Increasing the Effectiveness of Error Messages in a Computer Programming and Simulation Tool

by

Siddhant Bapusaheb Tanpure

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2018 by the
Graduate Supervisory Committee:

Sohum Sohoni, Chair
Scotty D. Craig
Kevin A. Gary

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

Each programming language has a compiler associated with it which helps to identify logical or syntactical errors in the program. These compiler error messages play important part in the form of formative feedback for the programmer. Thus, the error messages should be constructed carefully, considering the affective and cognitive needs of programmers. This is especially true for systems that are used in educational settings, as the messages are typically seen by students who are novice programmers. If the error messages are hard to understand then they might discourage students from understanding or learning the programming language. The primary goal of this research is to identify methods to make the error messages more effective so that students can understand them better and simultaneously learn from their mistakes. This study is focused on understanding how the error message affects the understanding of the error and the approach students take to solve the error. In this study, three types of error messages were provided to the students. The first type is *Default type* error message which is an assembler centric error message. The second type is *Link type* error message which is a descriptive error message along with a link to the appropriate section of the PLP manual. The third type is *Example type* error message which is again a descriptive error message with an example of the similar type of error along with correction step. All these error types were developed for the PLP assembly language. A think-aloud experiment was designed and conducted on the students. The experiment was later transcribed and coded to understand different approach students take to solve different type of error message. After analyzing the result of the think-aloud experiment it was found that student read the *Link type* error message completely and they understood and learned from the error message to solve the error. The results also indicated that *Link type* was more helpful compare to other types of error message. The *Link type* made error solving process more effective compared to other error types.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Figure Page

Figure Page

LIST OF TABLES

Tables Page

Tables Page

LIST OF ABBREVATIONS

1. PLP – Progressive Learning Platform
2. IDE – Integrated Development Environment
3. IRB – Institutional Review Board
4. FPGA – Field Programmable Gate Array
5. GUI – Graphical User Interface

CHAPTER 1

INTRODUCTION

1.1 Motivation

Writing programs without any syntax errors is a primary goal of every programmer. To achieve this goal, many techniques and tools have been invented. Some of these are visual effects, feedback mechanism, standard guidelines and programming methods to understand and avoid errors. Despite this, programmers, especially novice programmers, make mistakes and spend a considerable amount of time on correcting them.

There are three main reasons because of which novice programmers make mistakes (Traver, V. J. (2010)). The first one is lack of knowledge, a programmer who is new to programming has very little understanding of the syntax. The second one is incorrect understanding, sometimes programmer understands or learns concepts incorrectly. The third one is the blunders, which programmers make because of lack of attention.

Each programming language has a compiler associated with it. The compiler converts the program from one programming language (high-level language) to another programming language (low-level language). The compiler helps to identify errors in the program and provides potential solutions to the errors using error messages. The main purpose of these error messages is to help programmers to identify and correct these errors.

Error messages are one important way of understanding the problems in the program. If the error messages are difficult to understand then it takes more time to correct the errors (Prather, J., Pettit, R., McMurry, K. H., Peters, A., Homer, J., Simone, N., & Cohen, M. (2017, August)). The error messages are written by the compiler designers.

Sometimes error messages appear simple for compiler designers but could be very cryptic and uninformative to the students/novice programmers. Sometimes the error message developer makes assumptions based on their knowledge of programming language, but they forget that it might be ambiguous to students who are trying to learn the programming language or are new to programming.

Students spend most of the time on correcting the errors in the program (McCall, D., & Kölling, M. (2014, October); Pettit, R. S., Homer, J., & Gee, R. (2017, March); Chabert, J. M., & Higginbotham, T. F. (1976, April)). Repeating same errors are common mistakes students make while learning the programming language(McCall, D., & Kölling, M. (2014, October); Pettit, R. S., Homer, J., & Gee, R. (2017, March); Chabert, J. M., & Higginbotham, T. F. (1976, April); Becker, B. A. (2016, February)). It is a good indicator for understanding where students are struggling the most. An effective error message will provide critical feedback to the novice programmers.

This study is focused on understanding the effects of different types of error messages on students while writing assembly language programs so that developers can design effective error messages. A modified version of Progressive Learning Platform (PLP) tool is used to provide three different types of error messages. Those types are Default type, *Link type* and Example type to understand which error message type is effective. The effective error messages will be easy to understand which will help novice programmers fix them and avoid making the same mistakes repeatedly.

1.2 Related work

Most programmers will agree that error messages are sometimes difficult to understand or uninformative. There are many studies available to understand the reason behind the errors made by novice programmers (Hartmann, B., MacDougall, D., Brandt,

J., & Klemmer, S. R. (2010, April); Nienaltowski, M. H., Pedroni, M., & Meyer, B. (2008); Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003, February); Prather, J., Pettit, R., McMurry, K. H., Peters, A., Homer, J., Simone, N., & Cohen, M. (2017, August); McCall, D., & Kölling, M. (2014, October)). These studies help us to understand problem-solving methods, errors made and difficulty with understanding programming concepts by novice programmers. To design effective error messages, it is very important to look at the mistakes made by the novice programmer. Also, studying the prior work can also help in understanding methods used by other authors to design effective error messages.

In the study conducted by Becker, B. A. (2016, February), an editor was designed and implemented providing enhanced compiler error messages. The study was a controlled empirical study on CS1 students who were trying to learn Java. The study found that there were 10 most frequent errors which represented 79% of total errors. The study designed a tool which analyzed the students' source code and error messages which are then customized based on the offending code to create enhanced error messages. The study also found that the number of errors repeated by the students reduced after enhanced error message system. The study helped to understand that detail description of the error message improves the performance of the students. Based on the results of this study we designed detailed description of error messages for assembly language programming.

In another study conducted by Radosevic, D., & Orehovacki, T. (2011, June), a Verificator was created as a part of Tutor which is a C++ learning programming interface. The purpose of Verificator was to explain students the causes of syntactic and certain logical incorrectness in the program. The study was conducted on 154 students. The most frequent errors occurred during the study were variable declaration and usage. The

research indicated that if we provide effective error messages then the performance of student can be improved.

In study by Chabert, J. M., & Higginbotham, T. F. (1976, April), tabulated errors based on type and frequency of error messages. In this study, 25 students ran 1531 programs and out of 3150 errors 539 contained syntax error of one or another type. The programming language used in this study was IBM 370 (OS) Assembly Language. The study helped to understand most common types of errors found in assembly language are an undefined symbol, addressability error, Invalid delimiter, Invalid syntax, incorrect specification of a register or mask field, invalid and undefined op code, invalid displacement and end of data on Sysin.

In a study by Ebrahimi, A. (1994), indicated the problem novice programmers face while programming. The study divided 80 students into four equal groups and each group participated in two experiments. The two experiments were focused on language constructs and plan composition. The language constructs experiment contained small segments of the program to study input, output, loops, decision making and specific features of the languages. In plan composition, students were asked to create a plan for a common problem known as “rainfall”. The results of the study indicate that more emphasis is necessary for error checking.

In the study by Munson, J. P., & Schilling, E. A. (2016), the authors created a programming tool called as “Codework” and collected programming activities of 46 students in two sections. Total 5879 observations were made during this study from graded assignments and two in-class timed quizzes. This study shows that students do read error messages. This study also states, “We can understand a decrease in the percentage of time the first error was addressed, as students discover that it is not easy to

know what action to take in response to a compiler error message and therefore cast around blindly." This proves that if student receives an error message which is confusing then they tend to avoid reading compiler error messages. Similar observations were made when students were provided "*Example type*" of error messages.

In the study by Barik, T., Witschey, J., Johnson, B., & Murphy-Hill, E. (2014), to communicate error messages and their resolutions, authors have designed taxonomies which can help to create vocabularies in more controlled and expressive ways. These taxonomies also help to give consistent and unified semantics to error objects. The study did not provide any experimental data which can prove if this approach works or not.

