity also.

This thesis proposes an approach for outsourcing the implementation activity in an automotive company. The main aim of the project is to identify those delicate parts of the design, which are complex and can be prone to errors during implementation. We define the metrics which can be applied to the design documents and validate them by generating the code from the designs. These results can be used to identify the complex functionalities in the code and help the company to perform more rigorous testing on those parts.
ACKNOWLEDGEMENTS

First, I would like to thank the Manipal University (MU) and Technical University of Eindhoven (TUE) for providing me an opportunity to expand my knowledge through this dual degree master program. I am greatly indebted to Prof. Dr. Jos Baeten, Dr. Manohara M. Pai, Prof. Dr. Mark van den Brand and Dr. Radhika M. Pai for considering me worthy for this program.

I would like to express my sincere gratitude to my project supervisors Dr. Alexander Serebrenik, Ir. Martijn Klabbers and Ir. Joost Gabriels for their continuous support and guidance throughout the thesis work. The encouragement and the valuable suggestions given by them immensely helped in making the project a grand success. I also thank Mr. Sanjay Singh for being my guide from Manipal.

Finally, I would like to thank my family, friends and colleagues at LaQuSo for their moral support and help.
CONTENTS

1 INTRODUCTION ..............................................................................................................06
1.1 Introduction to an Automotive Company ...............................................................07
   1.1.1 Current Process ..............................................................................................08
   1.1.2 Problems faced ..............................................................................................09
1.2 Research question ..................................................................................................10
1.3 Report Outline .......................................................................................................10

2 PRELIMINARIES .........................................................................................................11
2.1 Outsourcing ............................................................................................................11
   2.1.1 Overview ........................................................................................................11
   2.1.2 Outsourcing in Automobile Industry ...............................................................12

2.2 Simulink Introduction ............................................................................................13
   2.2.1 Model structure ............................................................................................14
   2.2.2 Colour Coding ...............................................................................................15
   2.2.3 Source code generation ..................................................................................16

3 APPROACH ................................................................................................................20
3.1 Assessment process ...............................................................................................20
   3.1.1 Software Complexity Metrics ......................................................................21
   3.1.2 Model Metrics ................................................................................................22
   3.1.3 Source Code Metrics .....................................................................................33
   3.1.4 Applying metrics to simulink models .............................................................34

4 CASE STUDIES .............................................................................................................40
4.1 Case study 1 ............................................................................................................40
4.2 Case study 2 ............................................................................................................40
4.3 Metric results Comparison and analysis ..............................................................40
5 CONCLUSION.............................................................................................................46
  5.1 Problems and Solutions.........................................................................................46
  5.2 Future Work...........................................................................................................47

6 REFERENCES..............................................................................................................48
1. INTRODUCTION

Outsourcing of software development work is an important activity carried out by many companies. Outsourcing is carried out mainly because of the engineering challenges and also because the outsourced activity is not a central part of the core business. Outsourcing has both advantages and disadvantages. The major advantages are lower development costs, access to talent, increased capacity, decreased time to market etc [1]. At the same time problems related to communication and quality of the product may arise.

Geographical distributions and cultural differences between the outsourcer and the service provider can lead to lot of complications, especially in the transfer or hand-over moments. Also different development practices and skill levels may further add to the complexities.

Outsourcing of software development and management activity is carried out both in the IT industry and beyond. In the case automotive industry also outsourcing is gaining importance. The use of software for the automobiles is increasing every day and hence outsourcing of this software is also increasing. Many new innovative functions in automobiles are enabled and driven by software. The rapid increase of software and software based functionality has given rise to various challenges for the automotive industries [2] [3]. The company outsourcing the software for the various functionalities has to ensure that the software delivered is of good quality. The outsourcing company has to follow a systematic method to assess the quality of outsourced systems and should be in a position to assure quality of the product to the customers.

Hence in this research we propose a new approach to improve the quality of outsourcing in the case of automotive industry. It mainly gives insights on assessing the design models based on model’s complexity and also it gives a method to improve the testing process by prioritizing the test cases.
1.1 INTRODUCTION TO AUTOMOTIVE COMPANY

For our thesis, there were two case studies considered. These case studies were provided by an automotive company, which outsources the development of software for engines. The company develops the system design in the form Simulink models (discussed in section 2.2) which is given to supplier who carries out the implementation activity.

![Figure 1: Overview of Outsourcing Process](image)

In the next section we explain the detailed outsourcing process currently followed by the company and also explain the difficulties faced by them with respect to the quality of the product delivered.
1.1.1 Current approach

The automotive company has an Engine System Department which is responsible for constructing the Simulink models of the requirements. The Engine System Department gets the requirements from the pre-engine departments for various functionalities in natural language. The Simulink models are given to the supplier who develops the binaries for the chips and delivers back to the company. The following figure shows the detailed process of outsourcing carried out in the automotive company.

Figure 2: Detailed outsourcing Process of Automotive Company

The detailed outsourcing process currently followed by the company is explained in following steps.
1. The company handovers the requirement documents in text format and Simulink models to the supplier. It also provides the functional tests which are to be carried out at supplier side.

2. For supplier the input will be mainly in the form of text documents which has the requirements. Simulink models only give a pictorial representation of the system and requirements.

3. The supplier develops a hardware specific model out of the given inputs. This model is used to generate code for the system.

4. The code is further refined with manual labour and functional tests are applied on it to get PASS /FAIL results.

5. Finally the binaries of the code are handed over to the company.

6. Risk analysis is done at the company’s side and results act as inputs to tests.

7. The final product delivered is subjected to functional tests and tested for acceptance.

The functional tests which are given to the supplier are derived from the requirements. These tests mark an indication to the progress of the implementation activity. For example, if there are ten functional tests to be carried out and if the company wants to know the progress of the implementation, supplier may inform the company about how many tests are successfully carried out and how many are still remaining.

