

**EVALUATING TECHNIQUES TO REDUCE ERRORS MADE BY NOVICE
PROGRAMMERS**

by

Murtala Mohammed

A Thesis submitted to
The Graduate Institute of Science and Engineering

of

Meliksah University

in partial fulfillment of the requirement for the degree of
Master of Science

In

Electrical and Computer Engineering

August 2014
Kayseri, Turkey

APPROVAL PAGE

This is to certify that I have read the thesis entitled “Evaluating techniques to reduce errors made by novice programmers” by Murtala Mohammed and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Electrical and Computer Engineering, the Graduate Institute of Science and Engineering, Melikşah University.

August 20, 2014

Asst. Prof. Dr. Hasan KITAPCI
Supervisor

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

August 20, 2014

Assoc. Prof. Dr. Ihsan O Bucak
Substitute Head of Department

Examining Committee Members

Title and Name

Approved

Asst. Prof. Dr. Hasan Kitapci

August 20, 2014

Asst. Prof. Dr. Aytekin Vargun

August 20, 2014

Asst. Prof. Dr. Hasan Palta

August 20, 2014

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Science and Engineering.

Prof. Dr. M. Halidun Kelestemur
Substitute Director

EVALUATING TECHNIQUES TO REDUCE ERRORS MADE BY NOVICE PROGRAMMERS

Murtala Mohammed

M.Sc. Thesis- Electrical and Computer Engineering

August 2014

Supervisor: Asst. Prof. Dr. Hasan KITAPCI

ABSTRACT

Researches have shown that programming skills are not easily acquired. Although instructors are spending a lot of time in conveying the knowledge to students, students still find programming difficult to understand. Instructors teaching programming courses have shown a lot of concern on why students are still struggling to survive in programming courses. An exploratory and participatory study was conducted in Meliksah University to investigate the causes of the most common errors students are making in Java programming classes. Methods investigated to improve students programming ability by initially exposing them to programming errors and then educate them on how to prevent or avoid the errors.

In this research work, programming codes were collected from first year students and then analysed and modelled to find the different type of errors and what cause them. In order to evaluate the techniques, the students were divided into two groups; one educated about the errors through the use of a tutorial and the other group not. Then the experiment results of the two groups were compared and they showed that there is a significant difference between the two groups. The group that undergone the tutorial made less errors. The results of the research will be used in making recommendations to improve teaching programming practices while educating students.

Key words: Programming languages, Programming errors, Error types, Error modelling, Exploratory study, Participatory study, Novice programmers

ACEMİ PROGRAMCILAR TARAFINDAN YAPILAN HATALARI AZALTMAK İÇİN TEKNİKLERİN DEĞERLENDİRİLMESİ

Murtala Mohammed

Yüksek Lisans Tezi – Elektrik ve Bilgisayar Mühendisliği

Ağustos 2014

Tez Danışmanı: Yrd. Doç. Dr. Hasan KİTAPÇI

ÖZ

Araştırmalar programlama becerilerinin öyle kolay kazanılmadığını göstermektedir. Eğitimci, öğrencilere bilgi iletmek için çok fazla zaman harcamalarına rağmen öğrenciler hala programlamayı anlamakta zorlanmaktadır. Programlama derslerini veren eğitimci, neden öğrencilerin hala programlama derslerinde hayatta kalmak için zorlandıkları hakkında endişelerini ifade etmektedirler. Melikşah Üniversitesi'nde Java programlama derslerinde öğrencilerin yapmış oldukları en yaygın hataları ve nedenleri araştırmak amacıyla keşif ve katılımcı bir çalışma yapılmıştır. Öğrencilerin programlama yeteneklerini geliştirmek için başlangıçta onlara programlama hatalarını teşhir ederek hataları nasıl önleyecekleri hakkında eğitmek için teknikler araştırıldı.

Bu çalışmada, birinci sınıf öğrencilerinden programlama kodları toplandıktan sonra analiz edilerek farklı hata türlerini ve nedenlerini bulmak için modellemeler yapılmıştır. Teknikleri değerlendirmek amacıyla, öğrenciler iki gruba ayrılmıştır; bir gruba hatalar hakkında bilinçlendirmek için eğitim verildi, ve diğerine verilmedi. Daha sonra iki grubun deney sonuçları karşılaştırmış ve iki grup arasında anlamlı bir fark olduğu ortaya çıkmıştır. Eğitim almış olan grubun daha az hata yaptığı gözlemlenmiştir. Araştırma sonunda elde edilen sonuçlar, öğrencilere programlama eğitimini iyileştirmek için önerilerde kullanılacaktır.

Anahtar Kelimeleri: Programlama dilleri, Programlama hataları, Hata türleri, Hata modelleme, Keşif çalışması, Katılımcı çalışma, Acemi programcılar

DEDICATION

I dedicated this research work to the family of Mohammed Gambo Fagge and also the Executive Governor of Kano State, Dr. Rabiu Musa Kwankwaso.

AKCNOWLEDGEMENT

Thanks be to Almighty Allah for making me see the completion of this thesis project successfully.

I would like to express my great appreciation to my supervisor in person of Asst. Prof. Dr. Hasan Kitapci for his valuable and constructive advice and suggestions during the planning and development of this research work, his willingness to give his time so generously has been very much appreciated.

My special thanks go to Asst. Prof. Dr. Aytekin Vargun for his advice and assistance during the setting up of the research experiments questions, it was really appreciated sir.

I would also like to extend my thanks to the lab assistants in persons of Gokhan Ozsari, Nevin Cini, Hilal Hacilar and Bilge Kagan Dedeturk for their support in one way or the other during the lab data collection.

At last my special thanks go to my parents for their help, encouragement and prayers.

TABLE OF CONTENTS

APPROVAL PAGE.....	ii
ABSTRACT.....	iii
ÖZ	iv
DEDICATION.....	v
AKCNOWLEDGEMENT.....	vi
TABLE OF CONTENTS.....	vii
LIST OF TABLES.....	xi
LIST OF FIGURES	xii
FIGURE.....	xii
CHAPTER ONE.....	1
INTRODUCTION	1
1.1 INTRODUCTION	1
1.2 MOTIVATION	2
1.3 OBJECTIVE OF THE STUDY	3
1.4 RESEARCH QUESTIONS	4
1.5 THESIS ORGANIZATION	5
CHAPTER TWO.....	6
BACKGROUND AND RELATED WORKS.....	6
2.1 BACKGROUND	6
2.1.1 Programming.....	6
2.1.2 Programming Languages	6
2.1.2.1 Java	7
2.1.2.2 C.....	7
2.1.2.3 C++/C#	8
2.1.2.4 Python	8
2.1.3 Integrated Development Environments (IDE)	8
2.1.3.1 NetBeans.....	8

2.1.3.2 BlueJ	9
2.1.3.3 DrJava	10
2.1.3.4 Eclipse.....	10
2.1.4 Compiler Enhancement Tools and Plug-ins.....	11
2.1.4.1 Gauntlet.....	11
2.1.4.2 Espresso.....	12
2.1.4.3 Arjen	12
2.1.5 Novice Programmer	12
2.1.6 Programming Errors.....	15
2.1.6.1 Syntax Errors	16
2.1.6.2 Runtime Errors.....	16
2.1.6.3 Logical Errors	18
2.1.6.4 The General Error Lists	19
2.1.7 Techniques to Reduce Errors	22
2.1.7.1 Error Referenced Document	23
2.1.7.2 Tutorial.....	23
2.1.7.3 Plug-in (Tools).....	23
2.2 RELATED WORK	24
2.2.1 Online Protocol (Java BlueJ IDE).....	25
2.2.2 Experience (from past and present Instructors)	26
2.2.3 Automated Error Collection System (Custom-made IDE for Java).....	28
2.2.4 Observations (Practical sessions, Test and Exam).....	29
2.2.5 Screen Recorder (C Programming).....	30
2.2.6 Automated Error Collection System (JBuilder IDE)	32
2.2.7 Survey (Survey on International Opinions)	33
CHAPTER THREE	35
RESEARCH METHODOLOGY	35
3.1 IDENTIFYING THE ERRORS.....	35

3.2 DATA COLLECTION TECHNIQUE	36
3.2.1 Participants.....	36
3.2.2 Choice of Integrated Development Environment (IDE)	38
3.2.3 Lab Sessions.....	38
3.2.4 Data from Reviews.....	39
3.2.5 Video (Screen Recorder).....	39
3.2.6 Teaching Assistants’ Interview	40
3.3 LAB EXPERIMENTS ASSIGNMENTS	40
3.3.1 Pre-Experiments (Before Tutorial)	41
3.3.1.1 Writing Assignment (Pre-Experiments)	41
3.3.1.2 Lab Experiment (Pre-Experiments)	42
3.3.2 Tutorial.....	43
3.3.3 Control Experiment (After Tutorial).....	44
CHAPTER FOUR.....	46
DATA ANALYSIS AND MODELLING	46
4.1 DATA ANALYSIS	46
4.1.1 High-Level Analysis	47
4.1.2 Low-Level Analysis	47
4.2 DATA MODELING	47
4.2.1 Multiple if without ‘else’ concepts (if-then-else/nested if-then-else conditional control flow statements).....	47
4.2.2 Loop Concepts (within Array and Non-array).....	49
4.2.3 Method Concepts	53
4.2.4 Array Concepts.....	55
4.2.5 Incorrect placement of Printout statement and Missing Printout statement	56
4.2.6 Code outside Method/Main method.....	58
4.2.7 Missing Syntax.....	58
4.2.8 Keyword mistype and usage of uninitialized/undeclared variable	58

CHAPTER FIVE	60
RESULTS AND DISCUSSION	60
5.1 PRE-EXPERIMENT ASSIGNMENTS RESULTS	60
5.2 CONTROL EXPERIMENT ASSIGNMENT RESULTS	64
CHAPTER SIX.....	69
CONCLUSION AND FUTURE WORK	69
A.1: Pre-Experiments Data Showing Location of Error Occurrence.....	71
A.2: Students' GPA and the Grouping.....	72
A.3: Control Experimental Results Showing Location of Error Occurrences	73
APPENDIX B	74
Some Questions Treated In the Tutorial	74
APPENDIX C	77
Written Experiments Paper Based Assignment Analysis	77
References.....	82

LIST OF TABLES

TABLE

Table 2.1 Summary of Error Types	19
Table 2.2 Techniques to Reduce Error	24
Table 2.3 Summary of Related Works	25
Table 2.4 Lists of Errors from Emily, et. al. Works	26
Table 2.5 List of Errors identified in Flowers, et. al. works	27
Table 2.6 Lists of Errors collected in Jackson et. al. works	29
Table 2.7 Difficulties Experienced by Students Programming	30
Table 2.8 Lists of Barriers Faced by Students	31
Table 2.9 Lists of Errors Collected in Mow et. al. works	33
Table 5.1 Lists of Errors from Pre-Experiment	60
Table 5.2 Control Experimental Results	64
Table A.1 Pre-Experiment Error Analysis	71
Table A.2 Two Groups Formed based on GPA	72
Table A.3 Control Experiment Error Result Analysis	73

LIST OF FIGURES

FIGURE

Figure 2.1 NetBeans IDE Screen Shot	9
Figure 2.2 BlueJ Screen Shot	10
Figure 2.3 Screen Shot of a Syntax Error	16
Figure 2.4 Screen Shot of Run-Time error	17
Figure 2.5 Screen Shot of Logical Error	18
Figure 2.6 Showing Errors Subgrouping	21
Figure 3.1 Overview of Steps followed	35
Figure 3.2 Errors Identification Steps	36
Figure 3.3 Control Flow of Experiment	40
Figure 4.1 Control Flow of the Analysis	46
Figure 4.2 Position of Error Occurrences	48
Figure 4.3 Usage of Multiple if	49
Figure 4.4 Possible location of Error	50
Figure 4.5 Possible Locations of Errors	50
Figure 4.6 Wrong Termination Condition	52
Figure 4.7 Incorrect Accumulator Usage	53
Figure 4.8: Incorrect parameter match	54
Figure 4.9: Forgot to Create an Object	54
Figure 4.10 Missing [] in an Array	55
Figure 4.11: Usage of [] instead of {}	55
Figure 4.12: Incorrect Termination Condition	56
Figure 4.13 Missing Printout Statement	57
Figure 4.14 Incorrect Printout Placement	57
Figure 4.15 Usage of Undeclared Variable	58
Figure 4.16 Missing keyword	59
Figure 5 Control Experiment Results	66
Figure C.1 Wrong Result Output	78
Figure C.2 Showing Usage of i without Declaration	78
Figure C.3 Shows Placement of an Incorrect Placement of Printout	79
Figure C.4 Shows Usage of a While-Loop like For-Loop	79
Figure C.5 Wrong Result Printout in Loop	80
Figure C.6 Missing Object in Calling of Non-Static Method	80
Figure C.7 Incorrect Parameter Mismatch	81

CHAPTER ONE

INTRODUCTION

1.1 INTRODUCTION

With the high increasing demand of Information Technology stuffs, the need for properly trained IT Professionals is increasing rapidly and the demand for highly skilled programmers is on the increase as well. This is indirectly increasing the pressure on Information Technology schools to get more and more IT students properly trained.

On the other hand students at first are attracted to enroll in Information Technology related degree programs but later on changed to another field of study. This is believed to be as a result of the difficulty of the programming courses [1], [2], [3].

Programming has been described as a useful skill and also regarded as a rewarding career. In recent years the needs for programmers is growing high and the high rate of interest in programming by students is in rapid grow. First year programming students suffer from a wide range of difficulties and deficits. Programming courses are generally regarded as difficult, and are said to have the highest dropout rates [4], Winslow, L.E further stated that it takes 10 years for a novice to become an expert in programming.

Learning to program is hard, in a study conducted by McCracken, et al. [5], researchers found that 35% of programming students fail their introductory course, while in UK and the US almost 30% of IT students do not understand basics programming concepts [5]. This has led to a serious concern among computer science educators due to lack of understanding of programming courses by programming students. Novice programmers are

of special interest to the computer science education community which made computer science education researchers to have conducted investigations into several kinds of problems novice programmers come across while learning to program.

1.2 MOTIVATION

Programming is central to any IT field of study in the university. Generally in the first year of study of an Information Technology or Computer Science studies, a programming course is introduced to students in order to serve as their basic starting point for related IT courses. Instructors teaching programming courses have recognized it as a complex and difficult intellectual activity in which students struggling through their programming introductory course and instructors struggling to teach it [6]. Daryl D, et al. [7], stated that Institutions put more efforts to teach first year programming course to students and allocate much resources and time to ensure success and highly long term outcomes in a way of developing students' skill and confidence in programming. Despite all the allocation of time and resources, programming continues to be an issue to novice programmers or first year programming students.

Cooper et al.[8] Asserted that a problem might arise as students are struggling in developing an algorithms or thinking of how to apply problem solving skills in their programs, and also in the program constructs.

Early identification and addressing of these issues can help educators in IT institutions to provide useful help and proper intervention to novice programmers or first year programming students. By having the knowledge of errors made by programming students and time required to fix these errors, IT institutions can help in addressing the difficulties students come across while programming. A simple error can be too hard for a first year programmer to find and fix correctly. Sometimes an error that takes too much time to correct can be frustrating. Early addressing of these issues can help in reducing or prevention in the high dropout rates in computer programming courses.

1.3 OBJECTIVE OF THE STUDY

Researches in empirical evidences have showed that learning to program is difficult for most students [9], [10], [11]. Evidence showing the difficulties with programming is the high failure and dropout rates in introductory programming courses [9]. Researchers has made suggestions that the difficulty associated in learning programming might be due to inadequate mental model [12], [13], [14], [15], misconception of programming constructs [16], [17], [6], [18], [19], inadequate programming strategies [20], [16], [21], [19], [22] and lack absence of debugging strategies [23], [24].

E. Lahtinen, et al. stated that [6] IT institutions recognizes programming as a difficult and complex intellectual activity with educators put more effort in teaching it and students struggling hard to learn it. Cooper, et al. stated [8] that problem arises as students are struggling in figuring out how to come up with a problem solution. Over the years researchers has identified several factors affecting student programming learning but still more research is needed in understanding why some students learn programming quickly and others struggled with it [9].

Milne, et al. and E. Lahtinen, et al. Student behavior towards programming have been categories into three groups [25], [6], which are stoppers, movers and extreme movers.

- **Stoppers:** Programming Student if encountered with a problem got frustrated and abandons the hope of resolving the problem on their own. They lack confidence in handling the programming language.
- **Movers:** Programming Student If encountered with a problematic situation try using different approach, trying different solution methods, using feedback information effectively in their programming task and modifying the code until the goal is achieved. Movers eventually succeed in arriving at a solution.
- **Extreme Movers:** This group of students does not make much progress like the stoppers, they tend to ignore feedback and they made changes at random in their programming task.

Perkins D.N., et al. [13] and Lane, et al. [26] in their works stated that the main problems in learning programming are complexity and instability

- **Complexity:** The rapid increase in the number of programming details that student need to master has grown faster than the corresponding number of high level concepts.
- **Instability:** The languages, libraries and tools are changing more rapidly than in previous years. Now days there is rapid increase in languages, IDE and tools development. Each has its own way of usage in terms of syntax that differs from other languages i.e., new language implies new learning of usage syntax.

1.4 RESEARCH QUESTIONS

In this research study an exploratory and participatory study was used to help in providing information to educators towards the understanding of the most common errors students are making, and also provide tangible advice toward addressing it. This research work is going to going to answer the following questions below.

- What are the errors that students are frequently making?
- What causes those errors?
- What are the typical errors from the literature review?
 - What is relationship between the errors we got from our data with that of the previous literature reviews?
- Which method should be used to reduce/eliminate the bugs/errors made by novice programmers?
 - Does informing students about errors at the beginning of the semester prevent them from doing it?

