

**ENHANCING CONTROL FLOW COMPREHENSION USING
ZOOM VISUAL FLOW (ZViF) TECHNIQUE TO REPRESENT
CONTROL STRUCTURES**

ROZITA KADAR

**UNIVERSITI SAINS MALAYSIA
2011**

**ENHANCING CONTROL FLOW COMPREHENSION USING
ZOOM VISUAL FLOW (ZViF) TECHNIQUE TO REPRESENT
CONTROL STRUCTURES**

by

ROZITA KADAR

**Thesis submitted in fulfillment of the requirements
for the Degree of
Master of Science**

MARCH 2011

ACKNOWLEDGEMENT

In the name of Allah, (Al-Mighty) The Gracious, The most Merciful.

Alhamdulillah, my utmost thanks to Allah for giving me strength that allows me to complete this research. I am highly grateful to my supervisor, Dr. Shahida Sulaiman whose dedication, guidance, advice, idea and moral support have tremendously aided me in this study. The research experience that I have gained is something that I will truly treasure. I would like to thank Universiti Teknologi MARA (UiTM) for the opportunity to pursue my postgraduate studies, for the generous financial assistance through SLAB (July 2007 – July 2009). Many thanks also to the School of Computer Sciences and the Institute of Postgraduate Studies, Universiti Sains Malaysia (USM) for various facilities and kind assistance provided throughout the process. Special thanks to the Director of Universiti Teknologi MARA Pulau Pinang, Associate Professor Mohd Zaki bin Abdullah, the Coordinator of the Department of Computer and Mathematical Sciences Ms. Shakirah binti Mohd Abdul Rahman and the ex-coordinator, Ms. Tengku Muhaini binti Tuan Mat for their understanding and support. My appreciation also goes to all my colleagues, especially Ms. Suzana binti Ab. Rahim, Ms. Natasha binti Nordin, Ms. Shamsunarnie binti Mohamed Zukri and Ms. Siti Nurleena binti Abu Mansor for all their ideas and supports. Moreover, my special thank goes to my husband Mohd Zamri Udin and my beloved kids Luqman, Rijal, Najwa, Harith, and Raudhah for the advice and moral support in my journey of finishing this research. Last but not least, I would also like to thank my whole family for their prayers, love, care and support.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENT	ii
TABLE OF CONTENTS	iii
LIST OF TABLES	viii
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xii
ABSTRAK	xiii
ABSTRACT	xiv
CHAPTER 1 : INTRODUCTION	
1.1 Overview	1
1.2 Background of the Study	3
1.2.1 The Importance of Program Comprehension	4
1.2.2 The Importance of Program Visualization	5
1.3 Research Framework	6
1.4 Research Questions	8
1.5 Objectives of the Study	10

1.6	Scope of the Research	11
1.7	Research Contribution	12
1.8	Organization of the Chapters	13
1.9	Summary	15
CHAPTER 2 : LITERATURE REVIEW		
2.1	Introduction	16
2.2	Program Comprehension	16
2.3	Cognitive Models of Program Comprehension Strategies	18
2.4	Program Visualization	21
	2.4.1 Graphical Representation Techniques	22
2.5	Other Visualization Techniques	26
	2.5.1 Synthetic Personality Inventory (SPI)	26
	2.5.2 Multi-Agent Educational System	26
	2.5.3 Program Dependence and Slices	27
2.6	Program Visualization Tools	28
	2.6.1 jGRASP	28
	2.6.2 BRICS	30
	2.6.3 P-Coder	31
2.7	User Interface Design	33
	2.7.1 Components of Graphical User Interface and Design Principles	33
	2.7.2 The Elements of User Interface	35
	2.7.2(a) Colour	35
	2.7.2(b) Text	36
	2.7.2(c) Graphic	37
	2.7.2(d) Layout	38
2.8	Summary	39

CHAPTER 3 : RESEARCH METHODOLOGY

3.1	Introduction	40
3.2	Operational Framework or Research Procedure	40
3.2.1	Tripp and Bichelmayer's Rapid Prototype Model	41
3.3	Method for Developing Prototype Tool	42
3.3.1	Phase I : Gather Prototype Tool Requirements	42
3.3.1(a)	Hardware and Software	43
3.3.1(b)	System Specification	43
3.3.1(c)	Prepare and Analyze Contents for Control Structures	44
3.3.1(d)	Set Objective for Prototype Tool	45
3.3.2	Phase II : Construct Prototype Tool	45
3.3.3	Phase III : Evaluate Prototype Tool	45
3.3.3(a)	Research Design and Hypothesis	46
3.4	Summary	48

CHAPTER 4 : ZOOM VISUAL FLOW (ZViF) TECHNIQUE

4.1	Introduction	49
4.2	Overview	49
4.3	Zoom Visual Flow (ZViF)	50
4.4	Bottom-up and Top-down Strategy	51
4.4.1	AICoS Notation	51
4.4.1(a)	Colours	52
4.4.1(b)	Text and Information Display	53
4.4.2	VCoF Flow Diagram	55
4.4.3	Interface Design	61
4.4.4	Flowchart	63
4.4.4(a)	Enter Code Flowchart	63
4.4.4(b)	ZViF Generated Flowchart	64

4.5	Detail Design	65
4.5.1	Sequential Control Flow	65
4.5.2	Selection Control Flow	67
4.5.3	Iteration Control Flow	69
4.6	The Criteria Design	71
4.7	Summary	72
CHAPTER 5 : EVALUATION		
5.1	Introduction	73
5.2	Evaluation Procedure	74
5.2.1	Study 1 – Users’ Preference	75
5.2.1(a)	The Questionnaire Design	77
5.2.1(b)	Analysis and Findings	77
5.2.2	Study 2 – Users’ Comprehensibility	86
5.2.2(a)	Questionnaire Design	87
5.2.2(b)	Analysis and Findings	88
5.2.3	Study 3 - The Comparative Study on Graphical Representation Techniques	96
5.2.3(a)	The Questionnaire Design	97
5.2.3(b)	Result and Discussion	99
5.3	Summary	101
CHAPTER 6 : CONCLUSION		
6.1	Summary of the Thesis	102
6.2	Contribution	105
6.3	Revisiting of the Objectives	107
6.4	Future Research	108
6.5	Summary	109