In the study by Traver, V. J. (2010), the author focused on the problem of cryptic error messages from the perspective of human-computer interaction. Human-computer interaction is the study of user interfaces to make them more efficient and effective. The paper shows that error messages which are designed poorly affect novice programmers more adversely than expert programmers. The paper also provides actual compiler error messages as examples and some principles for compiler error message design. The paper states that there are many techniques available to design complex software such as design principles, visual programming techniques, and integrated environments. which are proven useful. Yet not much has been done with compiler error messages. According to the paper, error messages are difficult to interpret, which makes the error difficult to resolve and prevent in the future. The paper provides three main reasons for bad error message design by compiler designers. The first reason is concern about memory requirements by compiler developers which limits their ability to record feedback about the error messages. The second reason is that compiler developer uses their own knowledge about the language to design the succinct description of the error. The third reason is priorities: developers give more priority to other features than error messages.

The paper also discussed two approaches to resolve the cryptic error message problem: compiler driven approaches and programmer driven approaches. The paper states that the problem has been addressed previously and some interesting issues were identified. To address these issues more study and experiments are necessary. The paper also shows how error messages which are difficult to understand affect novice programmers and decrease their productivity and learning progress. This paper helped to design programmer centric error messages and helped to identify compiler centric error messages. The study helps to confirm that error messages are cryptic for novice programmers and needs to be studied and improved.

1.3 Prior work conducted at ASU

The study conducted by Kadekar, H. (2016), the author tried to understand the effects of error messages on the students' ability to resolve the error. The study analyzed the students' reaction to different types of error messages. The study answered two questions. The first question was aspects of the error message helped the programmer to understand the error. The second is understanding the aspects of the error message which helped to fix the error. In this study, each student was provided three programs with three different types of error messages and feedback questionnaires. The participants were chosen from two different classes. One class was undergraduate level class and another class was graduate level class. Each student was given 10 minutes to solve the program. Think aloud protocol was used to record students' thought process along with the screen. Each action was recorded using Chi, M. T., 1997, paper into different steps. Total 13 participants data were transcribed, coded and analyzed. The Cohen's Kappa for basic steps coding was 0.8736 which showed near a perfect agreement and for expected and unwanted steps were 0.62 which showed moderate agreement. The result indicated the reduced number of incorrect steps and better understanding in case of an error message with a hyperlink to a

relevant section in an online manual. The students took less time and steps to fix the error in case of Link type error message. The study also stated that the students were more confused in the Example type error message and in case of assembler-centric error messages i.e. Default type error messages.

1.4 Problem Statement

The main goal of this research is to find a way so that compiler designers for tools used in educational settings can write effective error messages. Effective error messages will be easy to understand. They will also help programmers understand their mistakes and prevent them from making same mistakes again and again. This will help to increase the confidence level in novice programmers and they will be more focused on writing good programs. This study will also help to understand the impact of different types of error messages on students learning ability.

The three types of error messages that are designed and used in this study are:

1. Default – *Default type* errors are indicated use a short error description with the line number indicating where the error occurred.
2. Link – *Link type* error messages are indicated use a detailed error description along with a link to the online manual and the line number indicating where the error occurred.
3. Example – *Example type* error messages use the same detailed error description as in the *Link type*, but also provide an example of the similar type of error and correction, along with the line number indicating where the error occurred.

1.5 Research Question

The relevant study indicated that error message affects the students' ability to understand and correct the errors. It is important to understand the effects of error messages on students process of resolving the errors in details. The study tries to answer following questions, given that this study is for an educational tool, and we would like the error message to either directly help a student learn the cause of an error, or direct them to a resource that will help them learn the cause of the error.

1. From the time students get an error message, to the time they fix the error, what is their strategy for resolving that error, and what impact does the error message type have on that strategy?

2. In what ways do the different error message types facilitate student learning, as evidenced by students either understanding the cause of the error directly from the error message, or students referring to resources to understand the cause of the error?

The answer to above questions gathered from the experiment will also help us to identify which error message type is more effective. Also, it will help us to understand do the students read compiler error messages completely or partially and whether they understand it or not.

CHAPTER 2

IMPLEMENTATION

This study is part of research and development on a simulation and visualization educational tool called the Progressive Learning Platform (PLP). PLP tool is a tool designed for use in multiple computer engineering courses, and has a number of facets to it, including a Verilog description of a CPU, a custom assembly language designed for teaching computer organization and embedded systems, and a simple Integrated Development Environment (IDE) PLPTool. “Progressive learning platform is an FPGA based computer architecture learning platform and was designed for students to anchor their conceptual learning about microprocessors and computer architecture, and for them to see the connections between assembly language and trade-offs in architecture” (Sohoni, S., 2014, June).

To conduct this study, modifications were made to the PLPTool. A separate module was designed to display customized error messages. While parsing the program, whenever the compiler detected errors, this custom module was called. The module contained three parts. The first one is the description, which provided a programmer-centric description of the error. The second one is Links, where links to the online PLP manual were stored. The online PLP manual is divided into sections and only the section of manual which is relevant to the error was given in the link. For example, if the error is due to an incorrect instruction, then the link would take the user to the section of the manual that listed all the valid instructions. If the error is related to register usage, then the link to the section on registers was given. The third part was an example, in which example for each type of error was stored. The example contained two parts, a “before correction” example and an “after correction” example based on the research conducted by Hartmann, B., MacDougall, D., Brandt, J., & Klemmer, S. R., 2010, April.

Based on the study of PLP assembly language and feedback from professors and teaching assistants, errors were grouped into 4 categories. (Kadekar, H. (2016))

1. Invalid label – This error has two sub groups.
    a. Duplicate label – This error occurs when the programmer tries to define same label at two different location in the program.
    b. Invalid target – This error occurs when the program tries to use a label which is not defined.
2. Invalid token – This has two sub groups.
    a. Invalid instruction type – This error typically occurs when a programmer makes a spelling mistake.
    b. Invalid label – This error occurs when the programmer forgets to put a colon after the label.
3. Invalid number of tokens – This error occurs when the programmer tries to use an invalid number of arguments or operands. It has two sub groups.
    a. Missing tokens – This error occurs when the programmer provides fewer arguments/operands than expected.
    b. Extra tokens – This error occurs when the programmer provides more number of arguments/operands than expected.
4. Invalid operand – This error occurs when the programmer gives an operand which does not adhere to the PLP language rules. It has four sub groups.
    a. Not Register – Occurs when the programmer provides value or invalid register name where a valid register name is expected.
    b. Not Number – Occurs when the programmer provides unexpected input to the operand instead of a valid number.

c. Not String – Occurs when the programmer provides unexpected input to the operand instead of a string.
d. Invalid address – Occurs when the programmer provides unexpected input to the operand instead of the valid address.

Figure *1* shows the Graphical User Interface(GUI) of PLP Tool used for the experiment. This is the tool which is used in class to teach assembly language programming at Arizona State University (ASU).