The risk analysis phase basically identifies the important functionalities and the risks involved in them. It also helps in improving the final testing phase. The functional tests are modified and prioritized based on risk analysis and these tests are used in acceptance testing.

1.1.2 Problems faced by Automotive Company

As the company is outsourcing the software implementation activity, it is unaware of the quality aspects considered by the supplier. Also it has limited amount of time to test the delivered product from the supplier. It does not want to lose its customers’ confidence by delivering poor quality products. Hence it wants to gain more control over the systems quality by performing more critical testing and deliver high quality engines to the customer.
Hence the automotive company wants to have a method by which they can get an indication of the product quality and they can gain more confidence about the products they deliver. This research is to help the automotive company in finding a new method which they can rely upon.

1.2 RESEARCH QUESTIONS

The aim of the research work is to give an indication of the quality of the product delivered by the supplier in the case of outsourcing. To assess the quality of the system, we try to estimate the complexity involved in the system design as it is directly or indirectly related to system quality. In this thesis we try to estimate the complexity of the design models which are developed in Simulink (discussed in Section 2.2). Hence we try to answer the following research questions.

1. How can we measure the complexity of the Simulink design models?

2. How can we validate the measured complexity of the models?

3. Can we prioritize the tests using the above results?

4. How can the automotive company make use of these results?

1.3 DOCUMENT OVERVIEW

The document is divided into five chapters. Chapter 2 will give more insights on outsourcing with respect to automotive company and the challenges involved. Also we discuss the design models and code generation from the design models. Chapter 3 defines a new approach to outsourcing and validates the process by applying it to case studies in Chapter 4. We also specify a method to prioritize the tests. Finally, Chapter 5 gives conclusions and recommendations.
2 PRELIMINARIES

This chapter gives a brief description of the preliminary knowledge related to this thesis. The basics include the literature on outsourcing, methods followed, its benefits, problems involved. We also discuss the toolset involved in developing the design.

2.1 OUTSOURCING

From the last decade, the software industry has become one of the fastest growing fields [4]. The use of software technologies is entering into every domain which has increased demand for the resources required for developing the software products. The market is growing exponentially which has opened up lot of opportunities. At the same time the scale and complexity of the markets has increased the risks involved in developing the systems. The demands are changing so rapidly that maintaining the quality of the product has become a great challenge. These issues have given rise to outsourcing, which is a convenient method for companies because of their difficulty in developing the software product fast enough or change in infrastructures or staff expertise to keep up with the changes in the information technology world and the demands imposed on them.

2.1.1 Overview

Outsourcing is the practice of subcontracting responsibility for all or part of a function to a third-party service provider that manages the work. Companies use outsourcing for functions like development, planning, management, training, maintenance, operation of software services, skills, products or applications.

Outsourcing has been driven by the following factors [1]

- **Less Time**: Software development takes less time when it is outsourced. People can work around the clock in different parts of the world and hence the product can reach the market more quickly.

- **Low cost**: Outsourcing helps in reducing the cost for development and reducing workload on the employee. Outsourced labour is usually cheaper than the in-house labour.
• Lack of in house experience: If the internal resources of the company are not enough to globalize the company’s business, Outsourcing software development will be a wise method to get the work done.

• Flexibility: When we outsource, we do not have to spend time in recruiting, hiring, training, and housing employees for short-term projects.

• Focus on core competency: Outsourcing will provide a focused strategy to have a competitive advantage in the technological race.

Outsourcing is usually categorized into two types. One is outsourcing within the country and the second is to a foreign country. Outsourcing within a country is called domestic outsourcing or near-shoring, where the outsourcer and service provider both are within the same geographical location. In this case of outsourcing, the communication problems and the cultural differences are relatively less. Outsourcing to a different country is called as off-shoring, which is more complicated than near shoring.

2.1.2 Outsourcing in Automobile Industry
Various aspects of the software development activity can be outsourced. It ranges from planning of design to maintenance of the product.

Outsourcing in automobile industry is mainly driven by

• Lack of in house experience
• Focus on core competency

Hence in automotive industry usually implementation activity is outsourced due to engineering challenges [5]. This is an intermediate phase of development where the design is transformed to a source code which performs the desired operations.

Due to the increased use of software in the automobiles, the automotive companies have also started giving more importance to outsourcing the software for the engines. In the car industry, within only 30 years the amount of software in cars went from 0 to more than 10,000,000 lines of code. More than 2000 individual functions are realized or controlled by software in premium
cars. 50–70% of the development costs of the software/hardware systems are software costs [6]. Due to these facts transferring the work to a more skilled and equipped people is gaining more importance [5].

2.2 SIMULINK INTRODUCTION

Simulation is defined as the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system [7]. Simulation in automotive industry is very popular as it used to simulate the systems before they are actually built. Simulation is used to show the eventual real effects of alternative conditions and courses of action. Systems are tested and verified for various conditions using a simulation tool. In our case the automotive company uses Simulink tool for developing the design and it is further used to implement the model.

Simulink, developed by Math Works, is a commercial tool for modeling, simulating and analyzing multi-domain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. Simulink is being used in various areas like aerospace and defense, automotive, signal processing and medical instrumentation [8].

Figure 3 shows the Simulink library which has blocks used in constructing models. The basic elements of Simulink are blocks and lines.

Simulink Blocks can be classified as continuous, discrete, math, non-linear, signals, sinks, sources etc. The functionalities of the system are embedded in these blocks. The blocks are connected to each other by using lines with arrow head. Lines transmit signals in the direction indicated by the arrow. Lines must always transmit signals from output terminal of one block to the input terminal of another block.
2.2.1 Model structure

As mentioned in previous paragraph, the model in the Simulink is made up of set of blocks which are used to generate, modify, combine, output and display signals and these blocks are connected to each other using lines which transfer signals. Here we show a basic Simulink model in the following figure.
The model shows a subsystem which takes three input signals and three outputs. The subsystem in turn can have other subsystems which are in a hierarchy. Depending on the functionality of the system various blocks are chosen and the model structure is built.