REFERENCES	110
APPENDICES	116
Appendix A: Manual for ZViF Tool	117
Appendix B: Program Use in Study I	123
Appendix C: Program Use in Study II	125
Appendix D: Questionnaire for Study I	127
Appendix E: Questionnaire for Study II	129
Appendix F: Certificate of Appreciation for Winning the Best Paper Award	134
Appendix G: Certificate of Appreciation for Winning the Bronze (Professional)	135
Appendix H: List of Publications	136

LIST OF TABLES

	Page
Table 2.1: The Strengths and Weaknesses of Graphical Representation Techniques	25
Table 2.2: Combination of Colours for User Interfaces (Brown and Cunningham, 1993)	36
Table 4.1: Action Icons Control Structures (AICoS) Notations	54
Table 4.2: Criteria Used in Designing the Technique (Baecker, 1988)	71
Table 5.1: The Number of Respondents and Their CGPA	76
Table 5.2 Performance Characteristics Items and Descriptions (Hendrix <i>et al.</i> , 1998)	78
Table 5.3: PCH Item Response Frequencies (the Number of Students Who Responded to a Particular Treatment)	79
Table 5.4: The Mean and p -value of the Responses Towards Each PCH	81
Table 5.5: The Number of Respondents and Their CGPA	87
Table 5.6: Summary of Findings	90
Table 5.7: The Mean and p -value for 14 Questions Between ZViF and jGRASP	95
Table 5.8: The Mean and p -value for 14 Questions Between ZViF and Text Based	95

Table 5.9: The Existing Graphical Approach and ZViF Approach Styles	
(a) ANSI Flowchart, (b) Nassi-Shneiderman Diagram,	
(c) Warnier-Orr Diagram, (d) Action Diagram, (e) CSD	
and (f) ZViF	98
Table 5.10: Comparative Study among Graphical Representation Techniques	99
Table 6.1: ZViF Technique Compared to Other Existing Techniques	106

LIST OF FIGURES

	Page
Figure 1.1: Frameworks Semantic Software Engineering (Semanticsoftware.info, 2009)	7
Figure 2.1: The Control Structure Diagram (CSD) Presented by jGRASP (jGRASP, 2010)	29
Figure 2.2: Viewer for BRICS, Showing the Source Code and Overview Windows (Pearson <i>et al.</i> 2008)	30
Figure 2.3: The Main Window of P-Coder (PCoder, 2010)	32
Figure 3.1: Flowchart of Operational Framework for Research	41
Figure 3.2: The Relationships of All Variables and Their Attributes	46
Figure 4.1: The Sequential Flow Diagram	56
Figure 4.2: The “If” Flow Diagram	57
Figure 4.3: The “Switch” Flow Diagram	58
Figure 4.4: The “While” Flow Diagram	59
Figure 4.5: The “For” Flow Diagram	60
Figure 4.6: The “Do” Flow Diagram	61
Figure 4.7: Flowchart to Enter the Source Code	63
Figure 4.8: Flowchart to Generate the ZViF	64
Figure 4.9: The Sequential Statement View in ZViF Tool	65
Figure 4.10: The ‘If’ Selection Statement View in ZViF Tool	67

Figure 4.11: The ‘While’ Loop Statement View in ZViF Tool	69
Figure 5.1: The Mean of the Responses Towards Each PCH	80
Figure 5.2: The Sequential Flow Diagram a) ZViF b) CSD	82
Figure 5.3: The Selection Flow Diagram a) ZViF b) CSD	84
Figure 5.4: The Iteration Flow Diagram a) ZViF b) CSD	85
Figure 5.5: Total of Correct Answers for Each Questions	89
Figure 5.6: “If” Selection Generated by a) jGRASP b)Visual C++ IDE c)ZViF	91
Figure 5.7: “Do” Loop Generated by a) ZViF b)Visual C++ IDE c) jGRASP	93
Figure 5.8: “While” Loop Generated by a) ZViF b) jGRASP c) Visual C++ IDE	94

LIST OF ABBREVIATIONS

AICoS	-	Action Icons Control Structures
BRICS	-	Blocks of Rationally Intuitive Control Structures
CGPA	-	Cumulative Grade Point Average
CSD	-	Control Structured Diagrams
GUI	-	Graphical User Interface
IDE	-	Integrated Development Environment
jGRASP	-	Graphical Representations of Algorithms, Structures, and Processes for Java
P-Coder	-	Pseudo Coder
PCH	-	Performance Characteristics
VCoF	-	Visual Control Flow
ZViF	-	Zoom Visual Flow

**MENAMBAHBAIK PEMAHAMAN ALIRAN KAWALAN
MENGUNAKAN TEKNIK ALIRAN VISUAL BERFOKUS
(ZViF) UNTUK MEWAKILI STRUKTUR KAWALAN**