The aim of this research work is to refine our teaching methods by learning from the students past mistakes, considering the errors types and recommend a solution. To achieve our goal we need to understand novices' behavior while programming.

1.5 THESIS ORGANIZATION

In the first Chapter some background information about novice programming and the purpose of the research work were presented. In Chapter 2, some information about programming languages, novice programmer characteristics, and related works on novice programming errors was highlighted. In Chapter 3, the methodology that was employed in carrying out the research was discussed. In Chapters 4, how the data was modeled and the results of the analysis were presented. Chapter 5 contains the discussions of our finding and lastly Chapter 6 contains the conclusion and future works.

CHAPTER TWO

BACKGROUND AND RELATED WORKS

In this chapter, the theoretical background and related works concerning the research work will be discussed.

2.1 BACKGROUND

2.1.1 Programming

It is important to understand what programming and how important it is in our modern life. People program in order to develop software which can be of different forms such as Operating system web applications or even games. Software consists of a series of instructions that direct or control hardware to perform a task or an action. In other word Programming in general can be referred to as the coding part in developing software.

A programming environment is the collection of tools used in software development. This collection may consist of a large collection of integrated tools, or may include only a file system, text editor, linker, and a compiler [27].

2.1.2 Programming Languages

Programming language is a set of rules, instructions and constructs that leads to a software development. Many programming languages are in existence and each of these languages has its own unique syntax. Programming Languages are sometimes classified into 2 types: high level language and low level language.

In recent time most software developers use high-level languages such as C, C++ or Java. High-level languages do not use instructions that the machine understands directly; instead, they utilize commands that are more intelligible for humans. Naturally, high-level languages are easier to use. Few high-level language instructions are required to perform the same task in a low-level language. While programmers using a high-level language lose direct control of instructions to the machine and this is not a concern for most developers.

Low-level languages have a one-to-one correspondence between a statement in the language and a machine instruction. These languages are normally called assembly languages and are unique to a specific computer. They provide a great deal of control over what can be done by a program but are very difficult and tedious to use.

2.1.2.1 Java

In the 1990s the Sun Microsystems developed Java as a class-based, object-oriented programming language. Today in an IT market Java is one of the most programming language in demand, it is a standard programming language for commercial software, web-based applications, games and even mobile applications, and even the Android operating system. Due to Java nature of compatibility it is designed to work across many software platforms [28].

2.1.2.2 C

C is one of the oldest and most widely used programming language, providing the building blocks for other popular languages, such as C#, Java, JavaScript and Python. It is mostly used for implementing operating systems and embedded applications.

C is a general-purpose programming language with features economy of expression, modern flow control and data structures, and a rich set of operators. C is not a “very high level” language, nor a “big” one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages. C was originally designed and implemented on the UNIX operating system [29].

2.1.2.3 C++/C#

Pronounced "C-sharp", C# is a multi-paradigm language developed by Microsoft as part of its NET initiative. Combining principles from C and C++, C# is a general-purpose language used to develop software for Microsoft and Windows platforms. C++ extends the C language to provide object-oriented features.

The language is backward compatible with C, and code from the two languages can be used with each other with little difficulty. C++ has found quick acceptance and is supported by a number of pre-built specialized classes [30].

2.1.2.4 Python

Python is a high-level, server-side scripting language for websites and mobile apps. It is considered a fairly easy language for beginners due to its readability and compact syntax, meaning developers can use fewer lines of code to express concepts than they would in other languages [28].

2.1.3 Integrated Development Environments (IDE)

An Integrated Development Environment (IDE) is an application that simplifies the process of application development. In general, an IDE is a graphical user interface based workbench intends to assist a developer in building software applications. Most common features, such as debugging, version control and data structure browsing, help a developer to quickly execute actions without switching to other applications. Thus, it helps maximize productivity by providing similar user interfaces (UI) for related components and reduces the time taken to learn the language. An IDE supports single or multiple languages [31].

2.1.3.1 NetBeans

NetBeans is an open source project intended only for the purpose of providing software development products. NetBeans is a pure Java development environment. It offers powerful source and visual editor capabilities, templates that aid in the creation of various types of Java classes, and the ability to compile, run, and debug applications, all in

one tool. Writing components to the JavaBeans specification means your code will be easier to read and you won't have to write as much custom code to glue components together. They communicate with their neighbors using events and other normal method invocations [32]. The subject in this research work uses NetBeans IDE in learning programming.

The NetBeans has some features such as auto word suggestion and code templates. Some instructors felt that by having code template automatically inserted into student programs, students would be at a disadvantage if they have to write code from scratch.

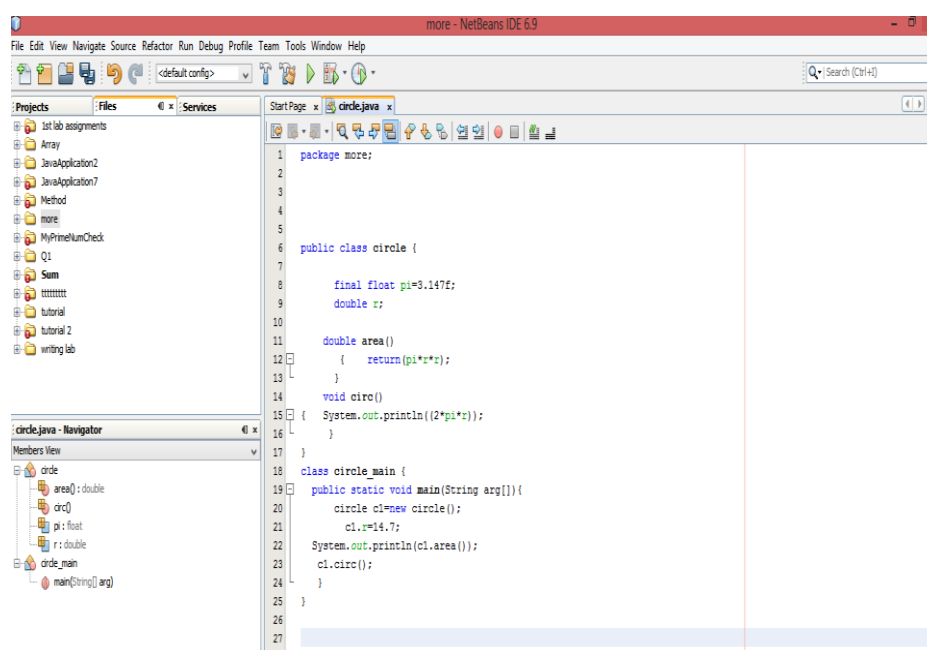


Figure 2.1 NetBeans IDE Screen Shot

2.1.3.2 BlueJ

BlueJ is a collaborated project between the University of Kent at Canterbury in the UK and Deakin University in Australia, and was supported by Sun Microsystems [33]. BlueJ is an integrated Java environment developed purposely for teaching introductory Java Programming. The aim of BlueJ is to provide an easy-to-use teaching environment that facilitates the teaching of Java to students learning to program so as not to overwhelm beginners with error messages. BlueJ only shows one syntax error at a time, the program has to be recompiled again to check for another error. Each time a code containing an error

is compiled, the line in question is highlighted, and the error message is reported at the bottom of the editor window [34].

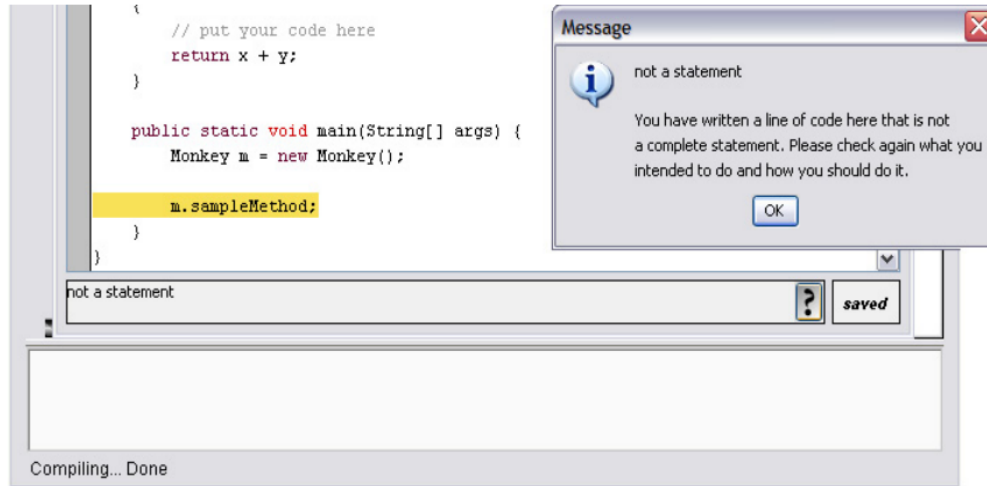


Figure 2.2 BlueJ Screen Shot

2.1.3.3 DrJava

DrJava was developed by Rice University in the USA [35]. It is considered as the perfect IDE for new programmer, it has an easy to use interface. It was observed that traditional Integrated Development Environments were not well suited for novice programmers so the developers came up with it. DrJava is use both as a plug-in for some IDEs and also as a standalone e.g. DrJava can be used as plug-in if integrated with eclipse. DrJava was developed in such a way that it assists a novice programmers a way that it highlighted the error and give useful information regarding it.

2.1.3.4 Eclipse

Eclipse was developed by IBM in November 2001 and supported by a consortium of software vendors. Eclipse is a popular IDE among Java developers. Eclipse gives room for plug-ins functionality extension. It organizes projects in workspaces and changes views with perspectives. Eclipse IDE comes in different flavors depending on user needs.

In Meliksah University NetBeans IDE is used in teaching Java programming to its students, it was believed to be more friendly to students compared to other IDEs, a user does not need to know more about java to start programming with NetBeans due to its easy to use functions of drag and drop. The participants in this research work use NetBeans IDE as their programming environment.

2.1.4 Compiler Enhancement Tools and Plug-ins

Enhancing the compiler itself can lead to the development of tools and plug-in that explains error messages more in layman's language. Unlike the error messages provided by some compilers these enhancements tools helps in pre-processes student source code and explains in layman's terms each of the syntax errors found. These tools are not widely available because many of the tools were developed by institution's which are concerned in students wasting time in trying to fix an error instead of concentrating on problem solving.

Some of the tools do not even have a specific name. Most of these tools were custom built developed by the institutions that felt the needs for it usage by their students. Although most of the institutions that developed these tools do not make it available for download, they only share their experiences with the tools; these experiences can be valuable resources for other programmers who are also tool developers.

2.1.4.1 Gauntlet

It was observed by the instructors teaching a Java course at the United States Military Academy that cadets were not focusing on providing solution to a given problem instead struggling with Java syntax trying to find and fix one or more errors, thereby wasting a large amount of time. As a result, Flowers, et al. [36] developed Gauntlet, a pre-compiler that provides a better explanation of the common syntax and semantic errors in a layman's language cadets are struggling with. Gauntlet provides error explanations for 9 errors which were suggested by the programming course instructors. These 9 common errors were determined from experience by the past and presents instructor teaching the course. Gauntlet helps students by providing steps that can be followed to solve their problems without needing help from their instructors. Gauntlet messages are written in a

friendly manner, and even praise a user when a code is compiled successfully. Gauntlet was reported to be successful due to an improvement in code quality of students.

2.1.4.2 Espresso

In 2003 a tool called Espresso was developed by Hristova et al. [37] at Bryn Mawr College to detect 20 most common errors made by programmers. The list of errors was gotten from a survey of lecturers, teaching assistants and students which helps in identifying the most common errors then. Espresso was developed as a pre-processor, which is first run before the actual compilation stage. If one of the 20 errors is detected an error message is displayed which provide a full descriptive information regarding the error. Espresso is a pre-compiler, like Gauntlet, which provide students with detailed on the error e.g. line number where the error occurred and suggestions for fixing the error.

2.1.4.3 Arjen

Another tool called Arjen was developed by Dr Warren Toomey at the School of IT, Bond University [38], just like the Espresso and the Gauntlet to identify common errors. Arjen identifies 29 programming errors, which were initially collected and identified by Hristova et al. [37] and Flowers et al. [36] in their research.

2.1.5 Novice Programmer

A novice programmer is the first stage of being a programmer. It has also been described as an end-user who wants to program a computer. An IT student who have none or have some basic background in programming experience can also be referred to as a Novice Programmer [39].

Over the years there are researches that have focused mainly on topics like program comprehension, mental models, and the skills and knowledge required for programming.

In 1993 Davies [40], differentiated between programming knowledge of a declarative nature, e.g., stating how a “for” loop works and programming strategies i.e. the

manner in which knowledge is used and put into practice, e.g., while working with a “for” loop should be used in a proper way.

Rogalski, et al. [41] and Detienne, et al. [42] both cited that Knowledge of programming is said to be acquired and developed through a highly complex process. It includes a lot of cognitive activities and mental visual depiction related to the design of the program, understanding of the program, program modification and debugging. Even at an expert level, construction of conceptual knowledge is needed, and the basic operations structuring is required. A novice devote little amount of his time in planning and testing of code when compare with expert and tend to try a small “local” fixes rather than in a significant manner [43].

They have poor tracing and tracking ability of their code [44]. They have a poor understanding of sequential execution of a program: ‘what sometimes gets forgotten is that each instruction operates in the environment created by the previous instructions’ [45]. Their knowledge tends to be context specific rather than general [46].

Perkins et. al. emphasis on some important issues regarding the aspects of novice programmer knowledge and strategies as “knowing”. Which he described “knowing” as not necessarily clear cut, and a novice that appears to be lacking in certain knowledge may in fact have learned the required information. Perkins et. al. [47] further refers to the knowledge that a student has but unable to use as “fragile”. This Fragile knowledge can be in different ways such as: missing (forgotten), inert (learned but didn’t used), or misplaced (learned but used not in a proper way). Strategies can also be fragile, with students failing to trace/track code even when aware of the process [40].

In his study explored variable concepts, the author [47] showed that initialization is a very complicated cognitive operation which he noted that novice understood concepts of reading input that assignment. In a study by Hoc, JEAN et al. [48] noted that variable testing and updating seems to be of same complexity, and is understood more than initialization.

Spoehrer, et al. made [49] conclusion that bugs associated with loops and conditionals were much more common than those associated with input, output, initialization, update, syntax and block structure, and overall planning. Soloway, et al. [50] Studied the use of loops, noting that novices more preferred a “read then process” rather than a “process then read” strategy. In a study conducted by Boulay [45], it was noted that “for” loops are problematic because novices often fail to understand that “behind the scenes” the loop control variable is being updated, Boulay also notes problems that can arise with the use of arrays, such as confusing an array subscript with the value stored.

Novice programmers are often known to have difficulty in understanding the foundation level of programming concepts, sometimes resulting in grief and frustration, and easily giving up but those who do manage to overcome their learning difficulties are able to move on, and even excel.

McCracken, et al. stated that Novices’ knowledge has been found to be limited and shallow; they struggle with writing syntactically correct programs [5]. Novice programmer usually lack detailed mental models of various programming constructs [51], Boulay, et al. noted that novice programmer tend to organize knowledge based on superficial similarities [45]. Robins, et al [1] and Winslow, et al. [4] both stated novices use general problem solving strategies instead of problem specific or program specific strategies, and approach programming "line by line" rather than at the level of meaningful program structures. Novice programmers have been characterized to have poor program comprehension in tracing their codes [13]. Ahmadzadeh, et al. [23] stated that novice programmer has a poor knowledge of steps involved in execution of a program and have problems with understanding that each instruction is executed in the state that has been created by the previous instructions.

Novice programmer often tries to repeat the lessons taught in the lectures, try coding similar examples and in this process, he/she gets a number of errors which they might not be able to deal with. Some students learned from the bugs they encountered while others got frustrated every time they got encountered with an error and tend to view bugs as a measure of their performance which indirectly lowers their moral confidence.

One big problem is that most novice programmers cannot make good use of error messages that the compiler displayed when there is an error. These messages are hard to comprehend for most novice programmers, because most of the messages only indicate a syntax error and the line in which it occurs without detailed explanation.

As a response to all these verdicts, educators and researchers in Computer Science field conducted many studies to identify the characteristics of novice programmers, and determine the causes to find possible solutions to their problems. In the past various methods had been used in the study of novice programmers among which I discussed some in the literature review.

2.1.6 Programming Errors

An appropriate definition of a programming error has to be decided. In this research, we use definitions of error, fault and failure from IEEE standard 610.12-1990. A failure occurs when a program's output does not match documented output requirements, or the programmer's mental model of output requirements. Failures are mostly cause as the result of a fault, which is a runtime state of a program that either is or appears to be incorrect. Faults occur as a result of errors, which are program fragments that do not comply with documented program specifications or the programmer's mental model of specifications (such as a missing increment or misspelled variable name). Failures are usually the first indication to the programmer that one or more errors exist in a program, although some errors are found before it causes failures, since they may be similar to errors already found or may be apparent when inspecting code. While a failure guarantees one or more faults exist and a fault guarantees one or more errors exist. Errors do not always cause faults, and faults do not always cause failures.

Programming Errors are also referred to as program bugs which occur as a result of bad coding which leads to producing an erroneous result. Errors are inevitable, so all programs have to deal with errors. These errors are unexpected conditions that are not part of the normal operation of a program.

2.1.6.1 Syntax Errors

Syntax errors are also referred to as Compile time error, these type of error is mostly common with beginners of programming, each time a programmer fails to obey one or more programming rules of a language a syntax error occurs. A common example is using a wrong case, missing closing braces etc. syntax errors are mostly caught by the compiler. Most of the time if there is a missing syntax, the compiler indicates and in some cases provides a hint about the possible caused of the error.