2.2.2 Colour Coding

Colour coding is a standard followed by a company to represent the Simulink models. It differs from company to company. The representation of the models in a specific colour format will help in identifying and understanding the model easily and clearly. In the case of outsourcing, the supplier can easily identify the blocks involved in the models and differentiate them. Thus colour coding improves the visual representation of the models.

A colour coding standard document specifies the standard followed by the company. In outsourcing when a new supplier is approached, the colour coding document is also exchanged along with other company guidelines.
2.2.3 Source code generation

Outsourcing implementation of the design is one of the major activities performed by the automotive companies during the engine development. The automotive company considered in this thesis get the implementation work done by its supplier. The company sends the Simulink models to suppliers which are used to generate source code. The automotive company can itself generate a prototype of the code using their models to get an idea of the complexity of the code. This section explains the process of code generation from the simulink models.

MatLab, which has Simulink extension for constructing the Simulink models, also has a plug-in Real Time Workshop which is used to generate code. Real Time Workshop generates and executes stand-alone C/C++ code for developing and testing algorithms modeled in Simulink [9]. The resulting code can be used for many real-time and non-real-time applications, including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. We can also tune and monitor the generated code using Simulink blocks and built-in analysis capabilities.

Key features of Real time workshop include generation of optimized and customized code, generation of processor independent code and support single and multitasking environments.

2.2.3.1 Working with Real-Time Workshop

Real-Time Workshop is an integral part of the Simulink environment. We can interact with Real-Time Workshop using the Simulink Model Explorer graphical user interface [9].

From the Model Explorer we can generate code for the Simulink models or subsystem by selecting a Real-Time Workshop target. The Model Advisor in Simulink checks the model configuration and offers advice on how to optimize or tune a configuration set. The configuration parameters can be set using a configuration window.

- System target file: It defines a target, which is an environment for generating and building code for execution on an intended platform or certain hardware. Every target has a specific code format which is followed in code generation process.
- Language: We can specify the language C or C++ in which the code is generated.
• Make command: The make command is invoked when a build process starts. Each target has an associated make command, which is automatically supplied when the target file is selected using the System Target File Browser.

• Check model before code generation: We can choose whether to run Code Generation Advisor checks before generating code. It checks whether the model meets code generation objectives like code efficiency, traceability, debugging etc. The code generation report summary and file headers indicate the specified objectives and the validation result.

• The generate Code only - If this option is selected the build process generates code and a makefile, but it does not invoke the make command.

![Configuration window](image)

*Figure 5: Configuration window [10]*

There are other parameters in the Real Time Workshop pane like to create a Code Generation report during the build process, specify which comments are in generated files etc.
2.2.3.2 Structure of the generated code

After setting the above parameters, the source code can be generated. The tool generates following types of files: model files, interface files, data files and utility files. The model files will have the generated code for the functionality of in the model. The utility files modules are for data conversion and the interface files defines the interfaces. The data files will have the initialized data variables.

Figure 6 and 7 shows a model developed in Simulink and the corresponding code. The model has a set of arithmetic operation blocks, inputs, outputs and constant blocks.

![Figure 6: Example of a Simulink Model](image-url)
The code for the above model is generated in a report format.

Figure 7 shows the set of files generated on the left hand side of the figure. The files include the C files and include files.
3 APPROACH

This chapter explains the new approach which can be followed by an automotive company in the outsourcing process. The approach can be used by an automotive company which outsources the implementation activity to a supplier and which delivers the design in the form of Simulink models. To explain the process we consider the situation of an automotive company whose existing outsourcing process is already explained in Chapter 1. The new approach follows in next section.

3.1 ASSESSMENT PROCESS

The proposed approach aims at identifying the possible defects in implementation. Testing is the process by which the automotive company can know the possible defects or problems in the implementation. To do an effective testing, the automotive company should be able to predict the hot spots in the implementation where the errors or bugs can occur.

Hence the new process of outsourcing proposed in the following figure is a method to identify such delicate parts of the implementation, so that more importance is given to such parts during testing.

Figure 8 depicts the activities to be carried out at both the automotive company and the suppliers’ side. The automotive company uses the Simulink models to calculate the model metrics using the structure of the model. The calculated model metrics can be validated by generating the source code from the model and then comparing the source code metrics with the model metrics. The calculation of model metrics and source code metrics are discussed in the later sections. Based on these model complexity metric values, we prioritize the design modules. This can serve as a valuable input to the risk analysis phase which will then reflect in the improvement of the testing strategy by prioritizing the test cases.
Figure 8 also shows the comparison of the source code metrics from the supplier with the model metrics. This will be useful to have a better idea of the suppliers’ source code and identify the complex modules before the software is actually delivered.

3.1.1 Software Complexity

Software complexity is a term that includes various properties of a piece of software which affect internal interactions. The properties can be related to design, implementation, testing, maintenance etc. Software complexity measurement is an area of software engineering concerned with the measurement of factors that affect these properties.

According to Zuse [11] the software complexity is the difficulty to maintain, change and understand software. It deals with the psychological complexity of programs. Three specific
types of psychological complexity that affect a programmer's ability to understand the software have been identified [12] [13]: problem complexity, system design complexity and procedural complexity.

Problem complexity is related to the problem domain. It is assumed that complex problems are more difficult to understand than simple problems. This type of complexity is ignored as it cannot be controlled.

System design complexity can be divided into structural complexity and data complexity for structured systems [12]. Structural complexity deals with the concept of coupling [21]. Coupling measures the interdependence of modules of source code i.e. C functions calling other C functions. It is assumed that as the coupling between modules increases, the difficulty for a programmer to comprehend a given module also increases. Data complexity is related to cohesion. Cohesion is the measure of the strength of functional relatedness of elements within a module [21]. It is assumed that the more cohesive a module, the easier it is for a programmer to understand the module. In the traditional imperative languages, the structural and data complexity measures are based on the module's fan-in, fan-out, and number of input/output variables [14] [15].