ABSTRAK

Apabila pengguna awal mempelajari pengaturcaraan, mereka harus memahami banyak perkara berkaitan pengaturcaraan. Berbagai-bagai teknik dan alatan telah dibangunkan untuk membantu pengguna ini dalam meningkatkan pemahaman pengaturcaraan tetapi kebanyakan alatan tidak sesuai untuk mereka. Beberapa alatan tidak bersifat mesra pengguna, rekabentuk hanya ditumpukan kepada pengguna mahir dan beberapa Persekitaran Pembangunan Bersepadu (IDE) sangat ringkas dan gagal untuk digunakan dalam persekitaran dunia nyata. Ini akan melambatkan proses pembelajaran dan menimbulkan kesukaran bagi pengguna yang tidak mempunyai latar belakang pengaturcaraan. Skop kajian ini merangkumi teknik visualisasi yang mewakili struktur kawalan untuk pengguna awal. Matlamat utama kajian ini adalah untuk memberi pendedahan tentang bagaimana meningkatkan kaedah persembahan visual dalam pengeditan aturcara atau IDE. Kajian ini cuba meningkatkan pemahaman aliran kawalan dengan menggunakan teknik Aliran Visual Berfokus (ZViF) yang mewakili kod sumber dalam paparan grafik. Sebanyak dua eksperimen dan satu kajian perbandingan dilakukan untuk menentukan keberkesanan teknik. Keputusan menunjukkan bahawa pengguna lebih memilih teknik yang dicadangkan dalam membantu meningkatkan pemahaman aliran kawalan dalam kalangan pengguna awal jauh lebih baik dibandingkan dengan teknik Gambarajah Kawalan Struktur (CSD) dan IDE berasaskan teks.

ENHANCING CONTROL FLOW COMPREHENSION USING ZOOM VISUAL FLOW (ZViF) TECHNIQUE TO REPRESENT CONTROL STRUCTURES

ABSTRACT

When novice users learn programming, they have to comprehend a lot of things related to programming. Many techniques and tools have been developed to help users to improve their program comprehension but most tools are unsuitable for novices. Some tools are not user friendly, some designs are just for expert programmers and some Integrated Development Environment (IDE) are very simple and fail to expose users to the real world environment. These hinder the learning process and may become obstacles to users who have no programming background. The scope of the study is on visualization technique to represent control structures for novice users. The main goal of this work is to give some insights on how to improve visual presentation method in program editor or an IDE. This study attempts to improve control flow comprehension by using Zoom Visual Flow (ZViF) technique that represents source code in graphical view. Two lab experiments and a comparative study were conducted to determine the effectiveness of the technique. The result shows that users prefer the proposed technique that helps to improve control flow comprehension among novices much better than Control Structured Diagrams (CSD) technique and text-based IDE.

CHAPTER 1

INTRODUCTION

1.1 Overview

Software engineering is an engineering discipline that concerns with all aspects of software production from the early phases of software specification to software maintenance (Sommerville, 2001). Its goal is to develop software that satisfies and possibly exceeds the users' expectation. In the original model of software lifecycle (Royce, 1970), there are five phases to be followed, which are (a) requirement analysis and definition, (b) system and software design, (c) implementation and unit testing, (d) integration and system task, and (e) operation and maintenance.

Program comprehension is a major activity during software maintenance (Maletic & Kagdi, 2008). Program comprehension is defined as a process whereby programmers will understand a software artefact using both knowledge of the domain and/or semantic and syntactic knowledge to build a mental model of its relation to the situation. According to O'Brien (2003), this activity is required when maintaining, reusing, migrating, reengineering or enhancing the software system. One of the activities to improve program comprehension is to enhance the **Integrated Development Environment (IDE)** that provides a **Graphical User Interface (GUI)**.

IDE normally consists of a source code editor, compiler and/or interpreter, build-automation tools and a debugger. GUI is a special screen image-based computer system that allows software commands to be issued through the use of graphic symbols to support the process of writing software. The system includes (a) a syntax-directed editor, (b) program entry graphical tools, (c) integrated support for compiling and running the program; and (d) relation of compilation errors back to the source (Timoty & Linda, 2005). GUI is a particular case of user interface in interacting with a computer by employing graphical images and widgets together with text to represent the information and actions available for users. Usually the actions are performed through direct manipulation of the graphical elements. In computer technology, graphical representation used to improve program comprehension is discussed within program visualization.

Program visualization uses the capability of human visual system to enhance program comprehensibility. The purpose of program visualization is to translate a program into a graphical view to show either the program code, data or control flow (Briand *et al.*, 1997). This technique is significant to users because the criteria of source code cannot be physically viewed. Human interpretation or imagination is needed to help the users to understand the source code. Visualization techniques can be used in teaching to help users understand on how programs work. It is applicable in the process of writing programs because it helps them to understand their codes better. This study uses visualization techniques to show the flow of control structures that consist of sequential, iteration and repetition. Control structures visualize its control flow to show the flow of a program. Thus, visualization technique can help users to understand such programs better mainly among the novices.

1.2 Background of the Study

In studying about program languages, users need to be able to comprehend a program that is completed with syntax, semantic and flow of a program. Most users especially the novices face a lot of problems when trying to learn a program. According to Winslow (1996), many novice programmers are unable to transform the problem solution into source code. They have to take a lot of time to understand the syntax, semantics and the program flow. They need techniques and tools to help them in the learning process.

Presently, there are many tools available to improve program comprehension but not all are suitable for different level of users. Some of the IDEs designed with advance features are suitable for professional users only (Eclipse, 2010; Zhou, 2008; JAVA, 2001). These features may overwhelm some novices especially those who have no programming background. Besides, some IDEs are too simple and fail to expose users to the real world environment (Maletic, 2008; Vainio, 2007; Lahtinen, 2007). The unwell-structured tools are caused by a lack of knowledge about the effectiveness of the IDE design (Sommerville, 2001). These tools lack of user friendliness. Hence, programs become more difficult to understand with a high possibility that the novice programmers will be neglected due to the complex features in IDE.

User interface layout should be good enough for the purpose of facilitating the process of learning programs effectively and efficiently. The user interface plays an important role to help users to visually understand the problem solving strategies (Chen & Marx, 2005). A good IDE helps software developers in writing programs more

quickly and produce better quality code. IDEs with standard features enable users to familiarize better and reduce their time to learn the features.