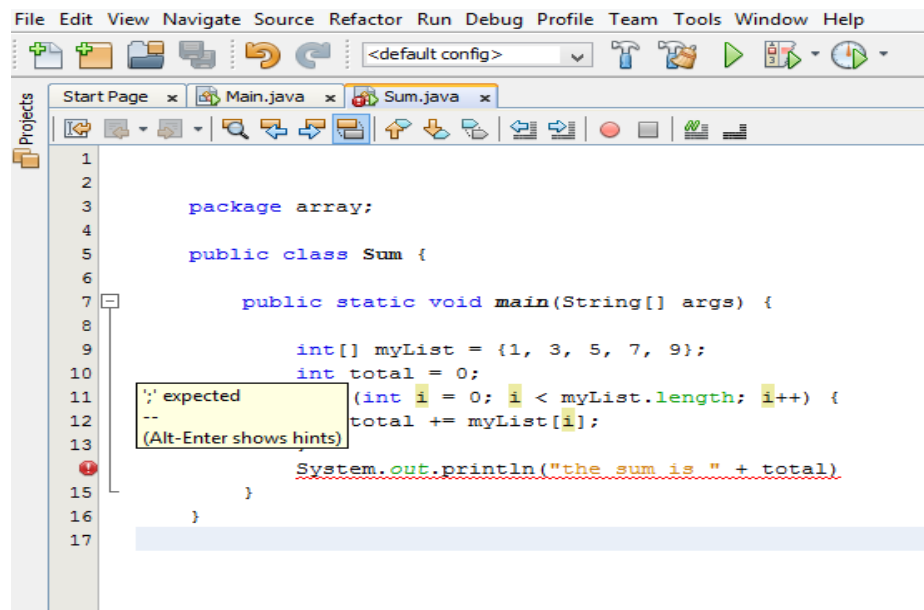


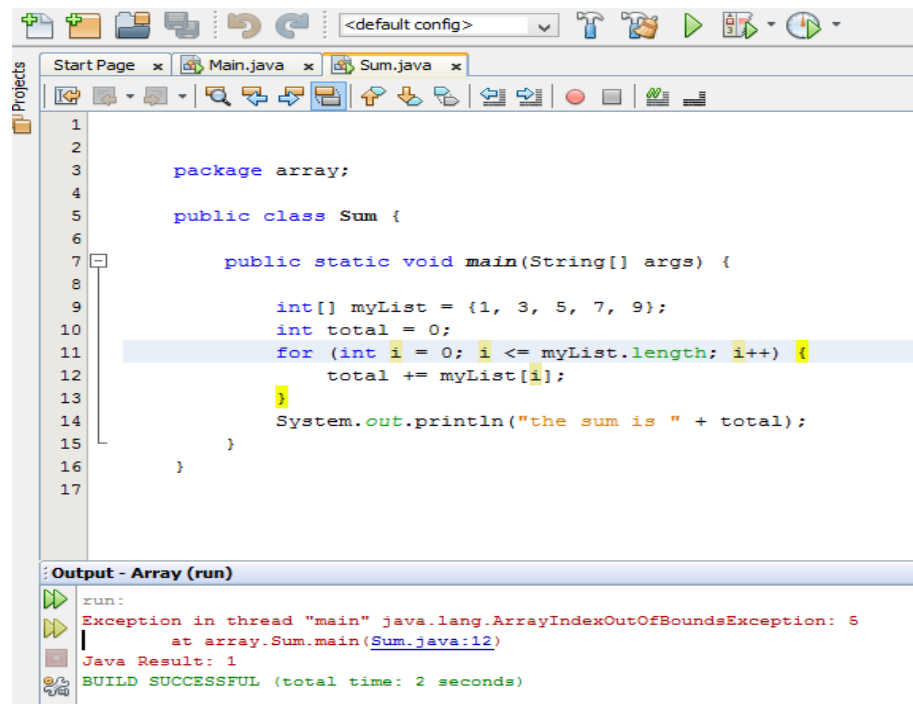
Figure 2.3 Screen Shot of a Syntax Error

The figure above shows a code fragments with syntax error. In this example, the error is a missing semi-colon. A semi-colon is required at the end of each statement. So to fix the problem, you just place a semi-colon at the end of the line where the compiler is complaining.

2.1.6.2 Runtime Errors

A runtime error is also referred to as a semantic error, these are errors mostly caused by the misuse of a programming concept, even though the program is syntactically correct. Each time a programmer instruct a program to do something that it is either incapable or

unwilling to do. A program can be syntactically correct but still does not run due to a runtime error; this type of error is related to the semantics of the language used. A simple example of a runtime error is the use of different data type or incompatible data types in an assignment statement, or if your program tries to use a variable that has not been initialized. Runtime errors are usually caught when the program code is compiled. The compiler does not indicate a runtime error until when the program halts and terminates after it has been compiled.



```

1
2
3 package array;
4
5 public class Sum {
6
7     public static void main(String[] args) {
8
9         int[] myList = {1, 3, 5, 7, 9};
10        int total = 0;
11        for (int i = 0; i <= myList.length; i++) {
12            total += myList[i];
13        }
14        System.out.println("the sum is " + total);
15    }
16 }
17

```

Output - Array (run)

```

run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
    at array.Sum.main(Sum.java:12)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)

```

Figure 2.4 Screen Shot of Run-Time Error

In Figure 2.4 above the program tries to iterate beyond the length of the array, so in attempting to do that, the program halts in the line where it tries to perform the improper operation and the compiler reports a runtime error exception in the line where the wrong operation was performed. In this example in line 5, since the array index starts from 0 and goes to array's length-1, the "<" operator must be replaced by "<=" to fix this error.

2.1.6.3 Logical Errors

Logical errors are the most difficult errors to find and fix, because the compiler does not indicate an error message instead the program runs successfully but does not output the correct results. In other words, it simply does not produce the expected results. Logical error is said to occur if the expected program output does not match the program input. This is as a result of wrong algorithm usage by a programmer in implementing the program solution.

Logical errors are usually caused by one of the following. If a programmer does not understand how a program is supposed to behave, if the behavior of each operation is misunderstood or the programmer is careless.

```

1
2
3     package array;
4
5     public class Sum {
6
7         public static void main(String[] args) {
8
9             int[] myList = {1, 3, 5, 7, 9};
10            int total = 0;
11            for (int i = 1; i < myList.length; i++) {
12                total += myList[i];
13            }
14            System.out.println("the sum is " + total);
15        }
16    }
17
Output - Array (run)
run:
the sum is 24
BUILD SUCCESSFUL (total time: 5 seconds)

```

Figure 2.5 Screen Shot of Logical Error

In Figure 2.5, the program supposed to sum the variables in the array index but there is a logical error in the code due to wrong initialization of *i*, which is the variable that iterates within the array length, an array index always starts from index 0 but in the case of this example, it was initialized to start from index 1 which is the 2nd variable in the array index. In this case it omits the first variable in the index 0 while doing the summation.

Table 2.1 Summary of Error Types

Summary of Error Types			
Error	Causes	Example	Error message
Syntax	when a syntax of language is violated.	if (i > j // unbalanced parentheses max = i // missing semicolon	. . expected
Runtime	when there is improper use of program statements.	int[] v = new int[10]; v[10] = 100; // 10 is not a legal index for an array of 10 elements. Array index out of range: The array v has been created with 10 elements with indexes ranging from 0 to 9), and we are trying to access the element with index 10, which does not exist.	java.lang.ArrayIndexOutOfBoundsException
Logical	Cause due to the fact that the specification is not respected. The program is compiled and executed without errors, but does not generate the requested result.	public static int sum(int a, int b) { return a - b ; } // this method returns the wrong value wrt the specification that requires // to sum two integers	No Error Message Displayed

2.1.6.4 The General Error Lists

The classification of errors by Hristova et al. [52] and Amadzadeh et al. [23] in terms of syntax, semantic and logic was too general, so a new and more specific category based on programming concepts was created. We classified the errors into the following 9 categories some of which has subcategories.

1. **E1:** Incorrect usage of selection structures (If and nested-If statements). Gobil, et al. [53] noted that novice programmers have difficulties in using selection structure. Multiple if without else, if-else used instead of an else in the last option (logical error).
2. **E2:** Wrong Iteration Looping usage (for/while/do-while): update [54] stated that novice programmers have difficulty in working with loop conditions, logical conditions variable initialization and update, also arithmetic operations. [54], further states that errors can occur because of students incorrect misconception of a control structure, e.g. updating a loop counter whenever a condition is checked, testing the condition of a do-while loop at the beginning of the loop, Missing

increment/decrement statement or wrong placement (infinite LOOP) or wrong accumulating/updating in a loop) of a variable in a loop.

- **E2.1:** (Off by one) loop initialization/termination (logical): it occurs when a loop iterated one or more additional times incorrectly. This problem arises when a programmer makes mistakes such as using "`<=`" instead "`<`" should have been used in a loop condition or fails to take into account that a sequence starts at zero rather than one. Exceedingly common error induced in many ways, such as by starting at 0 when you should have started at 1 or vice-versa, or by writing "`< N`" instead of "`<= N`" or vice-versa.
 - **E2.2:** wrong accumulator usage (logical error)
3. **E3:** Wrong Method/functions (defining, calling a static/non static method and passing arguments to the function).
- **E3.1:** Type incompatibility i.e. passing wrong arguments to a method (syntax error)
 - **E3.2:** Missing object in accessing a non-static method (runtime error)
 - **E4:** Incorrect Array usage (declaration and assignments of variables), creation & assignment of values (loop/not loop), (try accessing an index that was not created usually due to using `</<=` sign), Incorrect usage of Logical/relational operators (`< > =` usage of Single=`==` instead of double=`===` usage) [53], stated that novice programmers have difficulty in naming variables and assigning suitable data types and structures to these variables, analyzing and designing mathematical expressions, evaluating correctly output statements, arithmetic expressions usage, and relational expressions [53].
 - **E4.1:** Missing `[]/{}` in array (runtime error)
 - **E4.2:** Array index out of range (runtime error)
4. **E5:** Wrong placement of printout statement/ omission of printout statement
- **E5.1:** Missing printout statement (logical error)

- **E5.2:** Incorrect printout placement: (logical error)
5. **E6:** Writing of code outside the main method/method function or forgetting the static keyword in the main method (syntax error).
 6. **E7:** Missing Syntax: omission/adding extra semi-colon, parentheses: The semantic and syntactic errors can be easily recognized by the compiler. Schackleford, et al. stated that novice programmers spend more of their development time in debugging code and thereby solving the semantic and syntactic issues.
 - **E7:** Missing semicolon/parentheses: (syntax error)
 7. **E8:** Mistyping of variable or keyword (declared a variable with CAPS try using it with SMALL) or Usage of undeclared/uninitialized variable.
 - **E8.1:** Use of uninitialized/undeclared variables: (syntax error)
 - **E8.2:** Keyword mistype: (syntax error)
 8. **E9:** Testing of swapping of two variables

Figure 2.6 shows the errors type and it sub-grouping.



Figure 2.6 Showing Errors Sub-grouping

2.1.7 Techniques to Reduce Errors

Novice programmer's faces a lot of difficulties while writing a computer program: the difficulties might ranges from the programming concepts (flow of execution, functions and parameters etc), the semantics and syntax of the programming language being taught, the process of implementing a solution of a problem into a computer program.

The task is made more challenging for a novice, when the compiler issues an error message if the language syntax or semantic has been violated by a programmer e.g. missing semicolon, and further more things got more complicated if the error messages displayed by the compiler cannot be interpret correctly by the students.

A novice programmer would learn how to reduce or even learn not to make these mistakes if useful feedback would be comprehend to them in a proper way while they are learning to program. This should reduce the time spent on fixing an error and strengthen the learning of the programming concepts. Techniques that help in easier identification and reduction of an error allow students to focus on problem solving instead of debugging.

Different techniques can be used to reduce the rate of error occurrence, I am going to discuss three techniques when used effectively can help to reduce or even eliminate the mistakes programmers are making.

- Reference documents
- Tutorial
- Tool (plug-in)

Tools (plug-ins) are good at identifying syntax and semantic errors but are not actually good for identifying a logical error. Tutorial sometimes can give ideas to students in all aspect of the errors type including the logical error type that cannot be identify by a tool, further more during tutorial critical places where student have problem with can be emphasis. In documentation the students refers to a textbook or thesis report on their own without guidance from an instructor.

2.1.7.1 Error Referenced Document

A referenced document like a research report or thesis report related to a field of studies can be used by students to educate themselves or address the issue or topics ones has difficulties in.

2.1.7.2 Tutorial

Tutorial is a unique aspect of the educational experience that Institutions offers its students. It is a way of educating a group of students on what have already been taught in a class. Institutions such as University of Oxford, Pembroke College offered weekly tutorial to its students, some students who are shy to ask questions during the class lecture tends to be more comfortable and active during a tutorial session. At the tutorial, the tutor does not go through all the materials as if it were in a class, instead the tutor is there on issues that students have difficulties with, further more and make emphasis on any problems a student might have in understanding the class lecture, to indicate how well what a student have learned fits together and mostly to make emphasis on critically areas and ideas that have been discussed in the class lecture.

Fewer researches have been done in educating students in programming errors using a tutorial. In this research, we use tutorial to educate our students on common programming errors and then compared the results of their performance before and after the tutorial.

2.1.7.3 Plug-in (Tools)

Existing researches has showed that first time programmers often make the same types of mistakes. Tools or plug-in have been developed to help students with useful information each time they violate a syntax or semantic of a language. Among these tools some were discussed in chapter two. Tools discussed in chapter like Gauntlet, Espresso and Arjen were used to detect errors made by students learning programming. Each of the tools was integrated into an existing IDE. It provides more descriptive error information than a normal compiler, in a multiple lines and also recommends a possible solution to the error. Table 2.2 give a short summary of the techniques that can be used to reduced error.

From the three techniques mentioned above, in this study we used tutorial to test how effective it can be to help students in error reduction.

Table 2.2 Techniques to Reduce Error

Techniques to Reduce Errors		
Tools (plug-in)	Tutorial	Referenced Document
+ Automatic error detection	+ More critical errors are discussed	+ More details in form of a report
- Does not detect all errors	- Does not cover all the exercises	- No guidance students read and understand on their own
- Takes more time to develop	+ Can create awareness on novice programmers	- Takes time to have a complete list of errors

2.2 RELATED WORK

Understanding novice programmers is the first step in helping them learning to program. Different studies have been conducted in trying to understand novice programmers. Novice programmers lack programming strategies and have difficulties in using selection structure [53]. They have difficulties in expressing natural language solutions into computer programming language [55], [56]. Analyzing and designing mathematical expressions, naming variables and assigning suitable data types and structures to these variables, evaluating correctly output statements, arithmetic expressions, and relationship expressions [53]. They have difficulty in debugging loop conditions, conditional logic, arithmetic errors, and data initialization and update [54].

Many previous research works have tried to identify the common errors made by novice programmers. Reviews have shows that a variety of methods had been used by researchers to identify the student's common programming errors. Such as Faculty surveys [57], [52], past experience [57], manually counting and categorizing of errors [58], [59] and automated compilations of the most common errors [49]. Table 2.3 below shows the summary of the related works method used, number of participants and where it was conducted.

Table 2.3 Summary of Related Works

Summary of different Researchs				
	Research	Conducted at	# of participants	Methodology
1	[60]	Ateneo de Manila University	143	Online protocol
2	[36]	United States Military Academy	Instructors from the faculties	Survey from previous and currents instructors teaching programming
3	[61]	United States Military Academy	583 from 11 faculties	Real-time automated error collection system
4	[62]	University of KwaZulu -Natal and Mangosuthu Technikon	91	Observations during practical sessions
5	[63]	College of Engineering at Texas A & M University – Kingsville. C programming	16	Screen recorder
6	[64]	National University of Samoa	68	Real-time automated error collection system
7	[6]	6 universities.	500 students and teachers	International survey

2.2.1 Online Protocol (Java BlueJ IDE)

A number of studies had looked at novice programmers in order to identify their errors. Emily, et al. [60] Conducted a research in finding the common programming errors students taking part in an Introductory to programming course at the Department of Information Systems and Computer Science (DISCS) of the Ateneo de Manila University are making. The participants of this study were the students enrolled in an Introduction to programming. A total number of 143 students were used as the subject of the study. The computers for the study were connected to the local area network and the internet. BlueJ IDE was used during the programming assignments. Online protocol was used in the collection of the data, each time a user clicks on the compile button an extension which was

integrated into the BlueJ sends data to the server, each time a participant compiles and test their programs the IDE attempt stores the source code, the error message, the line number in which it occurred, and the exact compilation time in a database. The data were collected in a five assignments during the laboratory sessions. A total compilation of 28,386 was collected, of which 59% compilations generated an error. Emily, et al. concluded that there were 52 different error types encountered during the research and 76% accounted for the top ten errors. Table 2.4 below shows the top 10 errors encountered in Emily et al. works.

Table 2.4 Lists of Errors from Emily, et al Works

Top 10 Errors collected by Emily, et al.		
	Error	Percentage
1	Cannot find symbol – variable	20%
2	‘;’ expected	13%
3	‘(’ or ‘)’ or ‘[’ or ‘]’ or ‘{’ or ‘}’ expected	10%
4	Missing return statement	8%
5	Cannot find symbol – method	6%
6	Illegal start of expression	6%
7	Incompatible types	4%
8	<identifier> expected	4%
9	class, interface, or enum expected	3%
10	‘else’ without ‘if’	2%
	TOTAL	76%

2.2.2 Experience (from past and present Instructors)

In 2004, Flowers, et al. [36] at the United States Military Academy conducted a research on new intake cadets taking an introductory to programming course which all freshmen have to register where they learnt basic programming skills using Java. Flowers, et al observed that Cadets instead of focusing on problem solving they end up struggling trying to fix an error and therefore missing the aim of the assignments. Flowers, et al. conducted a survey among previous and current instructors where they identified 9

common errors mistakes students were making. Instructors observed that students keep making same mistakes each time and even got frustrated trying to figure out what an error message is, thereby wasting lots of time in trying to figure out how to fix a simple error. To help the students, a program that pre-processes students (cadets) source code called Gauntlet was developed in such a way that it explains in layman's terms each of the syntax errors a student encountered while doing his or her programming assignments. Gauntlet helps in explaining more in detail about the 9 identified errors by the past and present instructors and even gives suggestions on how to fix it.