Finally, procedural complexity is associated with the logical structure of a program. This complexity value is related to the length of the program (number of tokens) [16] or the number of logical constructs (sequences, decisions, loops) [17] that a program contains.

This thesis mainly focuses on identifying the structural and procedural complexities at the design level. We try to measure the complexities involved in the design of the system which will further reflect in the implementation. In the coming sections we identify a set of metrics to measure these complexities.

### 3.1.2 Model Metrics

In this thesis we are concentrating on the design complexity of the system. Since design complexity will reflect in the implementation activity, it is important to identify the complex
modules at the design phase and handle them. The design of the system will be in the form of a model. In our case we try to assess the design models which are in the form of Simulink models. Simulink models are commonly used in the designing the systems such as aerospace and defence, automotive, medical instrumentation, communications, electronic and signal processing.

3.1.2.1 Metrics selection

An examination of the literature on the measurement of software complexity provides us with various metrics like cyclomatic complexity [17], Halstead metrics [16], Henry Kafura metrics [15], lines of code [18], depth of nesting [19], OO complexity metrics like depth of inheritance, number of classes, and weighted methods per class [20].

In our thesis the complexity metrics are selected based on following criteria

- Metrics should be for imperative or procedural programming paradigm
- Metrics should be applicable to Simulink design models

Next we discuss each metric in detail and decide whether it satisfy the above constraints or not.

3.1.2.1.1 Halstead Metrics

Maurice Halstead developed Halstead complexity measures to determine a quantitative measure of module complexity using the operators and operands in the module. He introduced the metrics in 1977 and they have been used extensively since that time [16].

Halsted metrics need calculation of operators and operands from the source code. The source code is interpreted as a sequence of tokens. Each token is then clarified as an operator and an operand [16].

For example consider an operation: \( S = A + B + C \times D \)

Here \( S, A, B, C, D \) are operands and \( +, \times \) and \( = \) are operators.
Then the following are counted.

- number of distinct operators \((n_1)\)
- number of distinct operands \((n_2)\)
- total number of operators \((N_1)\)
- total number of operands \((N_2)\)

The number of unique operators \((n_1)\) and operands \((n_2)\) and the total number of operators \((N_1)\) and operands \((N_2)\) are calculated by collecting the frequencies of each operator and operand of the source program.

Once the \(n_1, n_2, N_1, N_2\) values are calculated, the following Halsted metrics can be calculated [16].

**Program Length:**
The program length is the sum of the total number of operators and operands in the program.

\[
N = N_1 + N_2
\]

**Program Vocabulary:**
The Program Vocabulary is the sum of the number of unique operators and operands.

\[
n = n_1 + n_2
\]

**Program Volume:**
The program volume depicts the information content of the program. It is calculated as the program length times the 2-base logarithm of the vocabulary size.

\[
V = N \times \log n
\]

Halsted Volume describes the size of the implementation of the algorithm. It is based on the number of operations performed and operands handled in the algorithm.

**Program Difficulty:**
The difficulty level of the program is proportional to the number of unique operators in the program. It signifies the error proneness of the program.
It is proportional to the ratio between the total number of operands and the number of unique operands i.e. if the same operands are used many times in the program, it is more prone to errors.

Difficulty: \[ D = \frac{n_1}{2} * \frac{N2}{n2} \]

Program Effort:
The effort to implement or understand is proportional to the Program Volume and the Program Difficulty of the program.

Effort: \[ E = D * V \]

Time to Implement:
The time to implement or understand the program is proportional to the Program Effort. Halsted has found that the effort by 18 give an approximate time in seconds.

Time to Implement: \[ T = \frac{E}{18} \]

Number of delivered bugs:
The number of delivered bugs correlates with the overall complexity of the software. The formula for Number of delivered bugs is

Number of delivered bugs: \( = (E * (2/3))/3000 \)

The basic idea for the calculation of Halsted metrics is the number of operators and operands. The model design will itself indicate the operations which will be reflected in the source code. Hence calculating the number of operators and number of operands from the design should be possible. Considering these facts we have decided to select this metric for evaluation of model.

3.1.2.1.2 Cyclomatic Complexity

Cyclomatic complexity is static software metric which calculated the complexity of the code. It is introduced by Thomas McCabe in 1976 and it measures the number of linearly independent paths through a program module. It provides an integer number that can be compared to the complexity of the programs. [17]
Cyclomatic complexity is often referred as McCabe's complexity. It can be used in the case of design and structural complexity of a system.

McCabe suggest seeing the program as a control graph, and then finding out the number of different paths through it. This count is known as the Cyclomatic Number and we refer to the Cyclomatic Number by $v(G)$ [17]. This is shown below

$$v(G) = E - N + P$$

where

$v(G)$ = Cyclomatic Complexity

$E$ = Number of edges

$N$ = Number of nodes

$P$ = Number of connected components or parts

We do not have to construct a program control graph to compute $v(G)$ for programs containing only binary decision nodes [22]. We can just count the number of binary nodes, and add one to this, as shown below

$$v(G) = P + 1$$

Where:

$v(G)$ = Cyclomatic Complexity

$P$ = number of binary nodes

The design model specifies the decision blocks which are responsible for various decision paths. These decision paths will be reflected in the source code and hence the cyclomatic complexity can also be calculated from the model.

3.1.2.1.3 Henry Kafura metrics

Henry and Kafura introduced Software Structure Metrics Based on Information Flow in 1981 which measures complexity as a function of fan in and fan out [15]. They basically indicate about the relationship between the various modules and the information flows between the modules.
These metrics are defined as

- Fan-in of a procedure as the number of local flows into that procedure plus the number of data structures from which that procedure retrieves information.
- Fan-out is defined as the number of local flows out of that procedure plus the number of data structures that the procedure updates.