This research discusses the visualization technique that will represent the source code in a graphical view to help novices to improve their control flow comprehension. According to Hendrix *et al.* (2002), representing any ideas with pictures rather than words is intuitively more appealing because a visual presentation will be more readily understood than its textual counterpart. This research aims to support beginners or novice programmers who have been exposed to programming languages by providing effective visualization technique and tool. Thus, the control flows and the source code structures will be shown visually. Therefore, a more suitable tool providing IDE elements that includes an effective technique is proposed in the production of a better IDE and well-designed GUI to improve learning process especially in program comprehension.

1.2.1 The Importance of Program Comprehension

There have been a large numbers of researches directed at the problem of program comprehension (Maletic & Kagdi, 2008). Zaidman *et al.* (2006) find that software engineers tend to spend up to 50% of their time trying to comprehend the structure of a software system. Many problems in program comprehension arise due to the use of textual representation as the primary source of information. In fact, a program is in the form of a hierarchical structure, but the actual behaviour of a program cannot be reflected as the program is represented in text form. A program can be understood if users manage to comprehend the flow of a program including its syntaxes and

semantics. The program comprehension activity is more difficult when users try to understand programs that are written by others.

Program comprehension is important in order to understand the problem domain written for a specific program. It also builds a mental representation of the program that is often seen as a hypothesis-driven process (Vainio and Sajaniemi, 2007). Moreover, the study of program comprehension can help to explain how programmers understand a program or software. The combination of theories and tools will help a programmer understands the codes or programs better (O'Brien, 2003).

1.2.2 The Importance of Program Visualization

The subject that is not physical in nature or hidden from view needs to be interpreted for its comprehension. Therefore, program visualization that concedes the process of making intangible things physically visible helps to generate better information for humans. Brusilovsky (2006) states that visualization can provide a clear metaphor for understanding complicated concepts and uncovering the dynamics of processes that are usually hidden from the users' vision. The purpose of program visualization is to extract information from a program and present it in a graphical form (Roman & Cox, 1992). Thus, program visualization uses graphics to enhance the art of program presentation and thereby facilitates the visualization, understanding and effective use of computer program for computer users. According to Lahtinen *et al.* (2007), program visualization is typically used in introductory programming courses because visualization can help students with difficulties in learning.

A suitable visual representation must be considered carefully to ensure the effectiveness of visualization viewed. Program visualization is important to comprehend a program because program is text-based and is not shown physically. Moreover, program visualization helps the users to understand the behaviours of a program due to difficulties that may appear when users try to understand the program. Additionally, it needs a technique that can represent the text-based information into graphic illustration so that users can increase their understanding. According to Hendrix *et al.* (2000), representing objects, process and ideas with pictures rather than words are intuitively more appealing.

In the midst of today's technologies and capabilities of computer graphics, many software programs become easier to use because of the availability of a graphical interface. This technology, especially graphical representation, contributes tremendously in the learning process because graphic-based program assists the understanding of the process better compared to text-based only. Thus, this study proposes a technique that represents text-based programming into graphical view.

1.3 Research Framework

In maintaining the software, more time is spent in comprehending the source code. Thus, in order to reduce the time used, this research refers to Semantic Software Engineering that encompasses theoretical aspects of the systematic design as well as the implementation and deployment of knowledge-oriented software systems (Semanticsoftware.info, 2009). There are four major areas that are discussed in this field: applications, system architectures, foundation, and forward and reverse engineering (see Figure 1.1).