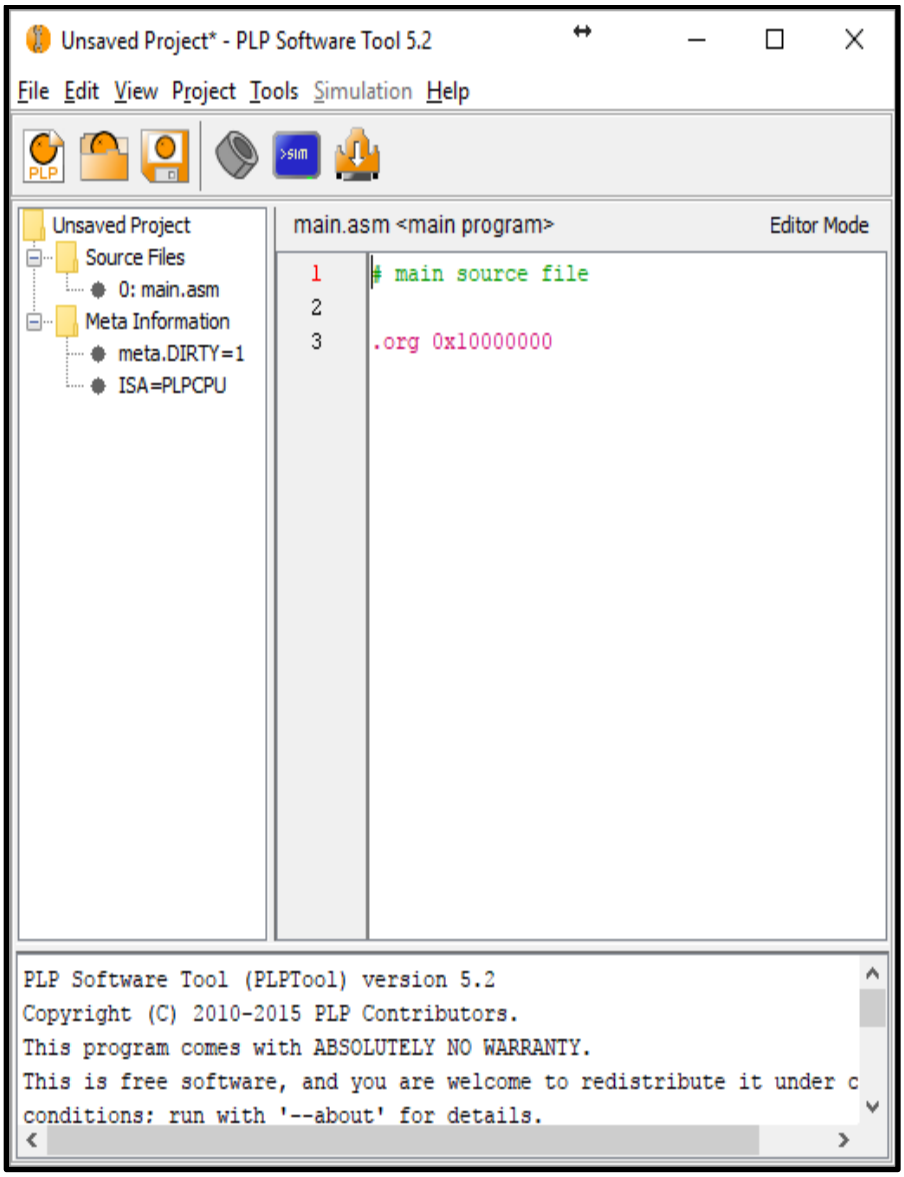

Figure 1 - PLP Tool Interface

Figure *2* gives an example of *Default type* error message. Figure *3* shows an example of *Link type* error message. Figure *4* shows an example of *Example type* error message. *Default type*, *Link type* and *Example type* error message were programmed based on number one, two and three respectively. By changing the number in the modified module different type error message was provided to the students.