Local flows relate to data passed to and from procedures that call or are called by, the procedure in question. Henry and Kafura’s complexity value is defined as "procedure length multiplied by square of product of fan-in and fan-out."

\[ C = \text{length} \times (\text{fan-in} \times \text{fan-out})^2 \]

where C is the complexity value.

If we consider a Simulink model, it basically has a set of blocks which perform some operations by taking inputs from other blocks. But the definition of fan-in says it is the number of procedures that a procedure calls. In the case of Simulink models the design is different. Most of the modules perform an operation and pass the output to the next procedure in a sequential manner. There are rarely procedures which call other procedures during their execution. Also the length factor refers to the number of statements which in the case of Simulink models is not possible to estimate by just using the model. Hence we exclude this metric as it is not applicable to Simulink models.

3.1.2.1.4 Lines of code

Lines of code is considered as program magnitude metric which determine the complexity of the program [18]. As explained in the previous metric, the lines of code is not possible to formulate for the model. If we have the design in the form of a pseudo-code then the number of statements can be compared with the line of code. But in our case we have only the design model which will be insufficient to estimate the lines of code. Hence we exclude this metric from our evaluation.

3.1.2.1.5 Depth of Nesting

Depth of nesting counts how deeply the nested modules are. As mentioned in the above metrics, not all modules in the Simulink models are nested. They usually occur in sequence and hence for
most of the modules the depth of the nesting will be zero. So this metric will not play a major role in deciding the complexity of the Simulink model and so we do not consider this metric.

3.1.2.2 Applying metrics to model

In the previous section we have discussed the set of metrics which can be used in this thesis. In this section we define the selected metrics for the Simulink model. We apply the same definition of the metrics as above to the Simulink models.

3.1.2.2.1 Model Halstead metrics

We apply the Halstead metrics to the Simulink models by calculating the number of operations taking in the model. The definition of an operator and operator are as follows.

- **Operator of the model**
  
  Any block in the model which has one or more incoming lines and has one or more outgoing lines is an Operator.

- **Operand of the model**
  
  Any incoming line to an operator is an Operand.

To identify the distinct operators we first have to compare the names (alphabets) of the blocks. If they match then we need to compare the operators shown in the block. If both of them match then they are considered as non-distinct blocks. For example consider

![Add1 and Add2 blocks](image)

Here both are Add blocks with addition operation. Hence they are non-distinct. But the blocks shown below have same block name but with different operators. Hence they are distinct.

![Add1 and Add2 blocks](image)
To identify the distinct operands we need to trace the incoming line backwards till the source. If it has a junction in between and that goes to another operator, then that operand is non-distinct.

To illustrate these definitions, consider Figure 9. The figure has three inputs, one output and three operational blocks. From the definition of operator and operand for the model we find that the total numbers of operators is three and total numbers of operands is six. Number of distinct operators is three since both the names and operators in the blocks are distinct. The number of distinct operands is five since one operator is shared by two blocks.

![Figure 9: Halsted Model Metrics](image)

If the operands are duplicated then also they are considered as non–distinct operands.
The general procedure followed to calculate the number of operators and numbers of operands are

1. Eliminate all the blocks which give the inputs to the model. That is the blocks which have only a link going out of the block but do not have link coming into the block. These blocks will usually include Inports, Constant blocks and calibration items.

2. Eliminate all the blocks which act as Outputs for the model. That is, the blocks which have only a link coming into the model but no outgoing link. These blocks are usually the Outports, Terminate block etc.

3. Count the blocks remaining after the above two steps which act as operators. Also count the number of distinct blocks which act as distinct operators.

4. Count the all the inputs to the remaining blocks which act as operands. Also count the number of distinct inputs which act as distinct operands.

5. Once the operators and operands are calculated, the Halstead metrics are calculated using the general formulae.

3.1.2.2 Model Cyclomatic Complexity

We apply the McCabe complexity definition to the model to find its complexity. We will concentrate on those parts of the model which implement the control flow in the model. In particular we find decision blocks in the model which directs the flow of information.

The basic decision blocks are IF-Else block, Switch block, While block and For Iterator block which are available in Simulink block library [9].

Consider the Figure 10 which describes a model with various conditional blocks.
Figure 10: Conditional Blocks

We can scan the model and see that there are two conditional blocks, If Else block and the Switch block.

Switch has three cases with a default case. Hence complexity of switch block is 2 and of If else block is 1. Hence total complexity is (2+1) +1 =4.

Steps followed in calculating the metrics are as follows

1. Scan through the model and count the number of blocks of type
   IF Else
   While Iterator
   For Iterator
   For each block add one to the model complexity.

2. For the Switch Case block,
   - Count the total number of case statements
   - Subtract one for default case and then add the count to the model complexity value.
3. For Max and Min blocks an implicit conditional statement has to be executed.

![Image of Min Max Block](image)

*Figure 11: Min Max Block*

Consider Figure 11 which uses Min function to get the minimum value out of In1 and In2, the operation is done as below.

\[
\text{MIN}[\text{In1, In2}] \rightarrow (\text{In1}>\text{In2})? \text{In1: In2}
\]

This adds one complexity value in the code since it has conditional statement. Hence we add one to total complexity value for MAX and MIN blocks.

4. Finally add one to the total count to get the final cyclomatic complexity.

In the IF Else block of Simulink models we come across feedback inputs which look like loops. But actually they just give the previous result as the input whenever the IF condition is false. Hence they will not add to the cyclomatic complexity.

Using the model metric values we can identify the complex modules and they can act as inputs to the risk analysis phase. Since we get the individual complexity of every functionality we also prioritize the test cases which will be useful if test time is less.
3.1.3 Source Code Metrics

The model metrics defined in the previous section give a complexity value for each model. We have tried to define the model metrics, without deviating from the actual definition of the metrics. Hence to make the model metric values more concrete we need to validate them.