Flowers, et al. concluded that Gauntlet has resulted in several tangible benefits for both students and instructors. It explains syntax error messages to students, and help in reducing the instructors load and save students from seeking assistance from their instructors. Gauntlet also provides students with ideas about typical causes of the errors. It deciphers error messages into simpler understandable way. Table 2.5 shows the 9 most common errors identified by the current and previous instructors in Flowers, et al. works.

Table 2.5 List of Errors identified in Flowers, et al Works
Errors Identified by the Instructors

Errors Identified by the Instructors	
	Errors
1	Mismatching curly braces
2	Mismatching quotations
3	Misplaced semicolon
4	Improper file name
5	Not initializing a variable before attempting to use it
6	Mismatching parentheses
7	Missing semicolon
8	Misspelling printLine method
9	Package does not exist

2.2.3 Automated Error Collection System (Custom-made IDE for Java)

Another interested study was conducted by Jackson, et al. [61] in the same military academy after instructors at the institution observed that [36] did not accurately identify the top errors when he did his survey [36].

Instructors observed that there was discrepancy between the errors identified by the instructors and the errors the students are still making. A further research was done by Jackson, et al. [61] which shows that Flowers, et al. [36] missed errors such as incompatible types, and syntactic errors such as missing “class” or “interface” keywords, and illegal expression statements. The authors and instructors felt that the students had good understanding of the learning objectives; they believed that lack of programming experience and simple syntax errors were primarily responsible for student errors [61].

Jackson, et al. used a real-time, automated error collection system that logged 100% of Java errors encountered by each student to a central database. This logging program captures the errors every time students compile code. The system collected a total of 559,419 errors over the course of one semester. It was found that the top ten errors accounted for 51.8% of the total errors made by the 583 students from across 11 faculties in the academy.

Jackson, et al. concluded that among the 9 identified errors by previous and current instructors [36] only five errors were also collected by the automated system. The automated system also verified the discrepancies between the errors identified by instructors [36] and the errors students are making [61]. Table 2.6 shows the top 10 errors collected in J. Jackson, et al studies using an automated error collector system.

Table 2.6 Lists of Errors Collected in Jackson et al Works

Jackson, et al. Errors Identify using Automated Error Collector		
Top ten	Name of Errors	Number of occurrence
1	Cannot resolve symbol	81655
2	; expected	47362
3	illegal start of expression	32107
4	class or interface expected	25650
5	< identifier > expected	25223
6)expected	21412
7	incompatible types	15854
8	Īnt	14185
9	not a statement	13878
10	}expected	12808

2.2.4 Observations (Practical sessions, Test and Exam)

In 2006, Pillay et al. [62] did a research studies to identify causes of students programming errors and the difficulties students come across while learning to program. The subjects in the research were students from two South African institutions, namely, University of KwaZulu -Natal and Mangosuthu Technikon. The participants investigated in this research were students enrolled in first year Java programming course. The courses in both institutions focused on the following programming concepts. 91 students from both institutions participated in the study. 61 students from University of KwaZulu –Natal and 30 students from the Mangosuthu Technikon. Data were collected in the form of Observations during practical sessions, Practical tests and examinations. Errors made by students and learning difficulties encountered during practical sessions were documented by lecturers and tutors. Also the programming errors made by students in tests and

examinations were traced and documented. Pillay et al. [62] classified the errors based on the following concepts: variables and typing, arithmetic operations, logical operations, input and output, conditional control structures, iterative control structures, and modularization. Errors for each of these topics include both syntax and semantic errors. Pillay et al. did not give more details on how the error was quantified.

Pillay et al. concluded that the main causes of errors made by students in the examination were a poor problem solving ability and the lack of knowledge and understanding of the programming language. Furthermore, students experienced the most difficulty with modularization and iteration. Table 2.7 shows the distribution of errors in the students' final practical examination based on the difficulties they experienced over different programming concepts.

Table 2.7 Difficulties Experienced by Students Programming
Difficulties based on Programming concepts by Pillay *et al*

	Programming Concepts difficulties	Percentage of Students
1	variables and typing	10%
2	arithmetic operations	3%
3	logical operations	11%
4	input and output	9%
5	conditional control structures	12%
6	iterative control structures	23%
7	Modularization	32%

2.2.5 Screen Recorder (C Programming)

At Texas A & M University – Kingsville Sevelia, et al. [63] conducted a research among the Under-Graduate students of Electrical Engineering and Computer Science students in order to know the barriers face by their students and suggest a remedy to those who needed assistance. There were 20 students who participated in the experiment. Students were given a programming assignments in which they have to complete in the lab.

Data is collected while the students are working on their programming assignment. Data is collected individually from each and every student through a screen recorder while they are coding and each student has an hour to complete their programming assignment. The assignments were based on the following C programming concepts: if and nested-if statements, Iteration concepts like using for or while or do-while, switch statements, files (manipulating data in files using C programming), functions (defining a function, calling that function and passing arguments to function, arrays (applying mathematical operations to arrays). The screen recording of each student is taken one at a time and properly analyzed, screen recordings of only 16 students were collected. The rest of the four students recordings were not properly recorded and was not used for the research purpose.

At the end of the experiment, it was observed that seven students out of sixteen students who participated in the experiment were able to complete their programming assignments successfully. Five students partially completed their assignment, and four students were in level zero and were unable to write the basic code. Table 2.8 shows the number of student and the barriers faced in the programming assignments.

Table 2.8 Lists of Barriers faced by Students

Barriers faced by students by Sevela, et al.		
Barriers	Number of Students facing Barriers	Total Number of Students Participated
Header files	4	16
Thinking about logic	4	16
Variables	3	16
Issues with printf & scanf	8	16
Zero level	4	16
Usage of Internet	8	16
If and nested-if	1	3
Iteration concept	3	3
Switch concept	3	3
Files concept	1	2
Function concept	2	3
Array concepts	0	2

2.2.6 Automated Error Collection System (JBuilder IDE)

In 2012 Mow, et al [64] conducted a research study at the National University of Samoa (NUS) to identify the common errors made by students in a Java programming courses and analyzed these errors to identify problematic areas which curriculum needs to focus on.

The study consisted of analyses of computer programs from 3 levels of Computer Science undergraduate programming courses HCS181, HCS281 and HCS286 at the National University of Samoa. The total numbers of participants from the 3 classes were 68 students. 25 from HCS181 students, 28 from HCS281 students and 15 HCS286 students were used in the analyses. Assessments for these courses were in the form of programming tasks, tests and home works.

The integrated development environment (IDE) used was JBuilder. Data were gathered from the assignment collected from the 3 classes, loaded into JBuilder and from compilation and running of the programs, the errors generated by the compiler were logged and categorized according to the type of errors that emerged.

The frequency distribution of programming errors for the 3 courses was generated. The errors generated across the 3 courses were combined into various categories. The percentages of the 8 most common errors are: variable not found (49.8%), identifier expected (14%), class not found (5%), mismatched brackets/parentheses (5.3%), invalid method declaration (3.7%), illegal start of type (3.4%), expected (2.2%) and method not found (1.6%). Table 2.8 shows the summary of the top 8 errors across the 3 classes and the occurrence frequency.

Table 2.9 Lists of Errors Collected in Mow et al Works

List of Errors collected by Mow et al. in National University of Samoa					
Programming Error	1st class	2nd class	3rd class	Total	% of Total
Variable not found	101	18	41	160	49.8
Identifier expected	23	22	0	45	14.0
Class not found	12	3	1	16	5.0
Mismatched brackets/parentheses	11	6	0	17	5.3
Invalid method Declaration	7	5	0	12	3.7
Illegal start of type	5	6	0	11	3.4
Method not found	4	1	0	5	1.6
Expected	2	4	1	7	2.2

2.2.7 Survey (Survey on International Opinions)

In 2005 another study was conducted by E. Lahtinen, et al. [6], based on an international survey of opinions from more than 500 students and teachers in evaluating the difficulties experienced when learning and teaching programming. A questionnaire was presented to BSc and MSc students who had experience in programming courses in 6 universities. It was advertised also to the teachers of the same universities teaching the programming courses. Responses were received from Fachhochschule Furtwangler (FHF, Germany), Reykjavik University (RU, Iceland), Tampere Polytechnic (TPU, Finland), Tampere University of Technology (TUT, Finland), Bucharest University of Technology (UTCB, Romania), and Ventspils University of Technology (VENTA, Latvia). The questionnaire was available for 10 days. In total, 559 students and 34 teachers answered the survey.

The most difficult issues in programming as reported by students were: (a) understanding how to design a program to solve a certain task; (b) dividing functionality into procedures; (c) and finding bugs from their own programs. The author made conclusion that these are all issues where the student needs to understand larger entities of the program instead of just some details about it. In addition, the most difficult

programming concepts were recursion, pointers and references, abstract data types, error handling and using the language libraries. It needs to be noted that, error handling requires understanding the program comprehensively. Using the language libraries requires independent searching of the information, which can make it difficult for the novices. Recursion, pointers and references, and abstract data types are abstract concepts and thus cognitively complex to understand without a similar phenomenon in the daily life for comparison. The teachers' opinions on the most difficult course contents were almost the same as the students. In addition, the teachers perceived understanding programming structures as difficult in issues about programming.

CHAPTER THREE

RESEARCH METHODOLOGY

In this chapter the methodology that was used in answering my research questions is discussed. An exploratory and participatory method was used which includes the subjects in the research, and the study procedure. How the data obtained was used to answer each of these research questions was discussed.

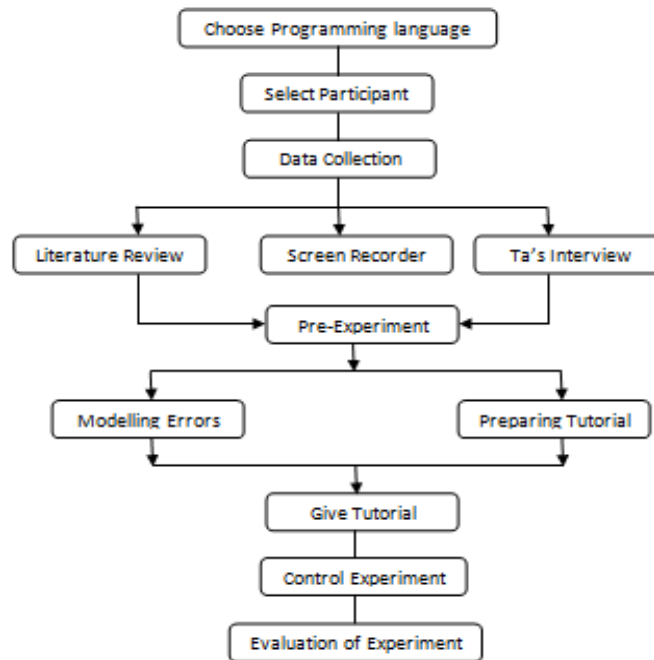


Figure 3.1 Overview of Steps followed

3.1 IDENTIFYING THE ERRORS

Prior to the experiments we started by asking ourselves how can we identify the common types of errors programmers are making.

In order to identify the errors novice programmers are making, an exploratory study was done which I first collected list of errors from the literature review. We also recorded the screen of the students while they are writing programs, so from the recordings we collected a list of errors also. We also interview the TAs after each and every lab session; we also document the errors that the TAs helped in identifying. All these information from the three sources was used to come up with a lab assignment (pre-experimental assignments) which was analyzed, from which a final error list students are making was formed. Figure 3.2 below shows the control flow of the three steps of error identification.

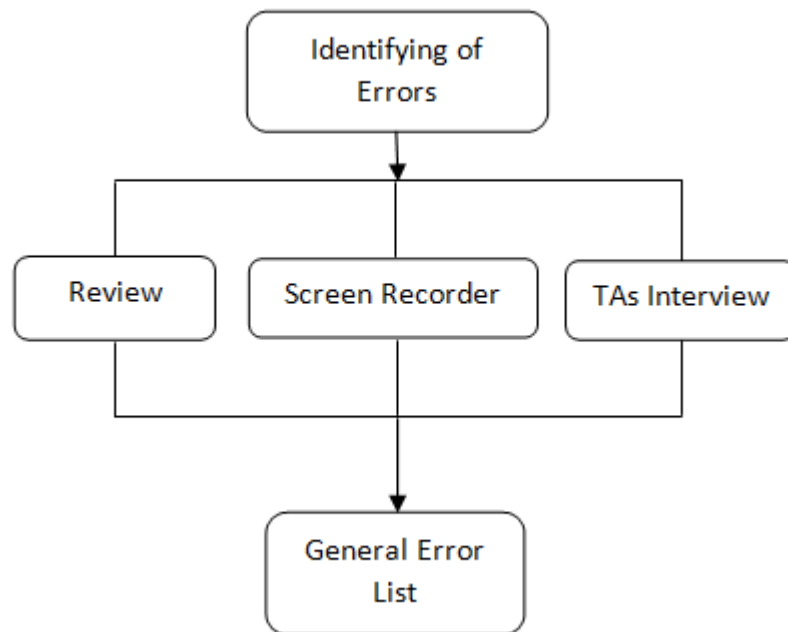


Figure 3.2 Errors Identification Steps

3.2 DATA COLLECTION TECHNIQUE

3.2.1 Participants

2013/2014 spring semester First year Computer Engineering students of Meliksah University were the subjects in this study. The first year students were enrolled in an introductory programming course CSE101 and CSE102 (CSE101 first semester and CSE102 second semester) which are compulsory courses for all first year Computer engineering students.

CSE101 is taken in the first semester of the session in which 45 students registered for the course but the attendance number was low during the lab sessions, but I think the reasons for the low number of attendance was that at the beginning of the semester the instructor given the course announced that “The students who take this class for the second time do not have to come to the labs if they want their previous year's lab grades to be used as their lab grades for this semester”. Most of the students enrolled in the course are taking programming course for the first time while others have programming experience because they are repeating the course for their second or third time. The topics covered in CSE101 include the fundamental basic concepts of programming which are; Variables and Types, Reading Inputs by using the Scanner class, Decision Making and Selection Statements, other topics include Switch statement, If else problem, Swap, Loops (while loops, for loops and Do while), Arrays and Methods.

CSE102 is a second semester course with 55 students registered for the course, the students in this course mostly took part in the CSE101 course i.e. have experienced in at least one programming course CSE101. The laboratory sessions in CSE102 have low number of attendance like the CSE101. The topics covered in this course include; Introduction to Object-Oriented Programming - Class - Object - Scope, Objects - static and non-static, Access Modifiers: public, private, package-specific, Encapsulation, Constructors, Classes as Data Types , Array of Objects, Inheritance, super, overloading, overwriting, Polymorphism, String class, Casting, Abstract classes and interfaces, Exception Handling

Our main concern in this research work is to identify the students’ common errors, by knowing their common errors we can provide a helpful solution to them. Our work will concentrate mainly on identifying the mistakes they are making and the difficulties they come across while working with the concepts in providing a solution to a giving programming task. The basic concepts are topics most covered during the first semester in the CSE101 and some part of the CSE102 topics. Our purpose here is that in knowing the problem the students are facing in the basic concepts, it would be easier to help them in providing a solution and help to those who need it most.

3.2.2 Choice of Integrated Development Environment (IDE)

Several factors need to be taken into consideration when choosing an IDE for a novice programming students. Based on a panel discussion of IDEs for first year programmers [65], it was stated that development environment for students should be simple, stable, and affordable. In addition, an IDE for novice should also have the features of: syntax highlighting, notifying of errors in case of it occurrence, code completion, debugger, should be able to support UML and should have the feature that allows the extension of plug-ins.

Pane and Meyers in their report on issues to be considered when designing an IDE for a novice programmer [65], Pane and Meyers made a suggestion for novice IDE include highlight on important information such as syntax, code formatting, immediate feedback, support for planning, consistency in notation, minimize working memory load, simple and useful error messages, support for testing and debugging, and documentation that describes everything a user needs to know about a system. Programming language can be taught in a variety of ways by an instructor, therefore there is no specific standard list of requirements for the development of IDE for novices.

The subject of this research works uses the NetBeans IDE, it was chosen by the university, it is believed to be an easy to use IDE for a beginner who is learning to program because of it easy to use interface and its drag and drop features.

3.2.3 Lab Sessions

Each week a laboratory exercise is assigned to the students related to the topics covered in the classroom during that week. The laboratory assignments are to be completed within three hours. The same operating system Mac OS is installed in the entire laboratory Computers, a recent version of NetBeans Java was installed on each of the Computers and also a screen recorder was installed on the computers that would be used for these experiments. In the lab each student sits in the ratio of 1 to 1 on a computer. During the lab assignments we uses the three different available methods: Review, Videos (screen

recorder) and TAs interviewing in the pre-data collection, from that we came up with a general error list.

The screen recorder program was installed on 15 of the laboratory computers in which the program is activated during the lab sessions while the students are working on their programming assignments. No external hardware is required for the screen recorder, as the software itself can record the screen of the novice programmers and the recorded data is saved in a video format.

The software used to record the computer screen was CamAsia. This software also helps in capturing screenshots automatically with a click of mouse. It helps us see the participants' characteristic as they write their codes. With this software, the collected data which was in video format can be exported to various other video formats like AVI, MP4.