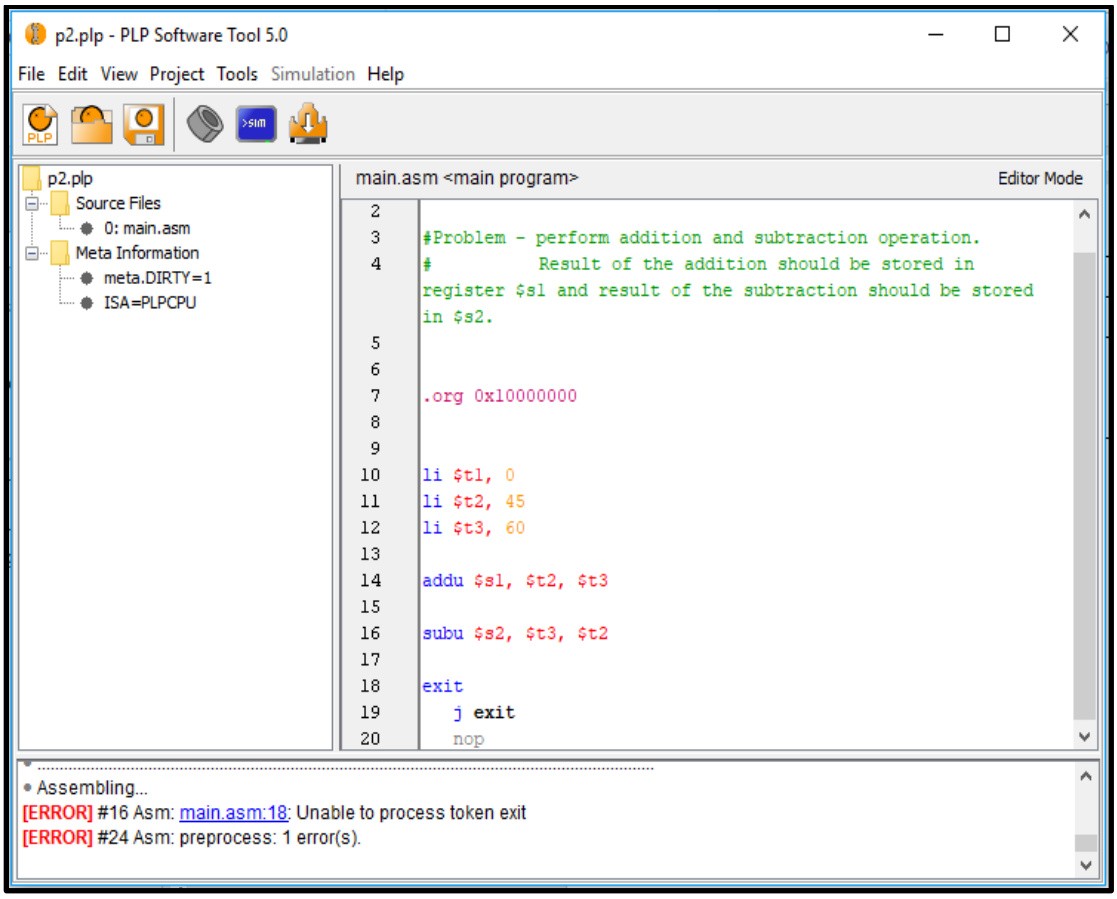

Figure 2- Example of *Default Type* Error

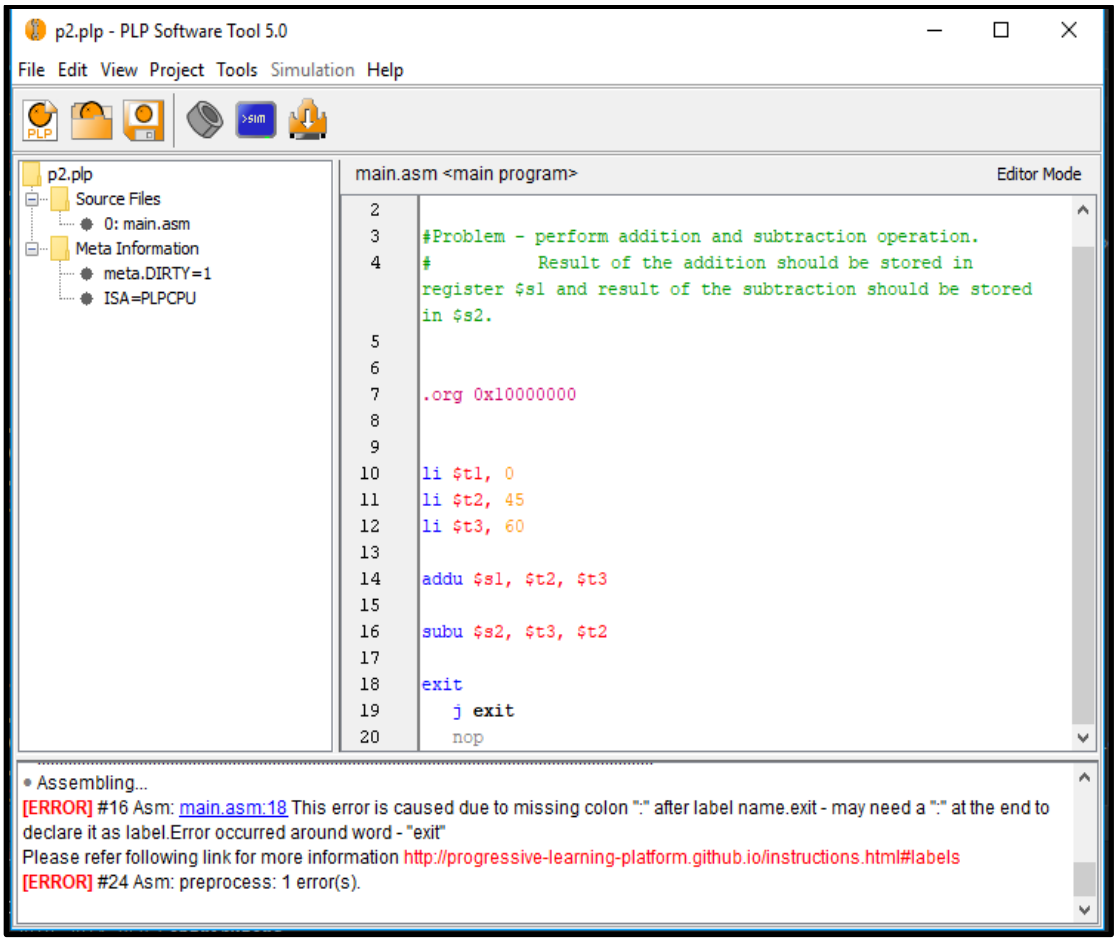

Figure 3- Example of Link Type error

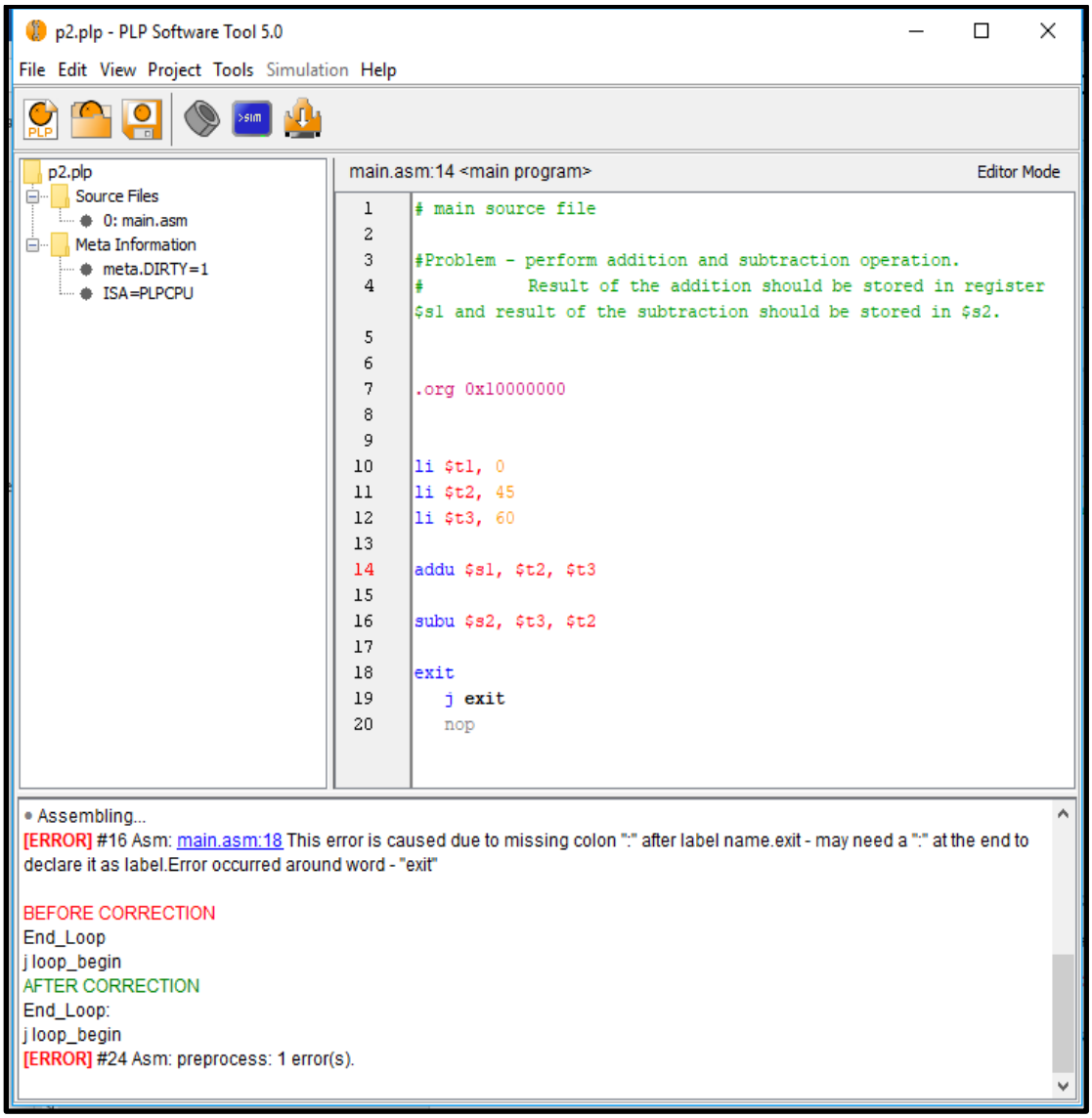

Figure 4- Example type error

CHAPTER 3

METHODS

3.1 Design

The experiment consists of three main stages. The first one is the consent forms, in which each student was provided one physical copy and one digital copy of the consent form to sign. The second stage was demographic survey, in which students were asked about their proficiency in programming. The third stage was 6 PLP programs, each one followed by a feedback questionnaire form.

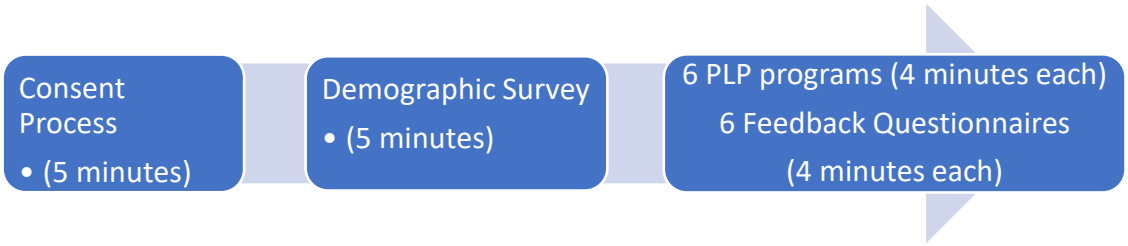

Figure 5- Three main stages of the experiment

The main components of the experiment are demographic survey, think-aloud programming activity and the feedback questionnaires.

1. Demographic survey – After getting the participant's consent, to understand the background and abilities questions related to proficiency in the programming were asked and the answers were recorded in the Qualtrics.

2. Think-aloud Programming Activity - Think-aloud activity helps in "obtaining a real-time insight into the knowledge that a subject use and the mental process applied while performing a process of interest" (Hughes, J., & Parkes, S., 2003). It provides important information about the participant's behavior. Though, sometimes think-aloud puts a burden on participant but in our case, there was no other way to understand the thinking process behind student actions (Van Den

Haak, M., De Jong, M., & Jan Schellens, P. (2003). Each student solved total 6 programs and for each program students were given 4 minutes. The students were informed that their performance is measured based on time, therefore, it is important to complete the activity as quickly as possible.

3. Feedback questionnaires: After each program students were given questions related to the program. These questions helped to understand the thinking process of the students and how they interpreted each error message. Also, we asked them if the error message from the previous program helped to correct the error message in the latest program. This question was asked only after 2nd, 4th and 6th program.