The method of validation involves generating the source code for the target model and then using a suitable metric tool, calculating the source code metric values. The source code metric values are then compared with the model metric values. The metric values may coincide or they may have a relation between them or there may be no relation.

For code generation we can use a MatLab plug-in, Real time work shop which has been discussed in the previous chapter. As we know that the structure of the code has three types of files

1. Model files- These files include the main source code files which represent the functionalities in the design. The files include Model.c, Model.h and other include files.
2. Utility Files – These files are mainly data conversion files. They are tool specific and are generated for all models.
3. Interface files – These files include appropriate header files while using a model.

For the analysis we consider only the Model.c file since it includes the source code pertaining to the operations in the model.

The Model.c file in turn has four functions.

- Model_Initialize () - This function is responsible for the initialization of the real-time model, initialization of timing info, task periods etc.

- Model_Update () - This function updates the timing variables. Also it will have count of number of times the code of a task is been executed.

- Model_Output () - This function specifies the operations that are performed on the variables. This function shows the clear representation of the design in the form of code.
• Model_Terminate () - This function is responsible for termination of the model.

For the analysis we consider model_output function, since it is the function which performs the calculations and gives the output. Other functions are like initialize, update and terminate does not differ for different models. Hence for the analysis we only consider the model_output function.

Once the code is generated and metric values are calculated, then we can analyse the results and check whether there is any relation between them or not.

3.1.4 Applying metrics to Simulink models

The metrics defined for design models and source code need to be applied for a set of Simulink models. Hence we construct three models which include set of operational blocks and a set of decision blocks to check how the model complexity and the source code complexity vary with each other.

First we consider a relatively complex model with many operational and conditional blocks. We generate code for the model and calculate both the model and source code metrics. The model and the corresponding source code are shown in following figures.
Figure 12: Relatively complex model

```c
/* Model output function */
static void complexmodel_output(int_T tid)
{
    real_T rtb_MultiportSwitch;
    real_T rtb_LookupTable;

    /* Switch: '<Root>/Switch1' incorporates:
    * Constant: '<Root>/Constant1';
    * UnitDelay: '<Root>/Unit Delay';
    */
    if (complexmodel_P.Constant1.Value >= complexmodel_P.Switch1_Threshold) {
        /* Lookup: '<Root>/Lookup Table' incorporates:
        * Import: '<Root>/In3';
        */
        rtb_LookupTable = rtb_Lookup((const real_T *)
            (complexmodel_P.LookupTable_Xdata[0]), 11, complexmodal_U.In#,
            (const real_T *)((complexmodel_P.LookupTable_Ydata[0]));

        /* Switch: '<Root>/Switch' incorporates:
        * Import: '<Root>/In4';
        * RelationalOperator: '<Root>/Relational Operator';
        */
        if (rtb_LookupTable <= complexmodal_U.In#) {
            complexmodal_B.Switch1 = rtb_LookupTable;
        } else {
            complexmodal_B.Switch1 = complexmodal_U.In#;
        }
    } else {
        complexmodal_B.Switch1 = complexmodal_U.In#;
    }
}
```

Figure 13: Sample source code of model in Figure 12
Next we consider a relatively simple model and calculate its model and source code metrics.

![Figure 14: Relatively semi-complex model](image1.png)

![Figure 15: Sample source code of model in figure 14](image2.png)
This model is relatively simple model with less number of conditional and operational blocks.

Figure 16: Relatively simple model

```java
35 /* Model output function */
36 static void simplemodel_output(int Tid)
37 {
38   /* Switch: '<Root>/Switch' incorporates:
39     * Constant: '<Root>/Constants'
40     * Import: '<Root>/In3'
41     * RelationalOperator: '<Root>/Relational Operator1'
42     * UnitDelay: '<Root>/Unit Delay'
43   */
44   if (simplemodel_P.Constant1 Value >= simplemodel_P.Switch1 Threshold1)
45     simplemodel_Y.Out1 = (simplemodel_U.In3 <= simplemodel_U.In4);
46   else {
47     simplemodel_Y.Out1 = simplemodel_UWork.UnitDelay_DSTATE;
48   }
49 /* Output: '<Root>/Out2' incorporates:
50     * Constant: '<Root>/Constant'
51     * Import: '<Root>/In1'
52     * Import: '<Root>/In3'
53     * RelationalOperator: '<Root>/Relational Operator2'
54     * Sum: '<Root>/Subtract2'
55   */
56   simplemodel_Y.Out2 = (simplemodel_P.Constant1 Value < simplemodel_U.In1 -
57                         (simplemodel_P.Constant1 Value - simplemodel_U.In3));
58 /* tid is required for a uniform function interface.
59  * Argument bid is not used in the function. */
60 UNUSED_PARAMETER(Tid);
```

Figure 17: Sample source code of model in figure 16
Next we calculate the model and source code metrics for the above three models and see if there is any relation between them.

We have considered the two selected metrics for evaluation, cyclomatic complexity and Halstead metrics. First we calculate the cyclomatic complexity and Halsted metrics for the model as explained in section 3.1.6.2 and 3.1.6.1. Next step is to construct the model. During construction of the models we mainly concentrated on the structure of the models. We gave more importance to how the blocks in the model are connected rather than the actual data it has to work on. Every block internally defines the type of data which it can work on and hence we assume that initialising data will not alter the model code pattern. The values of the metrics are as follows.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Model Metrics</th>
<th>Code Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cyclomatic</td>
<td>Halstead</td>
</tr>
<tr>
<td></td>
<td>(N_1+N_2=N)</td>
<td>(n_1+n_2=n)</td>
</tr>
<tr>
<td>Complex Model</td>
<td>7</td>
<td>53</td>
</tr>
<tr>
<td>Semi complex</td>
<td>3</td>
<td>41</td>
</tr>
<tr>
<td>Simple</td>
<td>2</td>
<td>18</td>
</tr>
</tbody>
</table>

*Table 1: Metric values comparison for models in figure 12, 14, 16*

In the Table 1 \(N_1\) and \(N_2\) are the total number of operators and total number of operands. \(n_1\) and \(n_2\) are the total number of distinct operators and total number of distinct operands.