3.2.4 Data from Reviews

In Chapter 2, it was showed that a lot of studies have been done in trying to identify the common errors novice programmers are making and the difficulties they are experiencing. So from the review we identify some common errors and come up with a list, which was also used in preparing of the questions to ask during the pre-experiment assignments.

3.2.5 Video (Screen Recorder)

Each week during the Laboratory session's students are selected at random to sit on the computers that has the recording software installed, and the recorder is then activated throughout the lab session. The Screen recorder serve two purpose after it helps in the identification of errors by reviewing the screen recordings. It also helps in identifying the characteristics and behavior of the students as they program.

Data in the form of video is collected individually each from the 15 of the lab computers installed with the recording program. The data collected is the screen recordings of the students.

3.2.6 Teaching Assistants' Interview

The TAs that supervises the students during the laboratory sessions were responsible for the grading of the students during the lab sessions. Each week after the laboratory session is over, I asked the TAs about the kind of errors and difficulties the students were facing while providing solution to their lab assignment. The errors and difficulties encountered during each practical session were documented.

With these data gathered from the reviews, screen recordings and the interviews of the teaching assistants, we came up with a unique classification of the errors based on programming concepts. The errors and difficulties the students are facing with the programming concepts was merged and grouped into 9 general categories E1 to E9, which were later sub-grouped into 14. The general error lists was listed in the related work section.

3.3 LAB EXPERIMENTS ASSIGNMENTS

Figure 3.3 shows the steps of the activity flow in the pre-experimental and the control experimental.

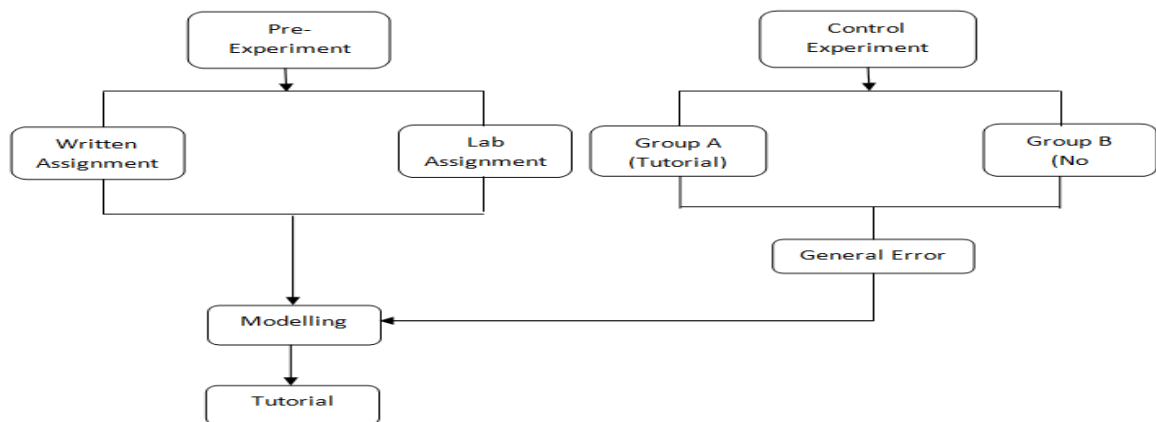


Figure 3.3 Control Flow of Experiment

Three assignments were organized, two of which were conducted before a tutorial was taught and the third one after the tutorial had been taught. One of the first two assignments that was conducted before the tutorial was done is a paper based (in written)

and the other was a computer based (lab assignment). The third assignment was conducted after a tutorial was taught which is also a computer based. For the sake of this study the main thing was that all analyzed programs covered the same programming constructs.

3.3.1 Pre-Experiments (Before Tutorial)

After the initial lists of errors was collected and documented from the three sources that were discussed in previous section above i.e. review, screen recorder and TAs interview.

Two pre-experiments assignment were conducted. The first one was in the form of writing (paper based) and the second one was using computer based. This was done to collect data for finding error types and analyzing modeling of the errors i.e. to find the positions of error occurrence and the number of frequency occurrence. This gave me ideas in preparing the tutorial and what kind of things to emphasis on.

These pre-experiments also helped in answering our first two research questions which are:

- What are the errors students are making?
- What causes those errors?

The results of the data modeling of the pre-experimental data were used in preparing of the tutorial contents i.e. the specific locations where most errors occurred was emphasis during the tutorial.

After the pre-experiments and the experiment questions were drafted, two professors validated the appropriateness and relevance of the questions. These two professors are from the Computer Engineering departments.

3.3.1.1 Writing Assignment (Pre-Experiments)

A writing paper based assignment was conducted in the form of a quiz and midterm exam. The questions asked consist of the basic concepts such as looping structure, selection

structure/nested if, array structure, method/function and Boolean structure. The questions asked in the writing based assignments consisted of the following: a questions was asked to test between students' knowledge of conversion within a loop structure (for-loop to while-loop) to identify the errors the students are making while converting within the loop structure, further more a code fragments containing an error was given to the students for them to trace, identify and fix, additionally a code fragments was given for the students to predict the correct output. Questions related to declaration, assignments and looping within an array index were also asked, students were tested and the errors related to it were identified.

In the written paper based assignment, the questions were given to test if the participants can read and understand a given code fragments and predicts the output. In a given code, the steps the compiler follows in printing out results, also syntax concepts like missing semi colon, method calling and parameter passing, looping concepts, and array usage were also tested. The details analysis of the written assignments is in Appendix C.

3.3.1.2 Lab Experiment (Pre-Experiments)

The first computer based experiment (pre-experiments) was performed in the lab during the lab session; it was done with the laboratory computer. It was composed of 10 questions that test the participants knowledge of the basic concepts of Java programming which include looping structure, array declaration and assignments of value, nested if and if then else, also include method calling and objects. The assignment was to be completed within one hour thirty minutes.

During the lab session the experiment questions were uploaded into the course website (Module) where the participants could login with their ID's to download the question. The time period for the assignment was 1:30minutes, after the participants had finished, they were required to copy the code fragments from the IDE NetBeans and paste in the space provided in the word document question template. A place was provided in the Module where each participant would upload his or her code for analysis.

35 students participated in the first computer based assignments, so each submitted file was renamed A1 to A35 based on the participants time of submission, it was assumed that participants that finished early became A1 followed by next A2 until the last participants submitted.

Each of the submitted file is downloaded and was analyzed based on the naming from A1 to A35, question number 1 was first analyzed for all the 35 participants followed by the 2nd question until the last question of the last participant was analyzed.

Each answer provided by a participant is copied and pasted into a NetBeans java for analyses. Every code for each question was carefully traced and checked in NetBeans IDE. If there is a syntax error as the code is pasted into the IDE, the error is indicated by the compiler and was documented.

If the code is syntax error free, then the code is compiled. If the required output is not produced then a logical error has occurred in the code implementation. If the code halts in the process of running, a runtime error has occurred, and then we try to trace the code in order to find which part is missing. Each error based on its concepts is properly documented. We repeat the procedure for all the questions.

After the questions were drafted, they were sent to two professors to inspect and make corrections if any. In addition, the questions were checked whether the concepts we are looking for were satisfied or not.

An instruction was given to the participants that at the end of the lab assignment each student should upload his or her code in the course Module. After all the data were gathered, the most common errors that students made frequently were extracted and analyzed.

3.3.2 Tutorial

40 students participated in the experiments. The participants were grouped into 2 (group A & group B) based on their GPA performance in their first semester programming CSE101 exam score. The sum total GPA for both group participants is 65.50, in which

group A compose of total GPA of 32.50 and group B 33.00. The average GPA for the whole participants is 1.64, and the average GPA for group A participants is 1.63 and for group B participants is 1.65. The details of the GPA are shown in Appendix C.

Group A was taught a tutorial about the common errors we found they were doing and how to carefully avoid it. Group B was not taught anything regarding the errors. So the same assignment was given to both groups and was carefully analyzed to see if there would be differences in the errors both group were making.

The tutorial was to be done within 2 hours of 2 sections, the first section 45minutes followed by a 15 minute break and then the second section which was also 45minutes.

First, I was introduced to the participants at the beginning of the tutorial and the purpose of the tutorial was announced to the participants. In order to motivate the students, I started by introducing error, how a simple error can lead to a loss of billions of dollars in some cases even loss of life. The over view of the tutorial outline is as follows: the 1st session covered the following area: wrong nested IF statement, wrong matching of { } curly braces and its effect while coding (IF and loop structure), effect of wrong placement of the printout statement in a LOOP structure, missing increment/decrement statement and wrong placement in a LOOP. While the 2nd session covered the following area: omitting of [] and wrong usage of “[]” instead of “{ }” in one line array declaration & assignment, off by one error done in a loop structure and array out of bound error (wrong usage of termination statement while looping in an array), object creation and usage and accessing a non-static method and how values can be swapped. Appendix D shows some of the contents in the tutorial.

3.3.3 Control Experiment (After Tutorial)

The purpose of the second experiments was to compare the performance between the two groups (those that undergone a tutorial (group A) and (group B) those that does not) and find which of the two groups has less errors occurrence, if the tutorial makes any positive impact to the students that undergone a tutorial.

The second lab experiment was also done during the laboratory session, same procedure was followed as in the 1st experiment, but this time around the participants were 40 in number for the experiments.

Prior to the experiment the participants were grouped into 2 groups (group A and group B), in each group there were 20 students. The group A participants were taught a tutorial about some basic difficulties/errors that students are mostly experiencing while writing a program and how to overcome it while the other group B were not, so the 2nd lab assignment was given to both group and the results of both group was collected and analyzed carefully same way the 1st experiments was collected and analyzed. The naming strategy for the 1st assignment was retained for the 2nd assignment except that 5 participants were included in the second lab making the total participants 40 as compared to the 1st experiment. All the data was carefully analyzed and documented.

CHAPTER FOUR

DATA ANALYSIS AND MODELLING

In figure 4.1 below, is the control flow of the data analysis, which shows that while the data from the pre-experiments was model at the same time the tutorial preparation was done.

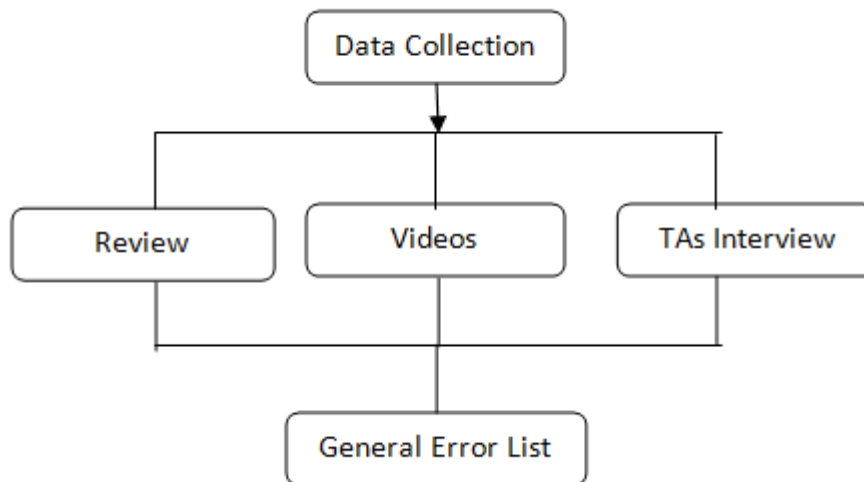


Figure 4.1 Control Flow of the Analysis

4.1 DATA ANALYSIS

The data analysis for the pre-experiments was done in two steps; the high level analysis and low level analysis.

4.1.1 High-Level Analysis

The high-level analysis was done to identify the error types in a general form i.e. syntax, run-time and logic error. In the high level analysis I only focused in identifying the main error types. Which was the first question in our research work; what are the main error types.

4.1.2 Low-Level Analysis

The low-level analysis involves details analysis and modeling of the data from the pre-experiments. The low-level analysis identifies the specific locations in a program where the errors occurred. It was information collected from the low-level analysis was used to create a tutorial for the students. It was used to answer our second research question that ask about what causes the error, more on the low-level analysis was discussed in the next session (modeling).

4.2 DATA MODELING

This section describe the locations and causes of the error, it was used to answer the question that asked the causes of the errors.

The data collected was analyzed. The lab assignment of each student is taken one at a time and properly analyzed. Each student's code was followed carefully, and the occurrence frequency of each error found was documented. Each code was pasted in a NetBeans IDE where it was carefully tasted for an error occurrence. Once the code is compiled, the compilation time error is found. If the program output is wrong, the causes of the wrong output was checked, identified and documented. All the issues faced by the students were carefully observed and documented.

4.2.1 Multiple if without 'else' concepts (if-then-else/nested if-then-else conditional control flow statements)

A *if* control statement has two parts/locations for possible errors:

1. Test/Conditional (Logical) Expression
2. then/else Statement

Figure 4.2 shows possible location in an “if” selection statement possibly error can occur.

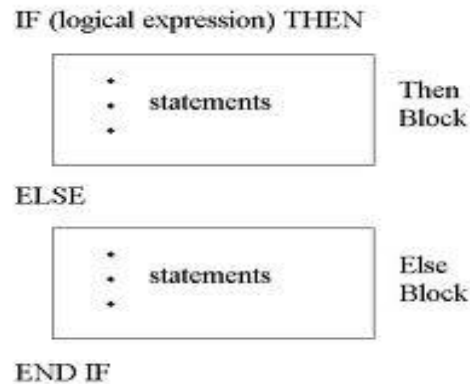


Figure 4.2 Positions of Error Occurrences

Logical-expression: The if-else statement depends on a boolean expression, if a boolean condition evaluates true, the immediate statement after the boolean expression is executed. If the boolean condition evaluates false, the else part is executed. Wrong boolean expression is a common issue for new programmers. So we carefully traced and documented the errors.

Missing else-if statements: It is found that novice programmers are omitting else-if where it is needed or are using else without if. It was traced and documented.

Forgetting necessary braces: The braces can be omitted if the block contains a single statement. However, forgetting the braces when they are needed for grouping multiple statements is a common programming error.

- **E1:** multiple if without else, if-else is used instead of an else in the last option (logical error)

```

public class studentsgrade {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter the student's Score");
        int grd;
        grd = input.nextInt();
        if (grd >= 90) {
            String a = "A";
            System.out.println("" + a);
        }
        if (grd >= 80) {
            String b = "B";
            System.out.println("" + b);
        }
        ⇒ if (grd >= 70) {
            String c = "C";
            System.out.println("" + c);
        }
        ⇒ if (grd >= 60) {
            String d = "D";
            System.out.println("" + d);
        }
        ⇒ if (grd >= 40) {
            String e = "E";
            System.out.println("" + e);
        } else {
            System.out.println("F");
        }
    }
}

```

Figure 4.3 usage of multiple if

In Fig. 4.3 above indicated by a pointed arrows shows how a student writes a code that reads in the scores of students and prints out their grade but uses multiple if-if in all the conditions and omits the else-if where it is needed. The program is to print only one statements that correspond to the inputted score value. In the case where else-if is missing the program might print more than one statements for some certain input.

4.2.2 Loop Concepts (within Array and Non-array)

If a statement is to be executed or repeated more than one time, a loop is recommended. Loop allows a programmer to execute a statement in a number of times using a short code fragment. A body of loop is executed only if the statements in the Boolean expression are satisfied. The statement executed inside a loop is called a loop body; the Boolean condition must evaluate true before the statements in the loop body is executed. Each execution of a loop body is referred to as iteration.

Three types of loops come under the loop concepts, but most of the participants in the experiments use the FOR and WHILE LOOP:

There are 4 possible positions in a for-loop which an error can occur. The positions were carefully traced, analyzed and documented. The possible locations are indicated with pointed arrow in figure 4.4 as shown below:

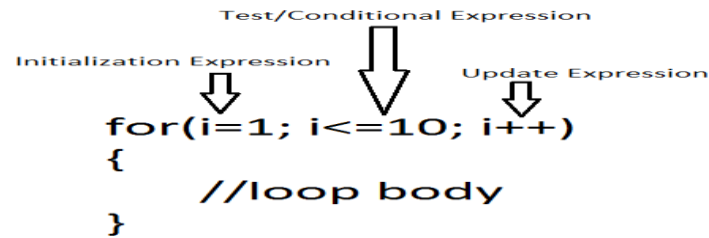


Figure 4.4 possible locations of errors

A *for* loop has four parts/locations for possible errors:

1. Initialization Expression
2. Test/Conditional Expression
3. Update Expression
4. Loop Body

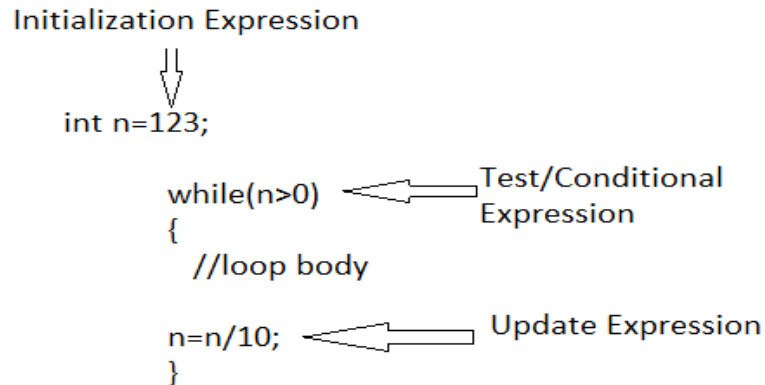


Figure 4.5 possible locations of errors

A *while* loop has four parts/locations for possible errors:


1. Initialization Expression
2. Test/Conditional Expression
3. Loop Body
4. Update Expression

In figure 4.5 the arrow shows the possible places where an error can occur in a while loop: so I traced the code collected from the participants based on the steps below;

- i. **Initialization Expression:** Incorrect initialization of a variable that is used inside the loop condition is a common mistake made by a novice programmer. Remember that the condition is first checked before the loop body is executed; if the loop is initialized wrongfully the output result will be wrong, so we traced and document the issues here. Example: when writing a program to display the numbers between 1 to 10 i.e., excluding 1 and 10. Instead of initializing the starting point to 2 it was initialized to 1 as indicated in a yellow marker in the example below. e.g. E2.1

E2.1: (off by one) loop initialization/termination (logical): it happens when an iterative loops one more time or too many times. This problem arises when a programmer makes mistakes such as using “less than or equal to” instead “less than” should have been used in a loop condition or fails to take into account that a sequence starts at zero rather than one. Exceedingly common error is induced in many ways, such as by starting at 0 when you should have started at 1 or vice-versa.