3.2 Procedure

Demographic survey: The survey contained total five questions. These questions were asked to decide the proficiency of the participant in the programming. The questions asked about the proficiency in PLP assembly language, high-level programming language, proficiency in the integrated development environment, and assembly language known to the participant other than PLP. If the user selects MIPS for other known assembly languages, then one more question was asked about their proficiency in the MIPS because PLP assembly language is like MIPS.

Actual questions asked to the participant are present in the appendix.

Think-aloud Experiment: In this experiment the participants were informed that this is a think-aloud activity. The students were instructed to verbally express all the thoughts and actions they are performing. During the experiment their thinking process was

recorded. To understand the actions performed by the students their screen was recorded along with voice.

Feedback Questionnaire: After completing each programming activity the participants were provided the feedback questionnaires. The questions were mainly focused on understanding how participants interpreted the error messages. Below are the actual questions that were presented to the students. The fourth question was asked only after completion of the 2nd program, 4th program, and 6th program. This question helped to understand whether the previous error message had any impact on the current program. This question gave us insight whether the error message in the previous program helped in solving current program or not.

1. Could you explain in your own words what was the error in the program?
2. How did the error message help you to understand the error?
3. How did the error message help you to fix the error?
4. Please describe the impact, if any, that fixing the error in the previous program had on fixing the error in this progr am (to be asked for the 2nd, 4th and 6th programs).

3.3 Materials

All the material used in this experiment was approved by the Institutional Review Board (IRB). The consent form was designed according to the guidelines provided by IRB. The demographic survey was to understand the participant's abilities. During the think-aloud experiments, each participant was given 6 programs. Please refer to appendix A for details regarding programs and error messages provided to the students. Each participant was given following type of programs in the same order.

1. Label Program: Each student was provided two programs with incorrectly labeled programs.

2. Instruction Program: Each student was provided with two programs with incorrect instructions.

3. Register Program: Each student was provided with two programs which contained incorrect register.

The experiment tested three types of error messages. These error messages were provided in different order. Table *1* explains the design of the order of the programs provided to the students.

Those three type of error messages are:

1. *Default type*: These are original error messages present in the PLP tool, which are designed from the developer perspective than user perspective.

2. *Link type*: These error messages contained a detailed description of the error along with a link to the online manual.

3. *Example type*: These error messages contained a detailed description of the error along with an example of the similar type of error.

Please refer appendix for actual programs given to the participants and the corrections required to the program to solve it. The programs and error messages were designed based on the feedback from the professors, teaching assistants review who has taken assembly language class previously and from literature review. The professor and TA provided the feedback that these are the most frequently occurring errors (Chabert, J. M., & Higginbotham, T. F. (1976, April)).

	Program Question 1 & 2	**Program Question 3 & 4**	**Program Question 5 & 6**	**Description**
Subject 1	P1 with M1 P2 with M1	P3 with M2 P4 with M2	P5 with M3 P6 with M3	For programming question 1 & 2, errors will be displayed using Message Type 1. For programming question 3 & 4, errors will be displayed using Message Type 2. For programming question 5 & 6, errors will be displayed using Message Type 3.
Subject 2	P1 with M1 P2 with M1	P3 with M2 P4 with M2	P5 with M3 P6 with M3	For programming question 1 & 2, errors will be displayed using Message Type 1. For programming question 3 & 4, errors will be displayed using Message Type 2. For programming question 5 & 6, errors will be displayed using Message Type 3.
Subject 3	P1 with M1 P2 with M1	P3 with M2 P4 with M2	P5 with M3 P6 with M3	For programming question 1 & 2, errors will be displayed using Message Type 1. For programming question 3 & 4, errors will be displayed using Message Type 2. For programming question 5 & 6, errors will be displayed using Message Type 3.

Table 1- Treatment Groups

3.4 Participants

To understand the effects of error messages on students it was important to select participants who have used the PLP tool before. If the participants didn't know how to use the tool, then it could have put an additional burden of understanding the tool. To avoid this, we chose participants from CSE 230 class. The students were using PLP tool to complete the assignments. To recruit the students, we made announcements on the slack

channel and on the blackboard with the help professor. Also, I went to the class and read the verbal announcement script. The students were informed that the experiment won't affect their grade in any way in any class. Also, it was informed that no extra credits will be given if they choose to participate in the study and on completion of the experiment 10$ target gift card will be given. The students were informed that they can stop participating at any time they want, and they have right to not answer questions which they do not want to answer. I received total 4 students for the study. Each student first signed the consent form and then also signed a digital consent form. After which they completed the demographic survey, programming activities and questionnaires.

3.5 Transcribe, Segment and Code verbal data

To analyze the steps taken by the student while solving the error in a program it was important to code them. The code part helped to understand the action performed by the students. Steps as prescribed by Chi, M. T., 1997, were followed for coding verbal data. Those steps are:

1. Reducing or sampling the protocols
2. Segmenting the reduced protocols
3. Developing a coding scheme
4. Operationalizing evidence in the coded protocols that constitute a mapping to some chosen formalism
5. Seeking pattern(s) in the mapped formalism
6. Interpreting the pattern(s)
7. Repeating the whole process, perhaps coding at a different grain size.

To understand the steps performed by the students each activity performed by the student was coded using below tables. This helped to get an insight of metal model of the students while solving problems in PLP tool.

Table *2* shows the basic steps used for coding the recorded data. The step was coded as “Examine” whenever student tried to understand the program or error message. Like at the beginning of each program there was a program description, if a student starts to read it then that was marked as examine step. Also, after assembling button if the program throws an error and the student started to read and understand it, it was marked as examine. Another instance where the step was marked as examine was when a student tries to understand the code. The “Examine” activity is the one which occurred the most in the coding part.

Code	Definition	Examples
Examine	Step where a student trying to read the description of the program or trying to understand program or error message	"so this problem is asking me to perform addition and subtraction operation so the result of addition should be stored in s1 then result of subtraction should be stored in s2.."
Explore	Step where student opens online PLP manual or Quick reference section from help section of the PLP tool or opened Link to the online manual in the *Link type* error message	"I think It should be okay so lets just look at s1 okay so first its gonna increment so I should get 70 in s1 which I do so that's good okay"
Hypothesize	Step where student trying to assume or guess	"so umm I guess one solution is just to line 28 we can just write a0 since we are at the end of the program & see how that works"
Repair	Step where student perform changes to the program	"I am gonna use s6 instead, So I am just gonna change that"
Evaluate	Step where student checked the program after making changes to the program to validate the program	"ok..it's running…I am gonna watch first s1.. just wanna make sure I did everything right…ok…I am gonna right…it should decrement it result 50 so that is right …if t1 is actually 0 result is 70.."

Table 2- Basic Steps Code

The step was coded as "Explore" when students tried to open the online PLP manual, the quick help reference section or clicked on the link provided to the PLP manual in the *Link type* error message. The explore step indicated that student is trying to

understand the problem with the help of material available and not by just assuming or using his previous experience.

Whenever student assumed something or tried to guess based on the behavior of the program or error message that activity was marked as “Hypothesize”. Hypothesize shows the gap in understanding or it shows that student is trying to understand based on her experience. The hypothesis was either correct hypothesis or incorrect hypothesis, but both were marked as hypothesize.

The activity was marked as “Repair” only when the student tried to make changes to the code. The changes could be either on line where the error occurred or at some other place in both cases it was marked as repair.

After solving the error if student decided to check the results or decided to verify whether the error is gone or not that step was marked as “Evaluate”.

The examine steps are coded as program description, code, error message or search information. *Table 3* provides the definition and example for the codes. The step was marked as “Program Description” when the student read the problem statement provided in the program. The step was marked as “Code” when student made changes to the program. Whenever student received an error message that step was marked as “Error Message”. The step was marked as search information whenever student tried to search on PLP online manual or PLP quick help section of PLP tool.