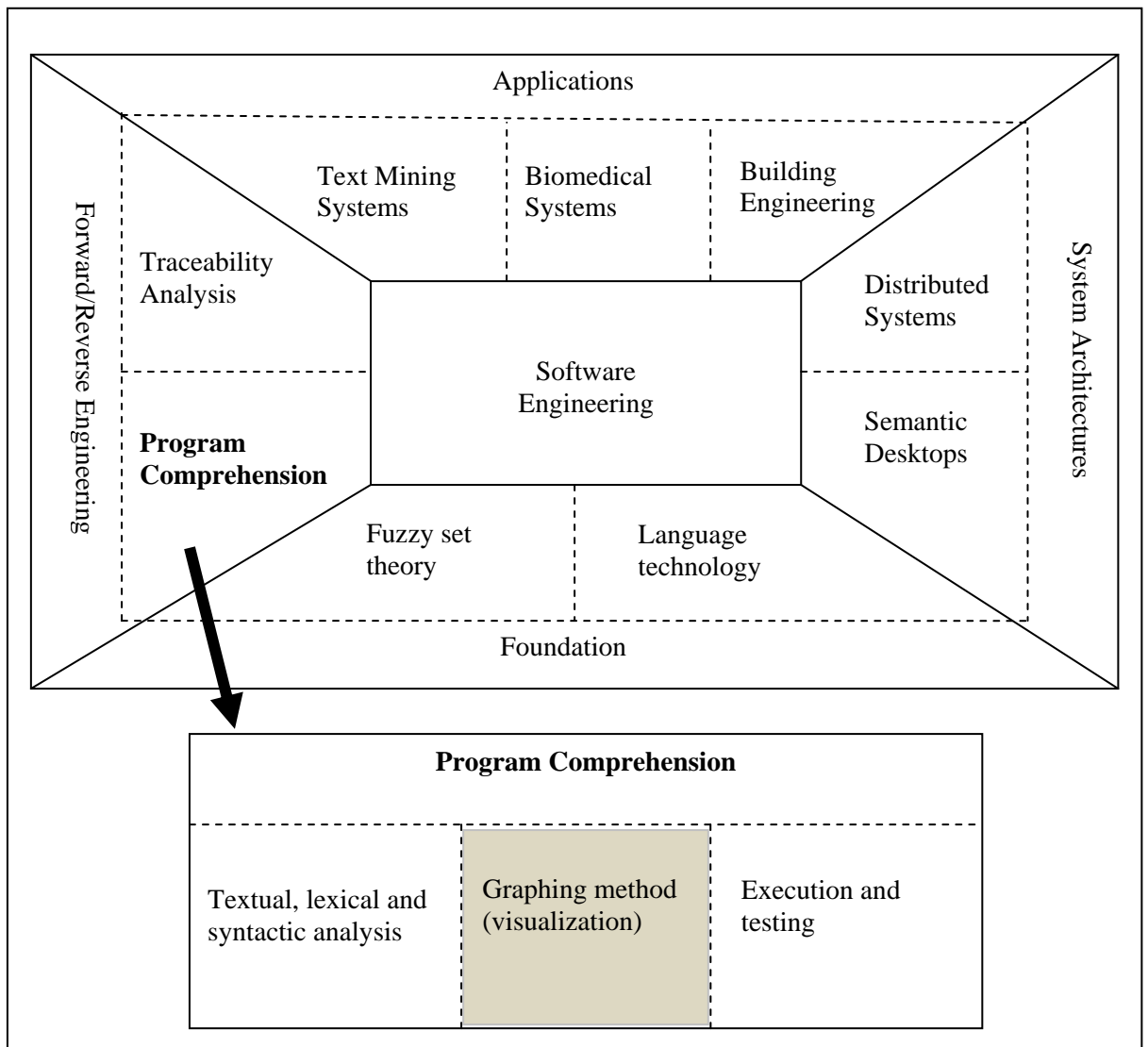


Figure 1.1: Frameworks Semantic Software Engineering (Semanticsoftware.info, 2009)

In this study, program comprehension is focused on the reverse engineering. The research area of this study discusses on how to develop tools and methodologies to assist in understanding the source code.

Nelson (1996) discusses on the specialization area in program comprehension and proposes the major approaches to improve program comprehension as described below:

- (i) Textual, lexical and syntactic analysis: These approaches focus on the source code and their representations.
- (ii) Graphing methods (visualization): There are a variety of graphing approaches for program comprehension to show the control flow of the program, data flow and data dependency. This approach is the mapping of program to the graphical view to show the programs visually.
- (iii) Executing and testing: These approaches are for profiling, testing and observing program behaviour.

The technique proposed in this study implements the graphing method. Most of the researchers discuss the graphing method under program visualization area.

1.4 Research Questions

For this research, the main question focuses on *How to provide an effective technique to visualize control structures that can improve novices' control flow comprehension?* The sub-questions are as follows:

- (i) **What are the suitable techniques for novices?** There are a lot of techniques that are suggested in improving control flow comprehension. Most of the techniques are not suitable to novices. This study compares the existing techniques concerning their strengths and weaknesses.
- (ii) **Are the existing tools effective?** Well-designed tools can avoid users from learning complex programming languages and help them to reduce the time in learning process. However, most of the tools are not suitable for novices. This

research studies the characteristics of the tools and techniques available besides comparing the strengths and weaknesses of the tools.

- (iii) **How to visualize control structures?** This study chooses the control structures program to test the technique. Visualization is the best technique to understand the control structures rather than text-based because users need to understand the flow of controls by looking at how each structure is operated as it can be illustrated. So, this study aims to find out the effective way to visualize the flow of the control structures.
- (iv) **What are the important features that should be provided and how do the features work?** One of the objectives of study is to develop the prototype tool that is applicable to the technique. This study has to find the features that novices are familiar with. The intended features are simple, easy and user friendly.
- (v) **How to ensure the technique is effective?** The evaluation of the prototype tool will be conducted. The result from the collected data answers the hypotheses to ensure the effectiveness of the technique that is applied on the tool.

In order to identify the effectiveness of the tool, it is crucial to discuss these four main arguments to produce the most effective tool that can support users to overcome any problems. The four arguments are as listed below:

- (i) **What should the tool support?** The tool should be suitable for novices. It must be easy and simple to use. The tool should focus on visualizing a program rather than representing a program in text-based in understanding the behaviour of a program.

- (ii) **Why should the tool support?** Without the capability to visualize the flow of a program, most novices will face many problems in program comprehension. The tool should attempt to visualize the flow of a program in graphical view.
- (iii) **When should the tool support?** The tool should be provided to the beginners or novices at the start of learning process. At present, a program can only be found in text-based software and they need an aid tool to comprehend a program in other methods.
- (iv) **How should the tool support?** The tool should help novices to comprehend a control flow in other methods rather than text-based. The graphical view of source code is more effective to help novice users in understanding and comprehending a program.

ZViF tool applied in this research implements ZViF techniques designed to fulfil users' needs. All features provided are in standard design and simple layout to overcome the lack in using existing tools. Besides, the tool is mainly built for novices.

1.5 Objectives of the Study

This research is important in order to help novices to improve their control flow comprehension in their learning process. Besides, this study can be seen as a form of guideline for designers or anyone who is interested in developing tools that focus on improving control flow comprehension. This study can contribute as an introduction in learning programming language for the novices. Therefore, the research objectives are as follows:

- (i) To enhance the visualization technique in order to improve understanding of control flow in program comprehension activity among novices;

- (ii) To produce effective tool that can be used as an aid among novices when they learn control structures of a programming language; and
- (iii) To measure the significance of the proposed technique and its tool in improving control flow comprehension if it is used among novices.

1.6 Scope of the Research

The scopes of the study are as follows:

- (i) **Visualization technique:** Visualization technique can be used to visualize the program code, data or control flow. The visualization technique proposed only covers the scope of the control flow. The technique visualizes the control flow of control structures by determining the sequence of statements.
- (ii) **Control structures:** The technique only chooses control structures to visualize the control flow. The control structure also known as control construct consists of three types: sequences, selections and iterations. The flow of control structures is difficult to determine because it has some conditions that need to be applied to control the flow of statements. This study chooses C++ programming language to apply the proposed technique.
- (iii) **Novices:** The people who can be categorized as novices are those with non-programming background or beginners in learning a program. When novices learn a programming language, they face a lot of problems. This study focuses on this group of people to help them in their learning process.