- ii. **Test/Conditional Expression:** This part contains a Boolean expression. The Boolean expression is evaluated immediately after the initialization condition is checked and it is done at the beginning of each iteration. If this condition is true, the loop body is executed. If it is false, the loop terminates and the program control turns to the line following the loop, i.e. the code fragments below supposed to prints number from 1 to 10 instead it prints from 1 to 9 because < was used instead of <=. In this case a logical error occurs where a wrong output will be printed.



```
for (int i = 1; i < 10; i++) {  
    System.out.println(i);  
}
```

Figure 4.6 Wrong termination condition

While looping within an array index and a wrong termination condition is used, it may lead to a run-time error. For example, in figure 4.6 above the program tries to loop beyond the size of the array, which the program does not allow by terminating itself and displaying a run-time error message e.g. E4.2

- iii. **Update Expression:** this is the statement that updates the control variable. This statement is executed after each iteration, it increments/decrements the control variable depending on the problem solved. At a certain time, the value of the control variable should force the loop-continuation-condition to become false; otherwise, the loop will not terminate and it becomes an infinite loop.
- iv. **Loop Body:** here lots of errors can come in depending on the programmer understanding of the problem. It can be from wrong arrangement of the statements to be executed to omission of programming syntax like missing semi colon. Participants were also tested on accumulation of results while looping, and we found the weakness of the participants and documented (each iteration should be accumulated with previous results, override and updated) as showed in figure.4.7 shows usage of wrong accumulation of values in a code fragments.

- **E2.2: Incorrect Accumulator usage (logical error)**


```

12 public class Sum {
13
14     public static void main(String[] args) {
15         Scanner key = new Scanner(System.in);
16         System.out.println("Lütfen bir sayı giriniz (soru2 için)");
17         int n = key.nextInt();
18         int sum = 0;
19         for (int i = 1; i <= n; i++) {
20             sum += sum;           //sum+=i;
21             System.out.println(sum); //wrong placements of printout
22         }
23     }
24 }

```

Figure 4.7 Incorrect accumulator usage

In figure 4.7, the program is supposed to find the sum of values from 1 to the value a user enters. The programmer uses `sum+=sum` instead of `sum+=i`; this is a logical error because the program does not indicate that something is wrong with the program, unless after it was executed and the results produced was wrong, that is when a programmer would know that something is wrong with the code. This program in fig 4.7 when executed would print the value zero if executed with any input value. This kind of issues were traced and documented.

4.2.3 Method Concepts

When dealing with methods Novice programmers are found to be making these common mistakes listed below:

- i. **Static and non-static modifier:** here we trace the static or non-static keyword use in the method; for non-static we traced to see if an object has been created first before invoking the method, and was documented.
- ii. **Return value type:** we trace to see if the return value type in the header matches the return statement type.
- iii. **Parameters of the method:** here we trace if the calling parameters invoke from the main methods matches the parameter header type.

All the errors were carefully noted and documented.

- **E3.1: Type Incompatibility i.e. passing wrong arguments (syntax error)**

Evidence is found that novice programmers are declaring method parameters as a type and attempts calling it with a different type. This was also traced and documented. Figure 4.8 the pointed arrow shows how a method was call with an incorrect parameter.

```

9   public class sum {
10
11  public static int sum(int n1, int n2) {
12      int result = 0;
13      for (int i = n1; i <= n2; i++) {
14          result += i;
15      }
16      return result;
17  }
18
19
20  public static void main(String[] args) {
21      System.out.println(" " + sum(1.2, 10));
22  }
23  }

```

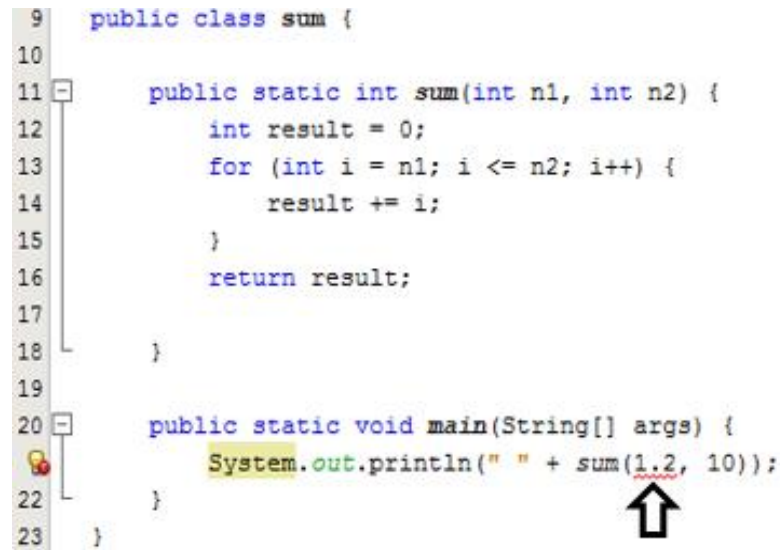


Figure 4.8: Incorrect parameter match

- **E3.2: Forgot to create object in accessing a non-static method**

Novice programmers are found to be making the mistakes of calling a non-static method directly without creating an object. An object is needed to be created in passing values to a non-static method. This was also traced and documented. Figure 4.9 shows how a participant tries calling a non-static method without creating an object.

```

15  public class printmessage {
16
17  public void nPrintln(String message, int n) {
18      for (int i = 0; i < n; i++) {
19          System.out.println(message);
20      }
21  }
22
23  public static void main(String[] args) {
24      nPrintln(("Hello"), 5);
25  }
26  }
27  }

```

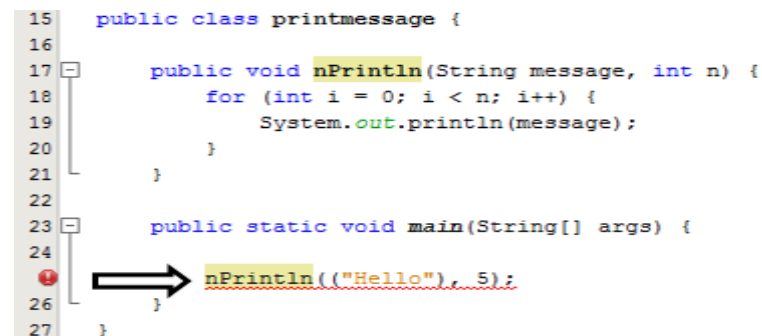


Figure 4.9: forgot to create an object

4.2.4 Array Concepts

Novice programmer experiences some difficulties in array usage; the following locations were traced for the possible occurrences of errors. The possibility of wrong array declaration was traced, and in the case of wrong array one line creation and initialization was also traced.

- **E4.1: Missing []/{} in Array (runtime error)**

```
public class DisplayMessageLoop {
    public static void DisplayMessageLoop() {
        int myList = new int[10];
        Scanner input = new Scanner(System.in);
        int sum = 0;
        for (int i = 0; i < myList.length; i++) {
            myList[i] = input.nextInt();
            sum += myList[i];
        }
        System.out.println("the sum is " + sum);
    }
}
```

Figure 4.10 Missing [] in an array

Figure 4.10 above shows a code fragments, the arrow indicated how a square [] bracket was omitted in an array declaration.

Furthermore novice programmers are found to be making mistakes in a one-line array declaration, creation, and initialization. Usage of {} instead of [] in array declaration or using [] instead of {} in a one line array declaration and assignment.

```
11 public class SumOfArray {
12
13     public static void main(String[] args) {
14         int i ;
15         int [] say = [1,3,5,9];
16         int total = 0 ;
17         for (i = 0; i < say.length; i++) {
18             total +=say[i];
19         }
20
21         System.out.println(say[i]);
22     }
23 }
```

Figure 4.11: Usage of [] instead of {}

In line 16 of the code fragments in figure 4.11 above. It shows how the participants tries assigning values to the array index but uses brackets [] instead of curly braces {}.

- **E4.2: Array Index out of Range (runtime error)**

```

13 public class SumOfArray {
14
15     public static void main(String[] args) {
16         Scanner in = new Scanner(System.in);
17         int[] array = new int[4];
18         int sum = 0;
19         for (int i = 0; i < 5; i++) {
20             System.out.println("Enter the numbers");
21             array[i] = in.nextInt();
22             sum += array[i];
23         }
24         System.out.println(" " + sum);
25     }
26 }
27
28

```

Output - 1st lab assignments (run)

```

> Enter the numbers
> 5
> Enter the numbers
> 7
> Enter the numbers
> 9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at stlabassignments.SumOfArray.main(SumOfArray.java:21)
Java Result: 1
BUILD SUCCESSFUL (total time: 9 seconds)

```

Figure 4.12: Incorrect termination condition

This type of error is classified under run-time error. The compiler does not indicate any sign of missing syntax or problem but after the code is compiled the program halts before finishing executing. The program tries to loop beyond the array index created. This kind of problem can also occur in trying to assign a variable to an index that was not created. Line 19 of Figure 4.12 above shows how an incorrect termination condition statement was used.

4.2.5 Incorrect placement of Printout statement and Missing Printout statement

The printout statement needs to be placed in a correct order to display the correct results in a proper way. Novice programmers are found to be placing a printout statement in wrong position i.e. before or after the statement it supposed to follow or even outside a curly braces when it is supposed to be inside. So care needs to be taken in placement of printout, a wrong results would be printed. In some cases novice programmers are found to

be omitting the printout statement even though the code is correct but the result would not be displaying due to the printout statement has been omitted. This issue related to printout is grouped under a logical error, e.g. the printout is placed immediately after the loop so for each iteration it prints each sum.

- **E5.1: Missing Printout statement (logical error)**

```

5      public class Sum {
6
7      public static void main(String[] args) {
8
9          int[] myList = {1, 3, 5, 7, 9};
10         int total = 0;
11         for (int i = 0; i <= 5-1; i++) {
12             total += myList[i];
13         }
14     }
15 }
16

```

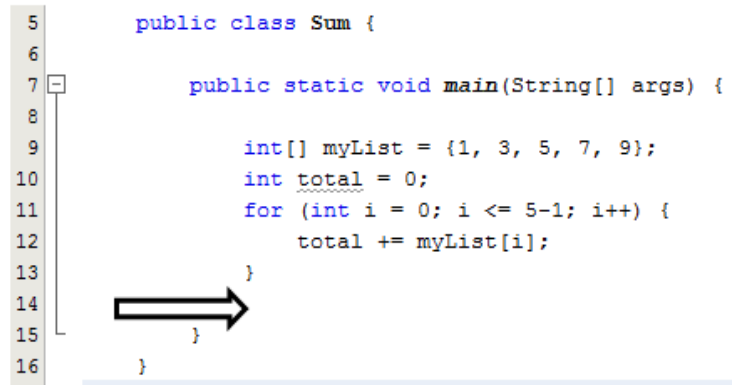


Figure 4.13 Missing Printout statements

Figure 4.13 shows a code fragment, a pointed arrow indicating a missing printout statement.

- **E5.2: Incorrect placement of Printout statement: (logical error)**

```

5      public class Sum {
6
7      public static void main(String[] args) {
8
9          int[] myList = {1, 3, 5, 7, 9};
10         int total = 0;
11         for (int i = 0; i <= 5-1; i++) {
12             total += myList[i];
13             System.out.println("the sum is " + total);
14         }
15     }
16

```

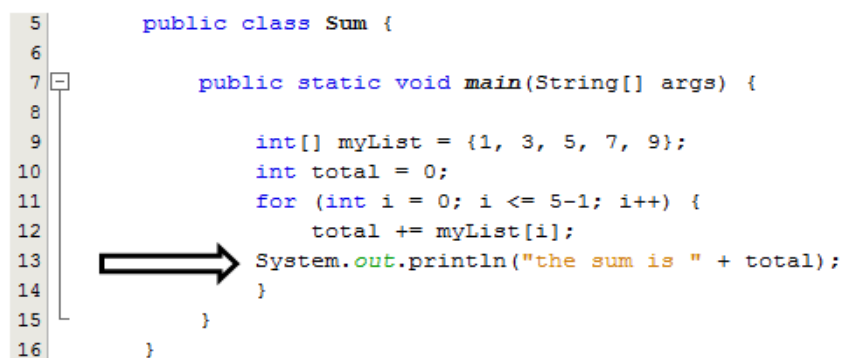


Figure 4.14 Incorrect printout placements

Figure 4.14 shows a code fragments showing an incorrect placement of printout statement position.

4.2.6 Code outside Method/Main method

- **E6: Code outside a method/main method class (syntax error)**

In this case the code was written outside the main method or outside a method just inside the class definition. In order for a code to be executed, it has to be written inside the main method or inside a separate method and called from the main method. Novice programmers are found to be writing code outside the main method. All these were traced and documented.

4.2.7 Missing Syntax

- **E7: Missing Semicolon/Parentheses: (syntax error)**


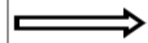
Novice programmers are prone to omitting important programming syntax like; omitting a semi-colon (;) at end of printout statement, additional/omitting of curly braces ('{', '}') in a place where it does not fit to be. So all these were traced during the data analyses modeling.

4.2.8 Keyword mistype and usage of uninitialized/undeclared variable

- **E8.1: use of uninitialized/undeclared variables: (runtime error)**

Novice programmers are found to be using a variable without declaring or initializing it, so we traced issues of using a variable without declaration and documented the results, e.g. in a for loop where the programmer forgets to declare it in or outside the for loop as shown in fig. for (i=0;i<100;i++)

```

6   public class DisplayMessageLoop {
7
8       public static void main(String[] args) {
9           // write your code below
10
11            for ( i = 0; i < 100; i++) {
12              System.out.println("welcome to java");
13          }
14      }
15  }
16  }

```

Figure 4.15 Usage of undeclared variable

Line 11 of the figure 4.15 shows variable `i` which was initialized and used without declaring it. So, novice programmers are found to be making this mistake. We traced issues like this and documented it.

- **E8.2: keyword mistype (syntax error)**

Novice programmers are found to be omitting keyword where it is necessary .i.e. missing keyword while creating scanner or using a small letter instead of a capital in writing Scanner. Also using small letter in writing the “s” in “system printout” or misspell of the length keyword in an array. All these were traced and documented.

```
6   public class DisplayMessageLoop {
7
8
9
10
11  public void main(String[] args) {
12      // write your code below
13
14      for (int i = 0; i < 100; i++) {
15          System.out.println("welcome to java");
16      }
17
18  }
19  }
```

Figure 4.16 Missing keyword

In line 11 of fig 4.16 above, the students omit the keyword `static` in trying to write the main method, the compiler does not indicate any problem in the code but when trying to execute the code, the code would not work and would only be alerting the programmer that no main method in the program. All programs start execution at the main method so every program in order to execute it has to have a main method.

CHAPTER FIVE

RESULTS AND DISCUSSION

5.1 PRE-EXPERIMENT ASSIGNMENTS RESULTS

The pre-experimental assignment was done in order to have more ideas on the error type the students were making and to have ideas of the critical point to emphasis on while doing the tutorial. All the data for the pre-experimental was carefully analyzed and modeled.

In the pre-experiments we were able to identify 14 common errors our students are making. Table 5.1 below shows the errors based on a ranking order from the highest occurrence frequency to the lowest.

Table 5.1 Lists of Errors from Pre-experiment

HIGH-LEVEL ANALYSIS FOR PRE-EXPERIMENTS RESULT				
RA NK	CODE	General form (high-label analysis)	Number of occurrences	Frequency occurrence %
1	E2.1	Off by one error (logical error)	50	18.12
2	E2.2	Incorrect accumulator (logical error)	40	14.49
3	E1	Multiple if without else (logical error)	38	13.77
4	E4.2	Array index out of range (runtime error)	31	11.23
5	E5.1	Missing print out (logical error)	23	8.33
6	E8.1	Use of initialized/undeclared(variables or mistype: (runtime error)	19	6.88
7	E4.1	Missing []/{ } in array (runtime error)	17	6.16

8	E3.2	Forget to create object in accessing a non-static method (runtime error)	15	5.43
9	E9	Incorrect swapping (logical)	14	5.07
10	E5.2	Incorrect printout placement (logical error)	9	3.26
11	E7	Missing semicolon/parentheses: (syntax error)	8	2.90
12	E8.2	keyword mistype: (syntax error)	8	2.90
13	E3.1	Type incompatibility (syntax error)	2	0.72
14	E6	Incorrect written of code outside a method/main method class (syntax error)	2	0.72
Total			276	100%

E2.1 Off by one Error (logical error)

The frequency occurrence table showed that the error with the highest occurrence is “off by one” with the frequency counts of 18.12%. After a low-level analysis was conducted on the “off by one” it was observed that 13.04% of the errors occurred due to wrong usage of the termination statements condition in a loop and 5.07% of the error occurred due to wrong initialization of a loop. Mostly either the loop will be looping more than its suppose to or will not reach as wanted, this kind of error is classified as a logical error because the compiler would display a wrong output due to it. The low-level analysis shows that most of the “off by one error” was caused due to an incorrect usage of termination statement in a loop.