Code	Definition	Examples
Program Description	Step where student read the problem statement given at the start of each program	"The program will turn on the LED least significant bit position & LED at most significant bit position in LED array"
Code	Step where student make changes to the program	Added colon after the addition.
Error message	Step where student received error message	ohh it is saying subui doesn't exist because its subu which then gives error register not recognized
Search Information	Step where student investigated the online manual or quick help reference section on PLP tool	"umm register PLP… online PLP manual"

Table 3- Examine Step Codes

During the think-aloud experiment while speaking loudly each student performed actions which are recorded in screen recording. To understand the behavior of the students, each action performed by them further coded using the Table *4*. The action was recorded as "Expected" when the student acted which was expected by us. The action was termed as "Gaming" only when the student performed an action without understanding the error message or the results of the action. The action was termed as "T&E" when the student understood the error message but not sure about the correct step. "Correct Independent" action was those when the student performed correct action, but it was not dependent on his understanding. The "Incorrect Interpretation" was used to indicate the action performed by the student because of wrong interpretation. The action which is incorrect and independent of the error message was marked as "Incorrect Independent". The actions which are incorrect because of lack of attention was marked as "Silly Mistakes".

Code	Definition
Expected	Correct action performed by the student based on correct understanding
Gaming	Action performed by the student without understanding
T&E	Random action performed by the student with understanding the error message
Correct Independent	Correct action performed by the student without understanding the error message
Incorrect Interpretation	Wrong action performed by the student because of wrong understanding
Incorrect Independent	Wrong action performed by the student independent of understanding
Silly Mistake	Wrong action performed by the student because of not giving sufficient attention

Table 4- Correct and Incorrect Steps Code

Table *5* provided information related to the status of error message read by the student. An error message is a statement which provides information regarding the error present in the program. It was important to understand whether the student read the error message completely, partially, or Ignored it. The step was marked as a "Complete" when the student read the error message completely. The step was marked as partial when the student only read part of the error message. The step was marked as "Ignored" when student ignored the error message went directly to the program to understand and correct the error.

Code	Definition
Complete	Student read the given error message completely
Partial	Student read only some part of the given error message
Ignore	Student ignored the error message completely or scrolled up and down in the error message window

Table 5- Read Error Message Code

Figure 6 shows a sample coding of student 23. Similar to these all 24 programs were coded for analysis.

Speak	Screen	Complete Partial Ignore	Expected Gaming Incorrect Interpretation T&E Correct Independent Incorrect Independent Silly Mistake		Examine Explore Hypothesis Repair Evaluate	Program Description Code Error Message Search Information
Okay so addition & subtraction the addition should be stored in register s1 the result should be $s2			Expected		Examine	Program Description
okay so then we have t1 t2 t3 we add t2 & t3 and stored in s1 we subtract t3 t2 stored in there			Expected		Examine	
okay lets say compile & there is an error somewhere so	Clicked on assemble button.		Expected		Examine	
	Received error message of example type	Ignored				Error Message
umm so okay we have check the register I guess t1 look good t2 t3 s1 t2 t3 s2 t3 t2 umm exit and then jump to exit			Expected		Examine	
umm exit is blue again meaning now add colon	Added colon on line 18 after exit		Correct Independent		Repair	Code
press compile again	Clicked on assemble button.		Expected		Examine	
works then open up inspector menu to see addition so then we have s1 it has value 105 that makes sense s2 has value 15 it seems like it is working example			Expected		Evaluate	

Figure 6- Sample coding

Chapter 4

RESULTS

This chapter is divided into five parts. The first section contains students' process model which is generated based on qualitative study of coded data. Each time students received an error message was treated as an instance, i.e. one student may have multiple such instances for a single program. Based on the total number of instances, the second section explains percentage of times student ignored or read the error message completely or partially for different types of error messages. The third section explains which type of error message helped students to resolve the error. The fourth section explains which type of error message student found confusing. The fifth section provides insight on which error message is more effective and why, based on the process model.

4.1. Student's process model to resolve the errors:

In this study, 4 students participated, and each student was given 6 programs. Each program contained errors and participants were informed that they had to fix the error within 4 minutes. They were also informed that the time taken to fix the error will be monitored and used as an indicator of performance. Each participant received two programs with the same type of error and same category of error message. After each program students were asked questions about the error message to understand their thoughts and reactions to the process of reading the error message and solving the error. After the experiment, each student's activity was coded as explained in the previous section, resulting in codes for 24 programs, and qualitative analysis was performed on the data to understand impact of each type of error message.

Figure *7* shows the process model generated by transcribing, coding and analyzing the steps performed by the students. Based on the steps coded and feedback which is presented in the following sections the process model was generated. This model helped categorize different actions performed by the students and impact of different types of error messages on student's ability to understand and resolve the error. Students performed two main actions after each time they assembled the program. The first action was examining the error message and the second to choose a method to resolve the error.