1.7 Research Contribution

When novices start to learn programs, they face a lot of problems in trying to understand the program code, its operation and its flow. Although many researchers had worked in finding different strategies and techniques to overcome these problems but most researchers still have yet to discuss in great length on how to help users when they learn a certain program.

Thus, the technique called **Zoom Visual Flow (ZViF)** is proposed to visualize the control structures. The ZViF technique uses diagram with the combination of text and color to show programs in visual manner. Program flows are presented by using **Action Icons Control Structures (AICoS)** and **Visual Control Flow (VCoF)**. The details of the technique are discussed in **Chapter 4**.

This research provides the graphical representation technique to comprehend a control flow of control structures because it is the best way to explain certain things rather than using text-based. This technique helps the users to understand a control flow by translating it into graphical view to show the program code, data and control flow. When the text-based method is translated and displayed graphically with the combination of text, it becomes more translucent for novices to understand information presented.

This study also develops a tool that implements the proposed technique. In designing the tool, the main criterion observed is the functions which are suitable for novices. The diagrams with the combination of text are used to represent each control structure in simple form. A different form of diagram represents a different task. It eases

the users' memories in using each of the diagrams. The diagrams intend to portray the semantic feature while the text provides the way to clarify the meaning to the users. The tool offers capabilities in highly visual and readable description of a program. The tool that implements this technique is intended for teaching purpose.

The technique is produced in an effort to provide a program visualization tool that addresses the problems faced by novices when learning a program related to cognitive model. This study hopes to provide valuable input to lecturers, curriculum planners and researchers on the aspects of designing and developing tools for program comprehension as well as to improve teaching and learning processes. Most importantly, it provides useful new knowledge to novices about the way to comprehend a control flow.

1.8 Organization of the Chapters

This thesis is organized into six chapters. Chapter 1 to Chapter 6 present in chronological order, an introduction, literature review, research methodology, the development of ZViF, evaluation and discussion respectively. The organization of the chapters of the thesis is as follows:

Chapter 1 introduces the overview of software engineering and program comprehension that is one of the core activities in software engineering through enhancing a GUI in IDE. In addition, this chapter also discusses on the program visualization that is used to translate a program into graphical view. It also describes the background of the study followed by the importance of program comprehension and program visualization. Research framework for program comprehension and the

research questions as well as sub-questions are designed. Objectives of the study are mentioned which is later followed by the research contribution list.

Literature review that is related to the study is discussed in **Chapter 2**. In this chapter, previous studies regarding program comprehension and program visualization are discussed in details by providing the ideas, strategies, techniques, and tools proposed by previous researchers. This chapter also discusses on the guidelines, characteristics and principles observed to produce a better user interface.

Chapter 3 is research methodology that discusses in details about the procedures taken since the beginning until the end of the conducted research. In the first section, the operational framework or research procedure conducted in this study is illustrated and the reasons for choosing Tripp and Bichelmayer's Rapid Prototype Model in this study are listed. Next, the methods in developing prototype tool are discussed in details. There are three phases in developing the prototype, which are gather prototype tool requirements, construct prototype tool and evaluate prototype tool. In the evaluation section, the variables, attributes and hypotheses are determined.

The development of ZViF prototype tool is described in details in **Chapter 4**. In this section, AICoS notations and VCoF flow diagrams used in the tool are discussed. In the next section, more discussion on the algorithm to display the flow of programs is presented. In the last part of the chapter, the source code that is written in Java language involves in developing ZViF tool is discussed.

Chapter 5 presents in details the procedure to evaluate the proposed technique and result of the research. The evaluation procedure includes questionnaire design, data collection procedure, data analysis and its findings. In evaluation, there are two types of lab experiment studies: users' preference and users' comprehensibility. The comparative study on graphical representation techniques is also done to compare the existing techniques and the proposed technique. The result and discussion are made at the end of each study.

Chapter 6 is the last chapter that concludes the research by drawing the summary of the thesis. The contributions of the study are determined based on the findings. It also provides suggestions for further research work, which are stated at the end of the chapter.

1.9 Summary

Specifically, this chapter provides an overview on the problem of learning a program especially among novices. The background of the study was also discussed together with the research questions. This chapter also mentions the objectives that need to be achieved in this study. The research's framework and contributions have also been listed. Finally, the organizations of the chapters in this thesis are presented briefly so as to show an overview of respective chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Various researches have been conducted in program comprehension over the last few decades (De'tienne, 1997; De'tienne *et al.*, 2001; Storey, 2005). Many researchers suggest ideas, techniques and tools, which can help the users to comprehend a program. Firstly, this chapter focuses on the program comprehension in general followed by program comprehension strategies that are proposed by some researchers. Program visualization is one of the techniques that give a lot of contributions in learning a program. The roles and importance of program visualization are discussed. Next, the existing graphical representation techniques and the existing tools are also discussed. Besides, this chapter also highlights why IDE plays an important role in the learning process especially to improve program comprehension. At the end of this chapter, several sections on finding a better way on how to design and develop a tool to provide the most effective IDE are discussed.

2.2 Program Comprehension

One of the software engineering activities involves program comprehension. In the original model of software engineering lifecycle, there are five phases to be followed: requirement analysis and definition, system and software design,

implementation and unit testing, integration and system task as well as operation and maintenance. Program comprehension is the major activity during software maintenance (Maletic & Kagdi, 2008). Program comprehension is important because it can explain how programmers understand a program or software. The process of comprehending a program becomes more complex and it has been recognized as a major time-consuming process in software maintenance (Storey *et al.*, 1999) taking up to 60% of the total time devoted to maintenance (Dunsmore *et al.*, 2000).