E2.2: Incorrect Accumulator (logical error)

The second highest error was “Incorrect accumulator” with the occurrence frequency of 14.49%. This is mostly cause while working with a loop, in each loop iteration the value outputted is temporary stored and accumulated (sum) with the previous

iteration value, so students uses wrong accumulated variable, which results in printout of wrong result.

E1: Multiple if without else (logical error)

The analysis shows that the third highest error occurred was “multiple if without else” with the percentage frequency occurrence of 13.77%. This occurred where the students uses multiple “if” statement without “else”.

E4.2: Array Index out of Range (run-time error)

“Array index out of range” accounted for about 11.23% of the errors. Low-level analysis showed that, the error occurred due to the usage of incorrect termination condition statement in an array. This is mainly caused by trying to access beyond the defined boundary of an array size.

E5.1: Missing Printout statement (logical error)

The fifth most error accounted for about 8.33% percentage is “missing printout statement”. Analysis shows that students will write a correct code but lack printout statement, so the program after compiled will not display result because the printout statement was accidentally omitted.

E8.1: Use of uninitialized/undeclared variables

The sixth most error on the list is “Use of uninitialized/undeclared variables” with the occurrence frequency of 6.88%. This error usually occurred if students attempts to use a variable while it was not declared, or in the case that it is declared but not initialized in case of a local variable, i.e. forgotten to declare the variable that was used, while iterating in a loop did not initialize the variable that will be holding the sum. In Java, all local variables must be initialized after they are declared but instance variable does not have to be initialized because it has a default initial value when declared in a class.

J. Jackson, et al. [61] believed that it occurred as a direct result of high school algebra, declaring a variable is foreign to most novice programmers.

E4.1: Missing []/{} in Array (run-time error)

“Missing []/{} in array” accounted for about 6.16% of the top 14 errors uncounted. It occurred due to students forgotten to include [] while declaring an array.

E3.2: Forgot to create object in calling a Non-static method (runtime error)

“Forgot to create object in calling a non-static method” the percentage occurrence of this error was 5.43%. An object is required in calling a non-static method.

E9: Incorrect Swapping of values (logical)

The error “Incorrect swapping” accounted for about 5.07% of the error occurred, this was caused due to the Wrong assignments of variables in swapping of two values i.e. $x=10; y=20; x=y; y=x$.

E5.2: Incorrect Printout placement (logical error)

The error “Incorrect printout placement” accounted for about 3.26% of the total error encountered. This is caused by students placing the printout statements in a wrong order which may lead to producing a wrong output.

E7: Missing Semicolon/Parenthesis (syntax error)

The error “Missing semicolon/parenthesis” accounted for about 2.90% of the error encountered in the pre-experiments, in java each statement has to end with a termination statement which is a semi-colon, so if a termination statement is missing the compiler usually give a warning. This was caused by careless of the students. Jackson, et al. [61] noted missing “;” error was anticipated because of the fairly rare use in the English language.

E8.2: keyword mistype (syntax error)

“Keyword mistype” also accounted for about 2.90% percentage of the error, this is a typographic error, mostly caused when student is in a hurry and eager to rush and finish the program coding, in the process he or she might type or strike a wrong key.

E3.1: Type Incompatible (syntax error)

“type incompatible” the percentage occurrence of this error is 0.72%, this was usually caused due to students passing a wrong variable type in a method calling, which causes mismatch between the declared method variable type and the variable used in calling the method.

E6: Code outside a Method or outside a Main method (syntax error)

The error with the least occurrence was “code outside a method or outside a main method” the percentage occurrence frequency of this error was 0.72. This was caused by students writing the code outside the main method. This shows that less of this error is made by our students.

5.2 CONTROL EXPERIMENT ASSIGNMENT RESULTS

The control experimental assignment (after a tutorial) was done in order to compare between the two groups: the group that under gone tutorial (group A) and the group that has not under gone (group B). All the data for the experiment (2nd assignment) was carefully analyzed and modeled. Table 5.2 shows the result of the control experiment.

Table 5.2 Control Experimental Results 1

Errors	Group A	Group B	Total Error
E1: Multiple if without else (logical error)	21	16	37
E2.1: Off by one (logical error)	16	26	42
E2.2: Incorrect accumulator (logical error)	6	1	7
E3.2: Forgot to create object in accessing a non-static method (runtime error)	3	7	10
E4.1: Missing []/{ } in array (runtime error)	6	10	16
E4.2: Array index out of range (runtime error)	9	18	27
E5.1: missing print out (logical error)	9	13	22
E5.2: Incorrect printout placement (logical error)	4	7	11
E6: written of code outside a method/main method class (syntax error)	2	15	17
E9: Incorrect swapping of values (logical error)	5	8	13
TOTAL	81	121	202

The result of the analysis in the comparing of the two groups shows that there is a little significant improvement with the group that undergone tutorial. 40 students participated in the control experimental, 20 from each group. The results of the control experimental analysis shows that out of the 14 identified errors from the pre-assignment (before tutorial) only 10 were repeated, 4 of which were first identified from the pre-experiment did not occurred in the control experimental assignment (after tutorial). The four errors that were not repeated during the experimental exercise were:

- “E3.1: type incompatibility”
- “E7: missing semicolon/parentheses”
- “E8.1: use of uninitialized/undeclared variables”
- “E8.2: keyword mistype”

Analysis shows that all the four errors that were not repeated by both groups in the control experiment (after tutorial) are syntax error. This study is consistence with that of Mow et. al. [64] that says most of the syntax errors are caused by careless of a programmer, this shows that students were more careful in the control experiment (after the tutorial) than in the pre-experimental assignments when dealing with the programming syntax.

Analyses shows that there are improvements in these 8 error types; E2.1, E3.2, E4.1, E4.2, E5.1, E5.2, E6 and E9. On the other hand, there is no improvement in E1 and E2.2 error types.

The total number of errors collected was 202, out of which 81 came from group A (undergone tutorial) which accounted for 40.10% of the total errors while 121 came from group B (without tutorial) which accounted for 59.90% of the total errors. Below is the description of the percentage occurrence of the 10 errors that occurred in the control experiment.

- **E1: Multiple if without else (logical error)**
- **E2.1: Off by one error (logical error)**
- **E2.2: Incorrect accumulator: (logical error)**
- **E3.2: Forgot to create object in accessing a non-static method (runtime error)**
- **E4.1: Missing []/{} in array (runtime error)**
- **E4.2: Array index out of range (runtime error)**
- **E5.1: missing print out (logical error)**
- **E5.2: Incorrect printout placement (logical error)**
- **E6: Written of code outside a method/main method class (syntax error)**
- **E9: Incorrect swapping of values (logical error)**

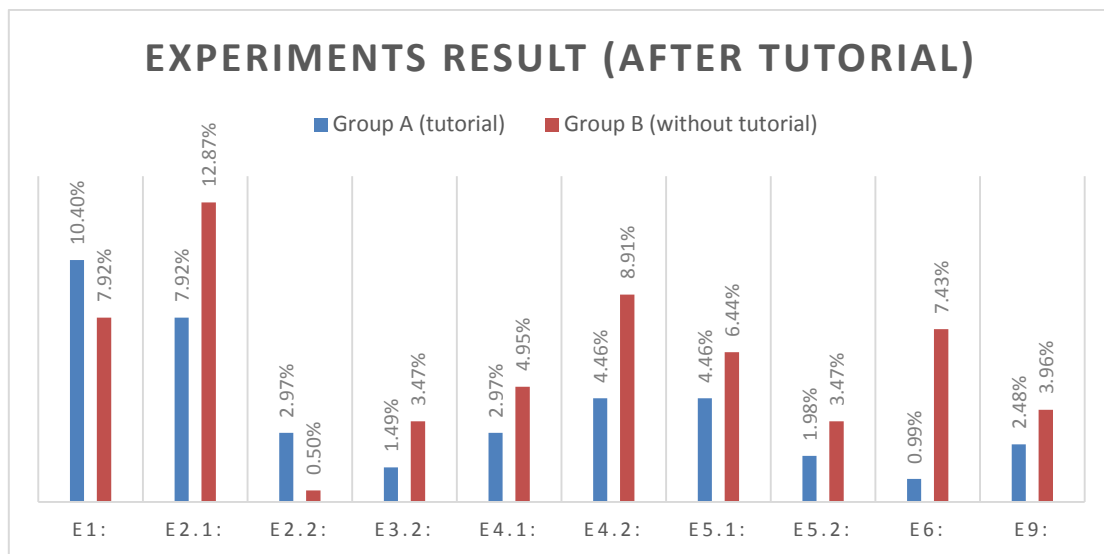


Figure 5 Control Experiment Results

- **E2.1: Off by one:** The error type “E2.1: Off by one” occurred 42 times, which is equivalent to 20.79% of the total errors encounter in the control experiment. In which 16 was collected from group A (with tutorial) which accounted for 38.10% of the error type while group B (without tutorial) with the frequency occurrences number of 26 accounted for 61.90% of the error type. Analysis shows that the students that undergone a tutorial makes less error of this type.
- **E4.2: Array index out of range:** The error type “E4.2: Array index out of Range” occurred 27 times. Group A (with tutorial) has a frequency occurrences of 9 which accounted for about 33.33% of the error type while group B (without tutorial) has the frequency occurrences of 18 which

accounted for 66.67% of the error type. Analysis shows that students participated in the tutorial did less of this error type.

- **E5.1: Missing Printout statement:** The error type “E5.1: missing printout statement” occurred 22 times (10.89%). It occurred 9 times in group A, which accounted for 40.91% of the error type while it occurred 13 times in group B, which accounted for 59.09% of the error type.
- **E6: Code outside a Method or outside a Main method:** The error “E6: code outside a method or outside a main method” occurred 17 times. Group A (with tutorial) has a total occurrence number of 2 only which accounted for 17.76% of the error type while group B (without tutorial) has a frequency occurrences of 15 which accounted for 88.24% of the error type. Analysis shows that the students that undergone a tutorial makes less error of this type.
- **E4.1: Missing [] in Array:** The error type “E4.1: missing [] in array” occurred 16 times. Group A (with tutorial) has a total occurrence number of 6 times, which accounted for 37.50% of the error type while group B (without tutorial) with the frequency occurrences of 10 accounted for 62.50% of the error type. Analysis shows that the students that undergone a tutorial makes less error of this type.
- **E9: Incorrect Swapping of values:** The error “E9: Incorrect swapping of values” occurred 13 times. It occurred 5 times in group A (with tutorial) which accounted for 38.46% of the error type while group B (without tutorial) with the frequency occurrences of 8 accounted for 61.54% of the error type. Analysis shows that the students that undergone a tutorial makes less error of this type.
- **E5.2: Incorrect Printout Placement:** The error “E5.2: Incorrect printout placement” occurred 11 times. Group A (with tutorial) with a total occurrence number of 4 accounted for 36.36% of the error type while group B (without tutorial) with the frequency occurrences of 7 accounted for 63.64% of the error

type. Analysis shows that the students that undergone a tutorial makes less error of this type.

- **E3.2: Forgot to create object in calling a Non-static method:** The error “E3.2: Forgot to create object in calling a non-static method” occurred 10 times. 3 occurred in group A (with tutorial) which accounted for 30.00% of the error type while 7 was collected from group B (without tutorial) which accounted for 70.00% of the error type.

The 2 error type that has no improvement were; E1: Multiple if without else and E2.2: Incorrect accumulator which were all logical errors. It was observed that the 2 errors are logical errors. Analysis shows that logical errors are difficult to find and fix, so it requires a logical thinking and strategy in finding and fixing it.

- **E1: Multiple if without else:** Analyses shows that this error type has a total occurrences frequency of 37. It occurred 21 times in group A (with tutorial) which accounted for 56.76% of the error type while it occurred 16 times with group B with 43.24% occurrences frequency.
- **E2.2: Incorrect Accumulator:** Analyses shows that this error type has a total occurrences frequency of 7. It occurred 6 times in group A (with tutorial) which accounted for 85.71% of the error type while it occurred ones in group B with 14.29% occurrences frequency.

One of our limitation is that the number of participants is very low when compare to other related researches done in this area, we believe that the number of participants can affect the error type and it occurrence frequency, the larger the number of participants the more the number of errors expected. Only future research with a larger number of participants will prove that.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

Programming has been described as a useful skill and also regarded as a rewarding career due to the high increasing demand of Information Technology stuffs, indirectly the needs for properly trained IT Professionals by institutions is needed. IT institutions recognize programming as a difficult and complex intellectual activity with educators put more effort in teaching it and students struggling to learn it.

While a programmer is programming, one or more errors might erupt in the process which can be as a results of many reasons such as lack of good knowledge of the programming language concepts. This research study put emphasis more on logical error type which is believed to be more critical because when it occurred the compiler does not identify it and also most of the researches done in this area mainly concentrated on the syntax and runtime error type.

The main contributions of this research are the detection and categorization of errors made by Meliksah University students taken first year programming courses and the identification of those areas which an instructor need to concentrate and put more emphasis on while teaching programming courses.

It was seen that 9 errors were identified based on programming concepts and were further sub-grouped into 14 different sub types. The pre-experiments shows a total of 276 errors were collected, which comprises of logical error with the highest occurrences accounted for 63.04% of the total error, runtime error accounted for about 29.71% of the errors and 7.25% accounted for the syntax error with the least occurrences.

This research further revealed that a tutorial can be used as a technique to help novice programmer in reducing the occurrence error rate by educating them on the error type. This research shows that the Tutorial as a technique we used in the control experiments has helped the students in reducing the number of errors made by them and has improved their understanding in programming. In the control experiment a reduction in the number of errors was observed, where only 10 out of the 14 subgroup errors was recorded. 202 numbers of errors was recorded, the group with tutorial accounted for 40.1% of the total error and the group without tutorial accounted for 59.9% of the error.

Future studies will involve developing instructional strategies to assist novices in overcoming these difficulties and errors. This strategy includes providing a reference document that contains detailed list of errors with samples and how to prevent it. Based on the error modeling, an error rules can be generated which will be integrated into an Integrated Development Environment as a Plug-in to detect error and even suggest solutions to the error types we identified in this study. Additional participatory studies will be done to investigate the effectiveness of the techniques to reduce errors made by novice programmers.

APPENDIX A

A.1: Pre-Experiments Data Showing Location of Error Occurrence

Table A.1 Pre-Experiment Error Analysis

LOW-LEVEL ANALYSIS FOR PRE-EXPERIMENTS RESULT		
ERROR SUB-GROUP (low-level analysis)	Frequency counts	% frequency
E1: multiple “if” without “else”	38	13.77%
E2.1: incorrect loop initialization	14	5.07%
E2.1: incorrect loop termination condition	36	13.04%
E2.2: wrong accumulator	40	14.49%
E3.1: Type incompatibility i.e. passing wrong arguments	2	0.72%
E3.2: didn’t create object in accessing a non-static method	15	5.43%
E4.1: missing [] in an array declaration	12	4.35%
E4.1: missing {} in a single-line array declaration and assignment	5	1.81%
E2.2: incorrect array termination condition	31	11.23%
E5.1: missing printout statement	23	8.33%
E5.2: wrong placement of printout statement	9	3.26%
E6: writing of code outside a method/main method class	2	0.72%
E7: missing semicolon/parentheses	8	2.90%
E8.1: use of uninitialized/undeclared variables	19	6.88%
E8.2: keyword mistype	8	2.90%
E9: Incorrect swapping of values	14	5.07%
Total	276	100%

A.2: Students' GPA and the Grouping

Two groups were formed based on the CSE101 first semester students GPA, where group A underwent a tutorial and group B does not. An analysis was made among the two groups after the tutorial.