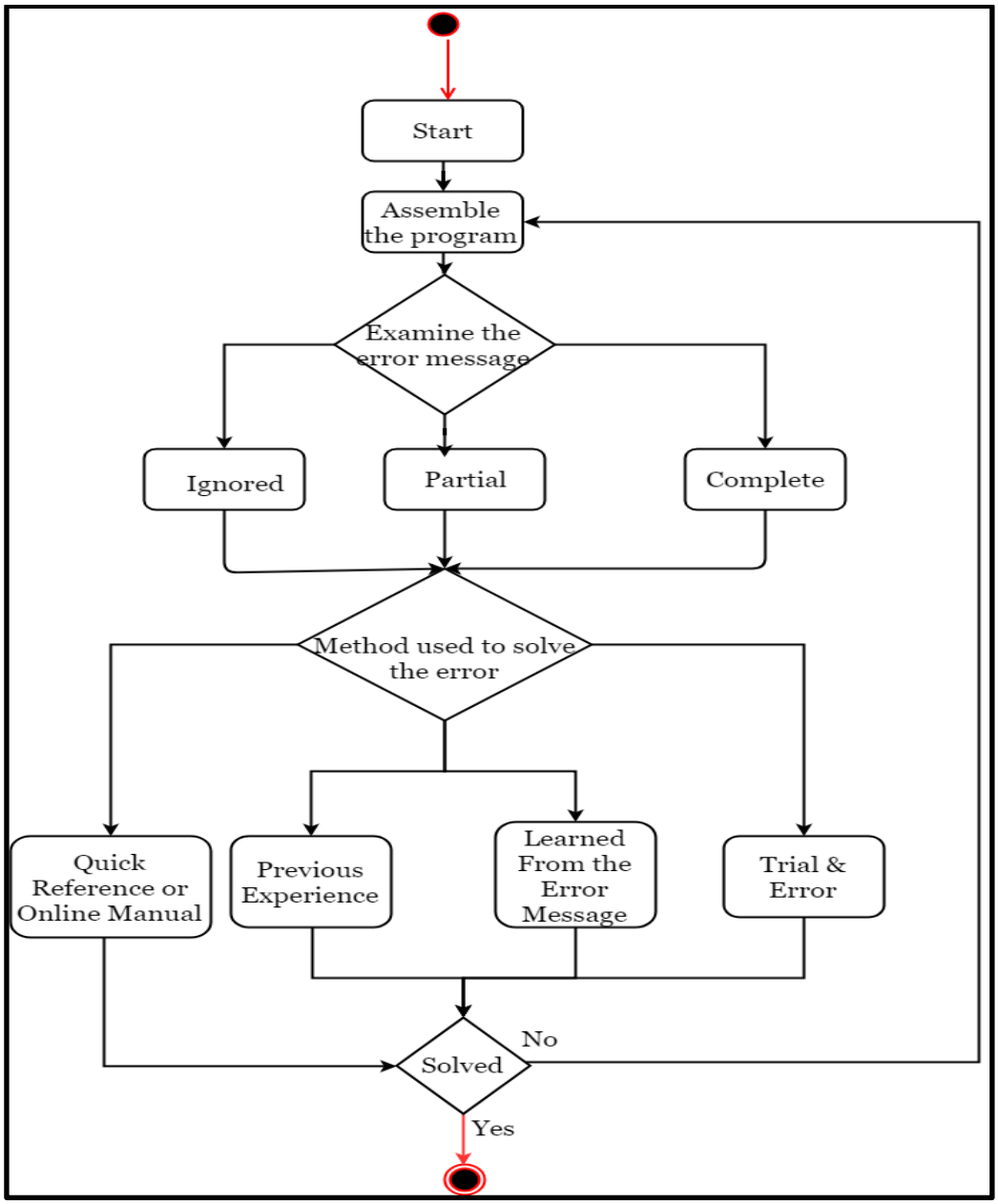

Figure 7- Process model of students for solving the error

4.2. **Error Message Read**

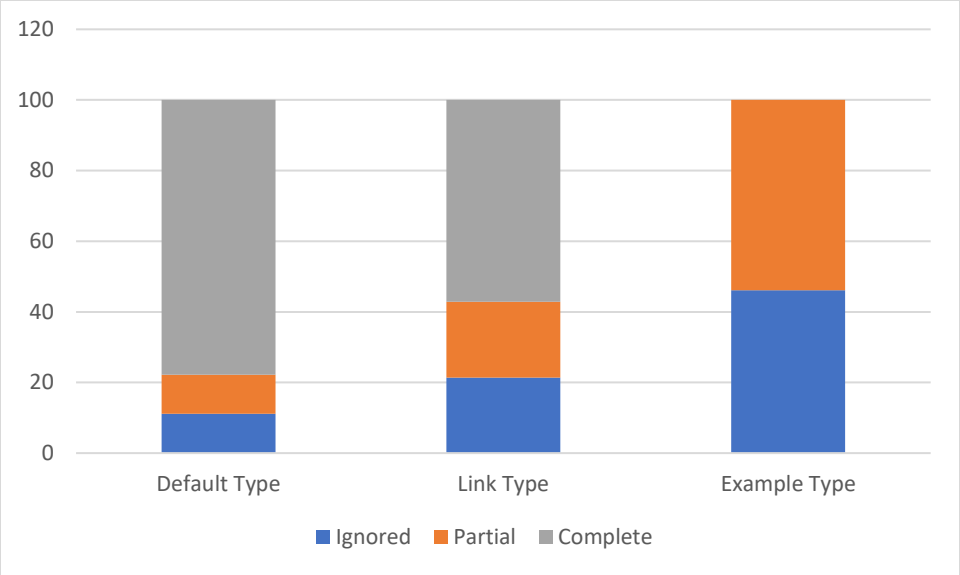

Figure 8- Error message read

	Ignored	**Partial**	**Complete**
Default Type	11.11111	11.11111	77.77778
Link Type	21.42857	21.42857	57.14286
Example Type	46.15385	53.84615	0

Table 6- Number times error messages were read

Table *6* shows percentage of instances in which student read the error message completely, partially and ignored. Each time student received an error message was counted as an instance. Sometimes students did not read the message loudly, in which case the determination was made based on the feedback and screen capture. In case of "*Example type*" no one read the error message completely. In case of "*Default type*" highest percentage of students read the error message.

4.3. Error message found confusing:

The feedback part from the questionnaire and students' steps were analyzed to understand if the students found the error message confusing or not. Table *7* shows the number of programs in which students found the error message confusing.

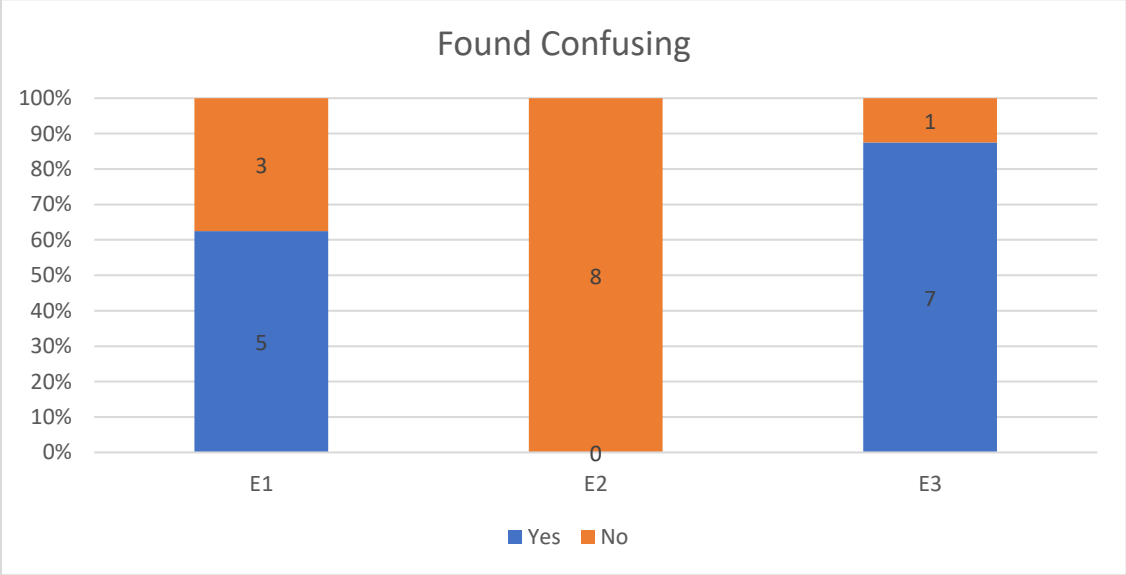

Figure 9-Number of programs in which student found error message confusing

Table *7* shows that student did not find the "*Link type*" error message confusing. On the other hand, students found "*Example type*" confusing.

	Yes	No
Default Type	5	3
Link Type	0	8
Example Type	7	1

Table 7- Number of programs in which student found error message confusing

In case of *Default type*, in 5 programming activities they found it confusing. Students provided feedback that the error message is too short and did not understand the

error in the program based on the error message. In three cases where students found the *Default type* error message helpful used their experience from previous assignments to solve the error. One student provided feedback that "The error message was poor. The syntax highlighting was more helpful." for *Default type* error. In case of *Link type* error students found the error message neatly written explaining what is wrong with the program and where to look. One of the student provided the feedback that "The error message helped to confirm to me that what I thought was an error really was an error. If I did not have the error message, I would not be as confident that the only error was the subiu token." for *Link type* error. In case of *Example type* students found it very confusing. The students were confused by the example. One of the student provided feedback that "The error message was excessively verbose. It told me that there was an invalid register name, then provided a bunch of unhelpful nonsense after the actual error. This was confusing as it did not relate to the problem I was having." for *Example type*.

4.4. Error message found helpful:

The feedback part from the questionnaire and students' steps were analyzed to understand if the error message helped or not. In the feedback questionnaire the third question was how the error message helped you to fix the error. The student explained in it how it helped them and what part of the error message they found unhelpful.