Corritore and Wiedenbeck (2001) state that program comprehension concerns on the individual programmer's understanding of "what a program does and how it does it in order to make functional modifications and extensions to a program without introducing errors". According to Von Mayrhauser (1995), program comprehension is "an activity in which the program reader extracts meaning by understanding how a particular program or code fragment performs its task, or what task a particular item performs". Deimel and Naveda (1990) define program comprehension as "the process of taking source code and understanding it".

Program comprehension is a combination of two characteristics: The theories that provide how to improve program comprehension and tools that can implement the theories. These two characteristics will change the way programmers understand the codes or programs. Storey (2005) reviews some of the key theories of program comprehension and discusses on how these theories are related to tools that support it. Many researchers also consider two models when they study program comprehension, which are mental model and cognitive model. A mental model describes the constructed combination of information contained in the source code and documentation with the

assistance of experts and domain knowledge that the programmer brings into the task (Grubb & Takang, 2003). In other words, mental model shows the maintainer's mental representation of the program that needs to be understood. A cognitive model describes the processes and information structures used to form the mental model (O'Brien, 2003). De'tienne (2001) also reviews cognitive models and conducts the experiment in this area.

Although much research has been done, Corritore and Wiedenbeck (2001) point out that the studying of program comprehension remains incomplete and it should be continued in order to produce the best strategies to improve program comprehension.

2.3 Cognitive Models of Program Comprehension Strategies

Many studies have been conducted to observe the process on how programmers understand the code. Finally, they propose five cognitive models of program comprehension strategies: bottom-up (Shneiderman & Mayer, 1979), top-down (Brooks, 1983), integrated approach (Von Mayrhauser & Vans, 1985), knowledge-base (Letovsky, 1986) as well systematic and as-needed (Littman *et al.*, 1986).

Shneiderman and Mayer (1979) suggest that some programs are understood from bottom-up comprehension strategy where programmers read the source code by constructing a multilevel internal semantic structure to present the program. Low-level software artefacts are mentally chunked or the lines of code are grouped into meaningful high-level abstraction. Chunking is the process of recognizing the function of program components and fragments. These pieces are then grouped until

understanding is formed. This strategy can help to improve program comprehension, especially to novices because users can focus on smaller programs.

The work by Pennington (1987) observes how programmers understand a program by using the bottom-up strategy, which focuses on gathering statements and controlling flow information. It believes that understanding the overall control flow is more important than understanding the function of programs. This strategy produces at least two mental models, which are program model and domain model. The micro-structure will be chunked and cross-referenced by macro structure to form a program model. The domain model relates objects and functions in the problem domain to language entity sources. According to O'Brien (2003), the bottom-up model of program comprehension primarily addresses situations where the programmer is unfamiliar with the domain. Comprehending program by using bottom-up strategy needs a mental model and cognitive model of a program. The process of chunking the source code will be based on the program domain. However, this strategy is not applicable to novices because they do not have the capability to determine the program domain.

The top-down strategy is the understanding by comprehending the top-level detail program such as what it does and when it executes. It also includes the understanding of low-level details such as data type, control and data flow, and arithmetic patterns. Brooks (1983) proposes the top-down strategies in which the programmer develops a hierarchy of hypotheses on what the program does and how the program works. The verified hypotheses depend heavily on the presence and absence of beacons, where indicators present a particular structure or operation of the internal and external program. According to Soloway and Ehrlich (1984), a top-down strategy is

used when the syntax of the program is familiar to the programmer. They also observe how expert programmers recognize program plans and exploit programming conventions during comprehension. In this strategy, they determine the hypotheses to know the program domain. Users have to select the beacons based on their knowledge foundation, mental model and external representation. Thus novices must have the ability to select the beacons and hypotheses.

Letovsky (1986) studies programmers who use either bottom-up or top-down strategy to comprehend a program that is called knowledge-base strategy. The work mentions that program understanding depends on the programmer's knowledge foundation and the assimilation process involving both top-down and bottom-up strategies. Other strategies include the integration of bottom-up, top-down and knowledge-base called systematic and as-needed which is theorized by Littman *et al.* (1986). Von Mayrhauser and Vans (1985) suggest the integrated approached strategy to improve program comprehension.

Most novices face difficulty in determining the flow of a program (Pennington, 1987), which causes them to fail in understanding what happens inside a program (Brusilovsky *et al.*, 2006). By using the combination of bottom-up, top-down and external representation, this study attempts to reduce problems in comprehending a control flow of a program. Bottom-up strategy can be used to determine the flow of a program and top-down strategy can be used to recognize the function and process of the program. This study utilizes the combination of these strategies to visualize a control flow of control structures from the source code. The strategy that has been used for this study is discussed in **Chapter 4**.

2.4 Program Visualization

Software visualization is divided into two groups; algorithm visualization which is used to study abstract algorithm and program visualization which visualizes on source code or data structure (Yehezkel, 2002). This research focuses on program visualization to improve understanding of the program. Baecker (1988) states that to visualize means “to see the mental image form of something”. According to the study, graphics are used as an art to enhance program comprehension because graphics encompass the disciplines of typography, graphic design, animation, cinematography and the technology of interactive computer graphics. Based on Myers (1986), program visualization is the program that is specified in the conventional textual manner where the graphics are used to illustrate some aspects of the program or its run-time execution.

According to Roman and Cox (1992), program visualization is a field of study that concerns with the use of graphical representations in the computer environment and deals with graphical presentations, supervision and exploration of program expressed in textual form. According to the survey that is done by Brusilovsky *et al.* (2006), the majority of respondents (89%) felt that enhancing graphical visualization with textual visualization will help to improve the value of visualization.

Program visualization is suitable to be used in the introductory program courses because visualization implements pictures or notations. It is suitable for novice users (Lahtinen *et al.*, 2007) but the usability and instruction should be planned more carefully to make it more beneficial for the users (Hendrix *et al.*, 2000; Lahtinen *et al.* 2007).