For the code generation purpose we used MatLabs’ Real Time Workshop. The default settings of the tool are used to generate the code. As already explained about the structure of the code generated in the Chapter 2, we considered only the model files generated for our evaluation. The other files are specific to the tool and remain same to any kind of model.

The table 1 shows that the cyclomatic complexity values of both model and source code exactly coincide. For Halstead metrics, the model metrics and the source code metrics show a positive increase with the complexity of the models. That is, as the model becomes complex (model
metrics value increase), the source code metric values will also increase. There the attempt was to see if there is any relation between the model and source code metrics. For actual validation we have to consider a large set of models.

3.1.4.1 Metric Tools used

For the source code metric calculation we have used two tools. The tools were selected based on the availability and functionality. The metric tools used are

- Understand: It is a static analysis tool for measuring various metrics including the cyclomatic complexity [23].
- Testwell CMT++: We use this metric tool to calculate Halstead metrics. [24].

In order to validate the method, we have considered two case studies from an automotive company. The case studies are the design models of a system which explain the functionalities involved in them using Simulink models. The next chapter gives more description on the case studies and the application and validation of model metrics. The results show that there is a relation between the model metrics and source code metrics and hence model metrics can be used to evaluate the models.
4 CASE STUDIES

Here we discuss two case studies which are from an automotive company. The case studies are the design documents which are developed in the automotive company and are given to a supplier to implement. This section deals with applying the new approach define in chapter 3 to the case studies.

4.1 CASE STUDY 1

The first case study which we have considered is Speed control management system. The module promotes the optimal shifting behaviour on vehicles equipped with manual transmission by limiting the acceleration and engine speed at lower gears. It has totally twelve modules.

4.2 CASE STUDY 2

This case study describes the requirements and design constraints for the ambient conditions fuel derate module. This module imposes a fuel limit based on the ambient conditions to protect the hardware. The system has mainly two modules.

4.3 METRIC RESULTS COMPARISON AND ANALYSIS

Both the case studies are analysed and metric values are calculated. The model metrics are calculated according to the definitions defined in the Chapter 3. The source code is generated using Real Time Workshop as explained in section 2.2.3. Then the metric values for the code are calculated as explained in the section 3.1.3.

4.3.1 Metric Values

The case study models have various modules which specify the functionalities. Each module is evaluated by calculating the model and source code metrics from them.

In Table 2, N1 and N2 are the total number of operators and total number of operands. n1 and n2 are the total number of distinct operators and total number of distinct operands. We calculate the N and n values separately for all the modules since all the Halsted metrics are based on these values.
The module J and module K represent different functionalities but since the number of blocks and links are same, the model and source code metrics are also same.

Next we find correlation between the model and source code metrics. Since for the cyclomatic complexity, the model metrics maps exactly to the source code metrics, we do not need any other analysis. For the Halsted metrics, we use Spearman’s Rank Correlation method [25]. It assesses how well the relationship between two variables can be described using a linear function.
Table 3: Correlation between Halsted model and source code metric

Table 3 shows that there is a strong positive co-relation between the Halstead model metric and Halsted code metric of both the case studies. The correlation between the total number of operators and operands of model and code is +0.84 and between total number of distinct operators and distinct operands of model and source code is +0.87. Both of them are calculated at 1% significance level [25].

4.4 PRIORITIZING THE TESTS
Testing is process which involves the mapping of requirements to the implementation. In this thesis we prioritize the modules by assigning a rank to the modules based on their complexity. Higher the complexity lower the rank assigned. First we assign rank for the modules based on their cyclomatic complexity.

<table>
<thead>
<tr>
<th>Case Study 1 Modules</th>
<th>Model Cyclomatic complexity</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module A</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Module B</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Module C</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Module D</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Module E</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Module F</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Module G</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Module H</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Module I</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Module J</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Module K</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Module L</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 4: Rank assignment based on Cyclomatic complexity

For the Halstead metrics, first we assign the ranks for \( N \) and \( n \) separately. Then the final rank for Halstead metrics is calculated by taking the average of the two ranks.

<table>
<thead>
<tr>
<th>Case Study 1 Modules</th>
<th>Model Halsted (N)</th>
<th>Rank</th>
<th>Module Halsted (n)</th>
<th>Rank</th>
<th>Avg. Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module A</td>
<td>19</td>
<td>10</td>
<td>14</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Module B</td>
<td>48</td>
<td>3</td>
<td>38</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Module C</td>
<td>29</td>
<td>5</td>
<td>24</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Module D</td>
<td>64</td>
<td>1</td>
<td>51</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Module E</td>
<td>15</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Module F</td>
<td>20</td>
<td>9</td>
<td>18</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Module G</td>
<td>26</td>
<td>6</td>
<td>21</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Module H</td>
<td>40</td>
<td>4</td>
<td>34</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Module I</td>
<td>17</td>
<td>11</td>
<td>14</td>
<td>10</td>
<td>10.5</td>
</tr>
<tr>
<td>Module J</td>
<td>21</td>
<td>8</td>
<td>20</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Module K</td>
<td>25</td>
<td>7</td>
<td>21</td>
<td>6</td>
<td>6.5</td>
</tr>
<tr>
<td>Module L</td>
<td>50</td>
<td>2</td>
<td>39</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5: Rank assignment based on Halstead metrics

Now we need to assign a final rank for the modules based on the cyclomatic and Halstead metric ranks.