Table A.2 two groups formed based on GPA

Grouping based on the Participants GPA							
S/N	Group A	GPA ranges		S/N	Group B		GPA ranges
		Letter	Number			Letter	Number
1	A23	DC	1.50	1	A38	CC	1.00
2	A5	CC	1.00	2	A39	FF	0.00
3	A24	FF	0.00	3	A32	DC	1.50
4	A17	FD	0.50	4	A4	FF	0.00
5	A16	FD	0.50	5	A18	FD	0.50
6	A21	FF	0.00	6	A37	FF	0.00
7	A20	DD	2.00	7	A12	FF	0.00
8	A35	CB	2.50	8	A40	CB	2.50
9	A22	CB	2.50	9	A29	CB	2.50
10	A19	FF	0.00	10	A31	DD	2.00
11	A9	FF	0.00	11	A27	FF	0.00
12	A2	FF	0.00	12	A1	BA	3.50
13	A6	BB	3.00	13	A3	DC	1.50
14	A34	BA	3.50	14	A28	CC	2.00
15	A30	FF	0.00	15	A26	DD	1.00
16	A8	BA	3.50	16	A33	BB	3.00
17	A36	CC	2.00	17	A13	BA	3.50
18	A11	AA	4.00	18	A7	AA	4.00
19	A25	AA	4.00	19	A15	AA	4.00
20	A14	CC	2.00	20	A10	FD	0.50
Total			32.50	Total			33.00

A.3: Control Experimental Results Showing Location of Error Occurrences

Table A.3 Control Experiment Error Result Analysis

LOW-LEVEL ERROR ANALYSIS FOR EXPERIMENTAL RESULT						
Error and locations	Group A (with tutorial)	% frequency for Group A	Group B (without tutorial)	% frequenc y for Group B	Total frequency	Total % frequenc y
E1: Multiple if without else	21	10.40%	16	7.92%	37	18.32%
E2.1: incorrect loop initialization statement	4	1.98%	8	3.96%	12	5.94%
E2.1: incorrect Loop termination condition	12	5.94%	18	8.91%	30	14.85%
E2.2: incorrect accumulator	6	2.97%	1	0.50%	7	3.47%
E3.2: forgot to create object in calling a non-static method	3	1.49%	7	3.47%	10	4.95%
E4.1: missing []	6	2.97%	10	4.95%	16	7.92%
E4.2: incorrect Array termination condition in a loop	9	4.46%	18	8.91%	27	13.37%
E5.1: missing print out statement	9	4.46%	13	6.44%	22	10.89%
E5.2: Incorrect printout placement position	4	1.98%	7	3.47%	11	5.45%
E6: Incorrect written of Code outside a method/main method class	2	0.99%	15	7.43%	17	8.42%
E9: Incorrect swapping of values	5	2.48%	8	3.96%	13	6.44%
TOTAL	81	40.10%	121	59.90%	202	100.00%

APPENDIX B

Some Questions Treated In the Tutorial

- Sample Declarations and Creations:
 - `int[] numbers = new int[10];`
 - `char[] letters= new char[29];`
 - `String[] names = new String[63];`
- Assigning values
 - `numbers[0] = 10; // numbers[0] is 10 now`
 - `numbers length is 10.`

- Sample Initializations in Java:
 - `int[] age = {17,18,55,42,34,27,66,13,4,0};`
 - `double[] bigNumber = {3.4, 2.71};`
 - `char[] letter = {'a', '#', 'b','x','l'};`
 - `String[] names =`
`{"Ahmet","Mehmet","Murat","Selim"};`

- ```
public class Sum {
 public static void main(String[] args) {
 double myList = [1,3,5,7,9];
 double total = 0;
 for (int i = 0; i <= 6; i++) {
 total += total;
 }
 System.out.print(total);
 }
}
```

```

➤ public class Methods {
 public static void Add(int x, int y) {
 int total = x + y;
 System.out.println("total = "+total);
 }
 public static void main(String[] args) {

 }
}

```

- This method prints a message n times, How can we invoke/call the method so that it print a message 100 times.

```

public class nPrintln {
 public void nPrintln(String message, int n) {
 for (int i = 0; i < n; i++)
 System.out.println(message);
 }
 public static void main(String[] args) {

 }
}

```

- Some common issues while dealing with methods

- Incorrect Semicolon at the End of a Method Header: There should never be a semicolon at the end of a method header

```

- Void methodName (arguments); {<statements>}
 should be
 Void methodName (arguments) {<statements>}

```

- Method call parameter types must match method definition parameter types

- Given definition `void methodName(int x, char y)` in method call `methodName(a, b)` *a* must be an *int* and *b* must be a *char*

Types don't have to match exactly - a *double* can take an *int*. *which care has to be taking in dealing with this kind of issues.*

- Swapping of values

You first define a temporary variable which should store the value of the first variable whose value will be destroyed at the first assignment.

```
int x,y, temp;
x=10; y=20;
temp = x;
x=y;
y=temp;
```

At the last step, we use the temporary variable to get the original value of the first variable and assign it back to the second variable.



## APPENDIX C

### Written Experiments Paper Based Assignment Analysis

- **E1: if-else (Read and provide the correct output)**

What output is produced by the following code fragment given the assumptions below?

Assuming the value of num1 is 2 and the value of num2 is 10?

```
if (num1 < num2) YES
 System.out.print (" red "); red
if ((num1 + 5) < num2) YES
 System.out.print (" white "); white
else
 System.out.print (" blue ");
System.out.println (" yellow "); yellow
```

The question above was given in order to test the students in reading and printing out of correct result in nested if-else. A participant is to read the question using the given input to predict the correct output, 28 students participated; only 4 students got it right the remaining 24 students were printing a wrong result. Figure C.1 below shows one of the participants result.

```

if (num1 < num2)
System.out.print (" red ");
if ((num1 + 5) < num2)
System.out.print (" white ");
else
System.out.print (" blue ");
System.out.println (" yellow ");

```

a. Assuming the value of num1 is 2 and the value of num2 is 10? red, white

**Figure C.1 wrong result output**

- **Loop (Conversion from for-loop to while-loop)**

Convert the following for loop statement to a while loop:

```

long sum = 0;
for (int i = 0; i <= 1000; i++)
sum = sum + i;
 System.out.println(sum);
}
}

```

The question in above requires a student to convert the given code from for-loop to while-loop, mistakes the students were making was using the iterative variable *i* without declaration and initialization E8.1, wrong positioning of the printout statement E5.2, usage of while-loop structure like for-loop structure. Figure C.2 shows how *i* was used before it was declared and initialize. Figure C.3 shows placement of an incorrect printout statement. Figure C.4 shows usage of a while-loop like for loop.

```

long sum = 0;
while (i <= 1000) {
int i = 0;
sum = sum + i;
System.out.println(sum);
}

```

**Figure C.2 Showing usage of *i* without declaration**

```

int i = 0;
long sum = 0;
while (i <= 1000) {
sum = sum + i;
sum = sum + i;
sum = i;
sum = sum + i;
 cout << sum;
 i++;
}

```

**Figure C.3 Shows placement of an incorrect placement of printout**

```

long sum = 0;
while (int i = 0; i <= 1000; i++)
 sum = sum + i;
 cout << sum;
}

```

**Figure C.4 shows usage of a while-loop like for-loop**

- **Traced and predict the output**

What output is produced by the following code fragment?

```

int value = 0;
for (int num = 10; num <= 40; num += 10)
{
 value = value + num;
}
System.out.println (value);

```

The question above was to test the students understanding of looping structure and predicting the correct output. Figure C.5 below shows an incorrect result was predicted.

```

int value = 0;
for (int num = 10; num <= 40; num += 10)
 value = value + num;
System.out.println (value);

```

Handwritten annotations on the right side of the code: 10, 30, 50, 80, and 120. An arrow points from the final output '120' to the right.

**Figure C.5 wrong result printout in loop**

- **Method Concepts**

This method prints a message n times, write a code that invoke/call the method and print a message 100 times.

```

public class nPrintln {
 public void nPrintln(String message, int n) {
 for (int i = 0; i < n; i++)
 System.out.println(message);
 }
 public static void main(String[] args) {

 }
}

```

The above question test student's usage of a non-static method i.e. objects creation E3.2 and parameter matching E3.1. Figure C.6 shows how a method was called directly without using an object and Figure C.7 shows how a method was call with a wrong variable.

```

public class nPrintln {
 public void nPrintln(String message, int n) {
 for (int i = 0; i < n; i++)
 System.out.println(message);
 }
 public static void main(String[] args) {
 nPrintln("hello", 100);
 }
}

```

**Figure C.6 Missing object in calling of Non-Static method**

```

public class nPrintln {
 public void nPrintln(String message, int n) {
 for (int i = 0; i < n; i++)
 System.out.println(message);
 }

 public static void main(String[] args) {
 nPrintln p = new nPrintln();
 p.nPrintln(100, "Hollo");
 }
}

```

**Figure C.7 incorrect parameter mismatch**

- **Array Concepts**

This program prints the sum of a numbers in the array index, there is an error in the program, identify and fix the errors.

```

public class Sum {
 public static void main(String[] args) {
 double myList = [1,3,5,7,9];
 double total = 0;
 for (int i = 0; i <= 6; i++) {
 total += total;
 }

 System.out.print(total);
 }
}

```

In the question above a code fragments was given to the students containing an error, i.e. missing [] in declaring an array, also wrong accumulation was used e.g. total+=total and trying to go beyond the array boundary. Which the students are required to trace, find and fix the errors.

## References

- [1] A. J. R. a. N. R. Robins, "Learning and teaching programming: A review and discussion," *Computer Science Education*, pp. 137-172, 2003.
- [2] M. Pendergast, "Teaching introductory programming to IS students: Java problems and pitfalls," *Journal of Information Technology Education: Research*, pp. 491-515, 2006.
- [3] R. H. a. P. T. Sloan, "A better approach to introductory computer science for majors," *ACM SIGCSE Bulletin*, pp. 271-275, 2008.
- [4] L. Winslow, "Programming pedagogy– A psychological overview," *SIGCSE Bulletin*, p. 17–22, 1996.
- [5] M. A. V. D. D. G. M. H. D. K. Y. L. C. T. L. I. a. W. T. McCracken, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," *In Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '01)*. ACM, New York, NY, USA, pp. 125-180, 2001.
- [6] K. A.-M. a. H. J. E. Lahtinen, "A study of the difficulties of novice programmers," *ACM SIGCSE Bulletin*. - New York, NY, USA : ACM Press, vol. 37, pp. 14-18, 2005.
- [7] H. M. H. J. M. P. T. C. W. C. Daryl D., "Transforming learning of programming: a mentoring project," *Proceedings of the tenth conference on Australasian computing education*, pp. 75-84, 2008.
- [8] S. D. W. P. R. Cooper, "ALICE: A 3-D tool for Introductory Programming Concepts," *Proceedings of the fifth annual CCSC north eastern conference on The journal of computing in small colleges Ramapo College of New Jersey, Mahwah, New Jersey*, pp. 107-116, 2000.
- [9] S. L. D. K. V. Wiedenbeck, "Factors Affecting Course Outcomes in Introductory Programming," *Psychology Interest Group*. Carlow, Ireland, 2004.
- [10] K. Ala-Mutka, "Problems in Learning and Teaching Programming- A literature Study for Developing Visualizations in the Codewitz-Minerva Project", *Codewitz Needs*

*Analysis*, 2004.

- [11] I. & T. G. Miliszewska, "Befriending Computer Programming: A Proposed Approach to Teaching Introductory Programming," *Issues in Informing Science and Information Technology*, 2007.
- [12] B.-A. M, "Constructivism in computer science," *ACM Press New York, NY, USA*, 1998.
- [13] S. S. a. S. R. Perkins D.N., "Instructional strategies for the problems of novice programmers," *Teaching and Learning Computer Programming: Multiple Research Perspectives*, pp. 153-178, 1988.
- [14] S. D, "An Introductory Pascal Class: A Case Study of Students' Errors," *Teaching and Learning Computer Programming: Multiple Research Perspectives*, pp. 237-257, 1988.
- [15] L. D. W. S. Ramalingam V., "Self-efficacy and mental models in learning to program," *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pp. 171-175, 2004.
- [16] H. P. R. A. Garner S., "My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems," *Proceedings of the 7th Australian conference on Computing education*, pp. 173-180, 2005.
- [17] K. J, "Problem-Based Learning for Foundation Computer Science Courses," *Computer Science Education*, pp. 109-128, 2000.
- [18] M. I. a. R. G, "Difficulties in Learning and Teaching Programming—Views of Students and Tutors," *Education and Information Technologies* , pp. 55-66, 2002.
- [19] M. J. a. F. T, "Why We Need to Develop Success in introductory programming courses," *Central Plains Conference*, 1999.
- [20] B. P. a. S. J, "Roles of variables and programming skills improvement," *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pp. 413-417, 2006.
- [21] S. J. a. K. M, "An Experiment on Using Roles of Variables in Teaching Introductory Programming," *Computer Science Education* , pp. 59--82, 2005.
- [22] W. L.E, "Programming Pedagogy--A Psychological Overview," *SIGCSE Bulletin*, pp. 17--22, , 1996.
- [23] M. E. D. a. H. C. Ahmadzadeh, "An analysis of patterns of debugging among novice computer science students," *In Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05)*, pp. 84-88,

2005.

- [24] K. S. a. K. J, "The neglected battle fields of syntax errors," *Proceedings of the fifth Australasian conference on Computing education.- Darlinghurst Australia : Australian Computer Society*, pp. 105-111, 2003.
- [25] I. a. G. R. Milne, "Difficulties in Learning and Teaching Programming—Views of Students and Tutors," *Education and Information Technologies*, pp. 55-66, 2002.
- [26] H. C. a. K. V. Lane, "Coached Program Planning Dialogue-Based Support for Novice Program Design," *ACM SIGCSE Bulletin*, pp. 148--152, 2003.
- [27] R. W. Sebesta, *Concepts of programming languages*, 10 ed., Pearson Education, Inc., publishing as Addison-Wesley, 2012.
- [28] "<http://mashable.com/2014/01/21/learn-programming-languages>," [Online].
- [29] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice Hall Software Series.
- [30] J. L. P. a. S. T. GOLDEN, *JAVA PROGRAMMING: PART 1*, 1998.
- [31] "Techopedia," [Online]. Available: <http://www.techopedia.com/definition/26860/integrated-development-environment-ide>. [Accessed 3 July 2014].
- [32] P. Niemeyer and D. Leuck, *Learning Java*, O'Reilly Media, 2013.
- [33] M. K. olling, 2014. [Online]. Available: <http://www.bluej.org/>.
- [34] E. S. Tabanao, M. M. T. Rodrigo and M. C. Jadud, "Identifying At-Risk Novice Java Programmers Through the Analysis of Online Protocols," 2008.
- [35] R. C. a. B. S. E. Allen, " Drjava: a lightweight pedagogic environment for java," *SIGCSE : Proceedings of the 33rd SIGCSE technical symposium*, 2002.
- [36] T. Flowers, C. A. Carver and J. Jackson, "Empowering Students and Building Confidence in Novice Programmers through Gauntlet," *Savannah, GA 34Th ASEE/IEEE Frontiers in Education Conference T3H-10*, p. 20 – 23, 2004.
- [37] A. M. M. R. a. R. M. M. Hristova, "Identifying and correcting java programming errors for introductory computer science students," *Proceedings of the 34th SIGCSE technical symposium on Computer science educatio*, p. 153–156, 2003.
- [38] D. W. Toomey, "Quantifying The Incidence of Novice Programmers' Errors".
- [39] J. & S. E. Bonar, "Uncovering principles of novice programming," *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*,



Austin, Texas, pp. 10-13, 1983.

- [40] S. Davies, "Models and theories of programming strategy," *International Journal of Man-Machine Studies*, p. 237–267 , 1993.
- [41] J. & S. R. Rogalski, "Acquisition of programming knowledge and skills," *Psychology of programming*, p. 157–174, 1990.
- [42] F. Detienne, "Expert programming knowledge: A schema based approach," *Psychology of programming* , p. 205–222, 1990.
- [43] M. & D. J. Linn, "Cognitive consequences of programming instruction," *Studying the novice programmer*, p. 57–81, 1989.
- [44] D. H. C. H. R. M. F. Perkins, "Conditions of Learning in Novice Programmers," *Journal of Educational Computing Research* , vol. 2, pp. 37-55, 1986.
- [45] B. D. Boulay, "Some difficulties of learning to program," *Journal of Educational Computing Research*, pp. 57-73, 1986.
- [46] D. M. R. D. P. C. C. a. R. M. Kurland, "A study of the development of programming ability and thinking skills in high school students," *Journal of Educational Computing Research* , pp. 429-458, 1986.
- [47] D. N. a. F. M. Perkins, "Fragile knowledge and neglected strategies in novice programmers," *In first workshop on empirical studies of programmers on Empirical studies of programmers*, pp. 213-229, 1986.
- [48] J.-M. E. S. a. J. S. H. N. L. E. A. Hoc, "Do We Really Have Conditional Statements in Our Brains? Studying the Novice Programmer," 1989.
- [49] J. C. a. E. S. Spohrer, " Novice mistakes: Are the folk wisdoms correct?," *Communications of the ACM* , pp. 624-632, 1986.
- [50] E. B. J. & E. K. Soloway, "Cognitive strategies and looping constructs: An empirical study," *Communications of the ACM*, pp. 853-860, 1983.
- [51] M. Ben-Ari, Constructivism in computer science education, vol. Vol. 30(1), ACM Press New York, NY, USA. , 1998.
- [52] M. A. M. M. R. a. R. M. Hristova, "Identifying and Correcting Java Programming Errors for Introductory Computer Science Students," *ACM SIGCSE*, pp. 19-23, 2003.
- [53] A. Z. S. a. I. A. M. Gobil, "Novice difficulties in selection structure," *International Conference on Electrical Engineering and Informatics*, p. 351–356, 2009.
- [54] S. L. G. M. R. M. L. S. B. T. L. & Z. C. Fitzgerald, "Debugging: Finding, fixing and

- flailing, a multi-institutional study of novice debuggers," *Computer Science Education*, p. 93–116, 2008.
- [55] A. Ebrahimi, "Novice programmer errors: Language constructs and plan composition," *International Journal of Human-Computer Studies*, p. 457–480, 1994.
- [56] C. & P. R. Kelleher, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys*, p. 83–137, 2005.
- [57] C. A. C. a. J. J. T. Flowers, "Empowering Novice Programmers with Gauntlet," *Frontiers in Education*, 2004.
- [58] A. a. E. E. Bruckman, "ACM SIGCHI," *Should we leverage natural language knowledge? An Analysis of user errors in a natural-language style programming language*, pp. 207-214, 1999.
- [59] J. a. T. F. H. Chabert, "An Investigation of Novice Programmer Errors in IBM 370 (OS) Assembly Language," *ACM 14th Annual Southeast regional Conference*, pp. 319-323, 1976.
- [60] M. M. T. R. & M. C. J. Emily S. Tabanao, "Identifying At-Risk Novice Java Programmers Through the Analysis of Online Protocols," 2008.
- [61] M. C. a. C. C. J. Jackson, "Identifying top java errors for novice programmers," vol. 1, p. 24–27, 2005.
- [62] N. a. V. R. J. Pillay, "An Analysis of the Errors Made by Novice Programmers in a First Course in Procedural Programming in Java," *Preface of the Editors*, 2006.
- [63] P. K. Y. L. J. Y. Sevela, "Determining the Barriers Faced by Novice Programmers," 2013.
- [64] I. T. C. Mow, "Analyses of Student Programming Errors In Java Programming Courses," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, pp. 739-749, 2012.
- [65] J. P. a. B. Myers, "Usability issues in the design of novice programming systems," *School of Computer Science Technical Reports*, pp. 96-132, 1996.