According to Krinke (2004) there are many program visualization tools that have been developed to support teaching and learning of programming but most of them have negative feedback. The reasons for this finding may be derived from the facts which the tools and visualizations are constructed in a uniform fashion and the visualization systems do not allow for enough interaction between users and the system (Ohki and Hosaka, 2003; Bednarik *et al.*, 2005; Pearson *et al.*, 2008; Raeder, 1985).

2.4.1 Graphical Representation Techniques

A variety of graphical pseudo code forms have been proposed; for examples: ANSI Flowchart (Nassi & Shneiderman, 1973), Nassi-Shneiderman Diagram (Cornelia & Marilyn, 1973), Warnier-Orr Diagram (Martin & McClure, 1984), Action Diagram (Martin & McClure, 1985) and Control Structured Diagram (jGRASP, 2009). All these techniques use symbols to represent most operations in programming. The goal of each technique is essentially the same, which is to provide a clear picture of the structure and semantics of the program through a combination of graphical construction and some additional textual notations.

Flow Chart technique (Nassi & Shneiderman, 1973) clearly describes the flow of action that can describe the program abstraction between program statement and the code of the completed program. Nassi-Shneiderman Diagram (Cornelia & Marilyn, 1973) has proven to be useful in all phases in programming development because of the excellent graphic technique, which are simple to use, elegant in appearance, easier to be understood by novice users and allow the users to build their own structure.

Warnier-Orr Diagram (Martin & McClure, 1984) uses a set of brackets to show the level of the system and aid the design of program structure by identifying the input, process and output. Action Diagram (Martin & McClure, 1985) is used to construct the flow of sequence, repetition, condition, module, and data store. CSD has been developed since 1995 and the improvement of the technique and tool that apply the technique that has been done until now (Cross & Sheppard, 1988; Hendrix *et al.*, 2002; Hendrix & Cross, 1998; jGRASP, 2009; jGRASP, 2010). CSD is designed to reduce the time required for program comprehensibility by clearly depicting the control structures and control flow at all relevant levels of program abstraction.

The goal of each technique is essentially the same, which is to provide a clear picture of the structure as well as semantics of the program through a combination of graphical construction and some additional textual notations. Each style has its own strengths and weaknesses. The strengths and weaknesses of graphical representation techniques are summarized in Table 2.1. From this table, CDS is the best technique because it uses simple graphical notations and its program flow is shown clearly by using the line. Nonetheless, there are still weaknesses of this technique as it can also be found in other techniques.

Therefore, this study proposes a technique that can reduce some of the weaknesses found in the five techniques discussed. The strengths of the proposed technique are as follows:

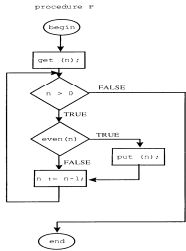
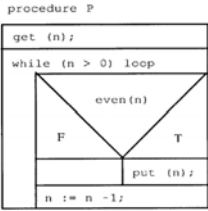
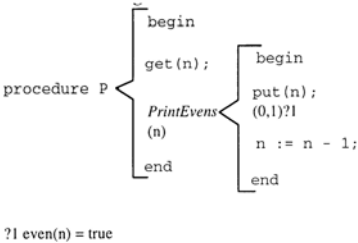
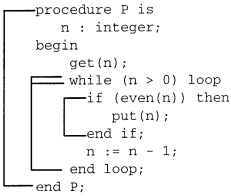
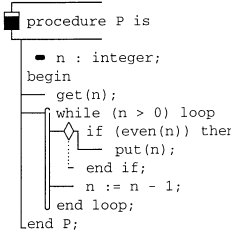
- (i) Using simple and special graphical notations to indicate the task of a program;
- (ii) Maintaining the original source code;
- (iii) Showing clear flow of action; and

(iv) Converting graphic view easily.

Meanwhile, **Chapter 5** discusses the result of comparative study among these existing techniques with the proposed technique.

Although most researchers develop with variety of techniques to improve program comprehension through program visualization, the effectiveness of visualization is still an open question and is certainly not universally accepted (Hendrix *et al.*, 2000).

Table 2.1: The Strengths and Weaknesses of Graphical Representation Techniques.

Techniques	Strengths	Weaknesses
<p>Flow Chart (Nassi & Shneiderman, 1973)</p> 	<ul style="list-style-type: none"> • Closer to structured programming. • Uses basic symbols to show the flow of a program. • Clearly describes the flow of action and level of abstraction. 	<ul style="list-style-type: none"> • Can only describe simple programming technique. • Describes more on control flow rather than program component.
<p>Nassi-Shneiderman Chart (Cornelia & Marilyn, 1973)</p> 	<ul style="list-style-type: none"> • Excellent graphic technique which is simple, elegant and easier to understand • Allow users to build their own structure • Better in displaying logic. 	<ul style="list-style-type: none"> • Does not provide automatic generation of source codes or correct errors.
<p>Warnier-Orr Diagram (Martin & McClure, 1984)</p> 	<ul style="list-style-type: none"> • Shows the flow of control construct very well. • Easy to convert into structured program code. • The appearance is simple, easy to understand and clear in showing groupings of process and data. 	<ul style="list-style-type: none"> • Does not show the sequential of the statements clearly.
<p>Action Diagram (Martin & McClure, 1985)</p> 	<ul style="list-style-type: none"> • Uses special graphical notations to indicate the task of a program. • The appearance is simple by using only lines to show the block statements. 	<ul style="list-style-type: none"> • Does not show clearly the sequence of statements.
<p>Control Structured Diagram (Cross & Sheppard, 1988; Hendrix <i>et al.</i>, 2002; Hendrix & Cross, 1998; jGRASP, 2009; jGRASP, 2010)</p> 	<ul style="list-style-type: none"> • Depicts control structures and control flow at all relevant levels of program abstraction clearly. • Uses basic symbols to show the flow of a program. 	<ul style="list-style-type: none"> • Does not clearly describe the source code. • Only visualizes the flow of a program. • Requires basic knowledge about programming languages.