We know that both the cyclomatic complexity and Halstead metrics measure different aspects of the program. One estimates the complexity based on the total number of decision paths and the other estimates the complexity based on the number of operators and operands used in the program.
For example consider the module D and module L from Table 2. The value of cyclomatic complexity of module L is 7 and that of module D is 4. But the total number of operators and operands of module L is 50 and that of module D is 64. Hence there are cases which show that the metrics are not related to each other. Hence, while measuring complexity, we have considered both of them equally important. Hence we decided that the final rank for the modules is assigned by calculating the average of both cyclomatic metrics rank and the Halstead metric rank. If the average rank calculated for two modules coincides then both of them will have same priority and hence any one of them can be considered first.

<table>
<thead>
<tr>
<th>Case Study 1 Modules</th>
<th>Average Rank</th>
<th>Final Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module A</td>
<td>10.50</td>
<td>12</td>
</tr>
<tr>
<td>Module B</td>
<td>3.00</td>
<td>3</td>
</tr>
<tr>
<td>Module C</td>
<td>4.00</td>
<td>4</td>
</tr>
<tr>
<td>Module D</td>
<td>1.50</td>
<td>1</td>
</tr>
<tr>
<td>Module E</td>
<td>7.50</td>
<td>10</td>
</tr>
<tr>
<td>Module F</td>
<td>10.00</td>
<td>11</td>
</tr>
<tr>
<td>Module G</td>
<td>7.00</td>
<td>8</td>
</tr>
<tr>
<td>Module H</td>
<td>6.00</td>
<td>7</td>
</tr>
<tr>
<td>Module I</td>
<td>5.75</td>
<td>6</td>
</tr>
<tr>
<td>Module J</td>
<td>5.50</td>
<td>5</td>
</tr>
<tr>
<td>Module K</td>
<td>7.25</td>
<td>9</td>
</tr>
<tr>
<td>Module L</td>
<td>1.50</td>
<td>2</td>
</tr>
</tbody>
</table>

*Table 6: Final rank of Modules*

The test case document was provided for the case study 1. It is an intermediate version which does not have all the test cases. It has six test cases. If we observe the test cases, each module in the design represents functionality and forms a test case. Hence we can prioritize these test cases based on the final rank assigned to the modules.
4.5 SUGGESTIONS TO AN AUTOMOTIVE COMPANY

• The first advantage of this thesis work is that the automotive company will get an indication of the complexity of the modules implemented by the supplier. The approach specified will help the automotive company to calculate the complexity of their design models. This complexity will be reflected in the implementation of the design by the supplier. Hence when the supplier delivers the software, the automotive company will have an estimated complexity of every module in the implementation which gives the company an opportunity to give suggestions to the supplier about the implementation.

• The approach will be very useful if the automotive company can get the source code metrics from the supplier. During implementation the supplier not only generates the code from the supplier but also manually modifies it. Hence the final source code metrics of the supplier will differ from the generated code metrics. So if the automotive company gets the final source code metrics from the supplier, then comparing the model metrics with those metrics will give a clear picture about the deviation of the metrics. Based on the amount of deviation between the metrics, the company can instruct the supplier to handle the implementation.

• Testing is an important task which needs to be carried out when the product is delivered by the supplier. Testing process is carried to check that the expected functionalities are implemented properly. In this thesis we estimated the complexity of every module in the design. These modules will be reflected in the test cases also. Hence the complexity values of the modules are used to prioritize the modules. This will give an order for the test cases, to be followed during testing based on the complexity. Such an ordering of the test cases will help the company to save time, by concentrating more on the complex modules rather than non-complex modules. Also the testing will be more effective.
5 CONCLUSION

5.1 PROBLEM AND SOLUTION

Outsourcing of software development is an important activity in most of the companies including the automobile industries. In automobile industries the software required for the engines is outsourced mainly due to lack of required resources. Quality of the product is the main concern in any outsourcing process. In this thesis we proposed a new method to improve the quality of the product in outsourcing.

We considered a scenario where an automotive company delivers the design to a supplier for implementation. The design will be in the form of Simulink models. The supplier implements the design and delivers the final product. Here the automotive company does not have any control over the implementation activity. The proposed method shows how to calculate the complexity of the design. Design metrics are defined and they are applied to the design models. These metrics are then validated by generating the source code from the model. This process is applied to a set of case studies from an automotive company and the results show that there is co-relation between the source code metrics and model metrics. This will help the automotive company to assess their design before its implementation and also it will help the company to instruct their supplier to do necessary modifications in the implementation. Thus the thesis will help in improving the quality of product and also the confidence of the company about the product.

We also explain how to prioritize the test cases based on the complexity values of the design. This will help the company to improve the testing process by concentrating on the delicate parts of the implementation and thus improving the quality of testing. Thus the thesis answers all the research questions mentioned in Chapter 1.
5.2 FUTURE WORK

In the thesis work, though we have shown the relation existing between the model metrics and source code metrics, it is for only two set of case studies. For any statistical analysis more the sample data more accurate the results will be. Hence to make the results more precise we recommend considering more set of case studies and performing the analysis.

For the analysis we have considered two metrics which satisfy the defined constraints. But there is scope for defining more metrics for the simulink models and then they can also be used as quality measure.

The model metrics which we have defined are calculated manually for the models. Further work can be done to develop a tool which calculates these metrics automatically.

The proposed approach shows that the model metrics and generated source code metrics are compared for validation of model metrics and based on the model metric values the complexities are identified. The generated code is just a prototype of the final code. It does not represent the final executable code satisfying all requirements. The supplier modifies the generated code by manual work and only when it satisfies all the user requirements it is delivered. Hence to validate the model metrics more precisely, there should be mutual understanding between the company and the supplier. They should work together in identifying the exact relation between the design models from the company and the final source code from the supplier. It is an evolving process. Over a time after applying the metrics to a set of projects, they can derive a more accurate and precise relationship between the model and source code.
6 REFERENCES


Available at: http://www.cs.colorado.edu/events/lectures/horn/horn.pdf


[9] Simulink Basic Tutorial:

[10] Simulink Configuration command box
http://www.mathworks.co.kr/access/helpdesk/help/toolbox/simulink/gui/brhq1w1.html