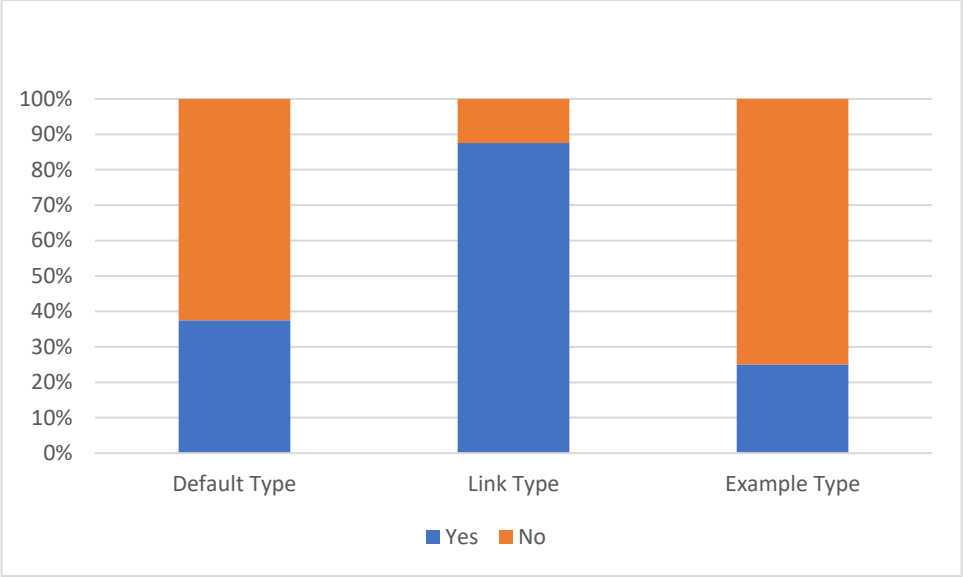

Figure 10 - Error message helped

	Yes	No
Default Type	37.5%	62.5%
Link Type	87.5%	12.5%
Example Type	25%	75%

Table 8-Error message helped

Table *8* shows highest percentage of times student found the "*Link type*" error message helpful. One student provided feedback that "The error message helped me to understand the error because as I read further down the error lines, I saw that there was a problem around the addu word. I found in the program where the addu word was, and found the problem described above and made the necessary changes to the program." The students found "*Default type*" and "*Example type*" unhelpful. One student provided feedback that "It did not help since it provided no information about the error except that it existed." for that *Example type* error message. Similarly, another student provided feedback for *Default type* is that "The only thing useful was the line number."

4.5. Detailed analysis of three error message types and their effects on student's problem-solving skills:

a. Detail description of Student's process model:

Figure *11* was designed using coded data. It shows the thinking process of the students participated in the study. It was observed that once the compiler generates some error message, students either read the error message completely or they read the error message partially, or they completely ignored the error message.

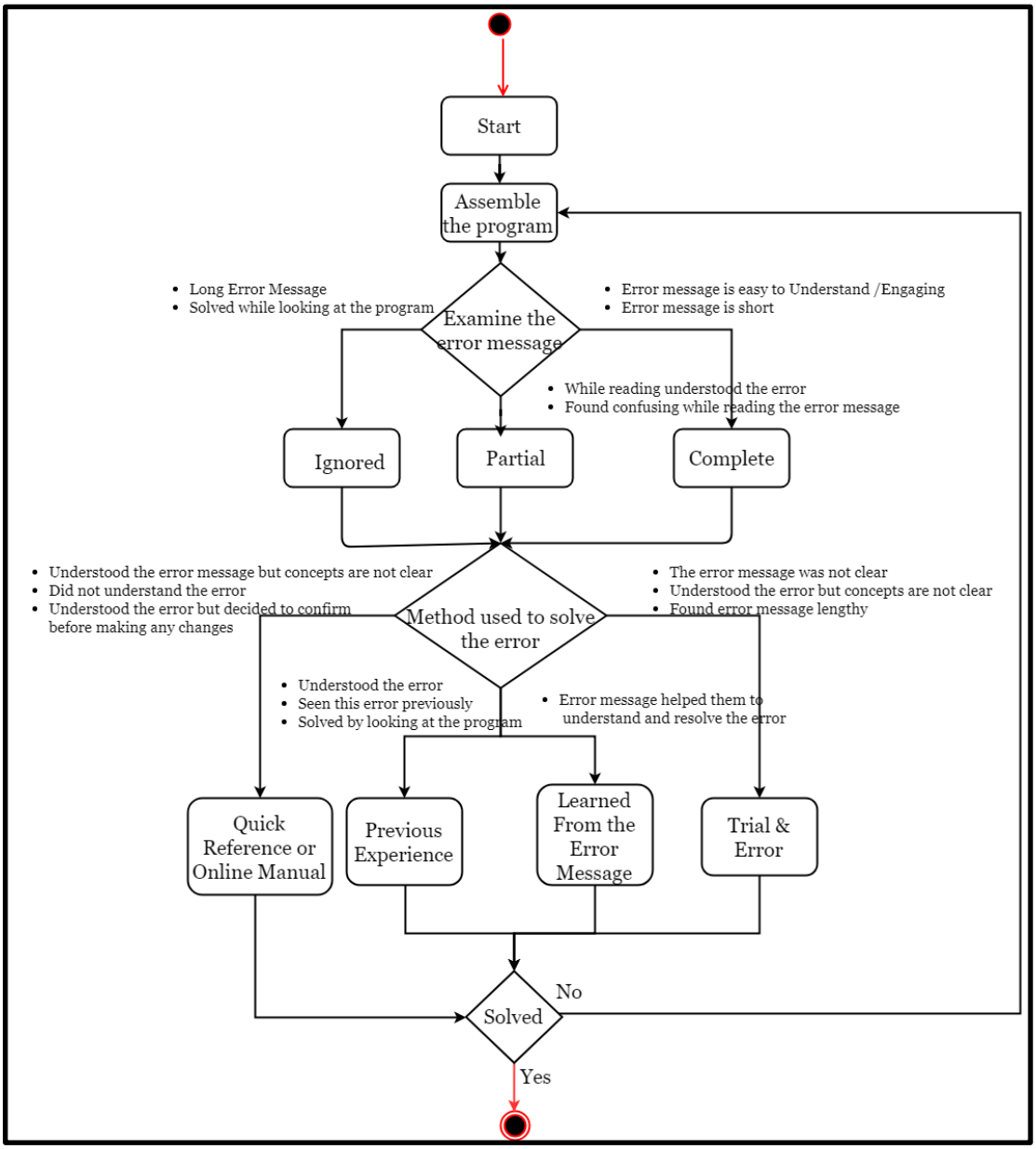

Figure 11 - Detailed Student process model

1. The students read the complete error message in two cases:
    a. When the error message was short.
    b. When the error message was easy to understand.

2. The students read error messages partially in two cases:
    a. When the students understood the error while reading it in that case the rest of the error message was ignored, and they directly went to the part where the error occurred.
    b. When the students found the error message confusing, they ignored the rest of the error message and went directly to the program to understand and resolve the error.
3. The students ignored the error message in two cases:
    a. When the students found the error message very long.
    b. When the students solved it while reading the program.

After examining the error message, based on the situation, students chose one of the four methods to resolve the error. Those methods and situations are given below.

1. Quick reference or online manual – Students used this method in three situations.
    a. When students did not understand the error.
    b. When students understood the error, but the concepts were not clear.
    c. When students understood the error and decided to confirm the understanding or concepts from the online manual or quick reference.
2. Previous experience: Students used their previous experience in three situations.
    a. When students understood the error and they had similar type of error while doing the assignments.
    b. When students got the same error message which they had seen previously.
    c. Resolved the error while they were trying to understand the program.

3. Learned from the error: this method was observed when students understood and corrected the error from the error message.
4. Trial & Error: student opted for this method in three situations.
    a. When they read the error message but did not find helpful.
    b. When students understood the error message but did not know how to fix it.
    c. When students found the error message confusing or tedious.

Based on the coded data, it was observed that students who understood the error message and learned from it were more confident at the end. This model also helped to understand the effects different types of error message had on students.

b. Effects of different error message types on students' process model:

Figure 12 shows the effects of *Default type* error message on the students' ability to understand and resolve the error. It shows that 78% times students read the complete error message and 58% times they chose to use trial & error approach. Table *9* shows percentage of times students chose 4 methods to resolve the error after receiving *Default type* error message. Table 10 shows steps performed, and feedback provided by all the students for *Default type*. After giving the *Default type* error message to the students, the highest percentage of times students opted for Trial & Error method. The reason for that could be observed from the Table 10 feedback column. The main reason for this was students did not understand the error. One student provided feedback that "It was similar to the previous error ("*Default type*"). It did not help me to identify the problem on my own but led me to a similar solution to the previous program". When the *Default type* error message was given to the students, they tried to guess or assume the answers instead of understanding the error.

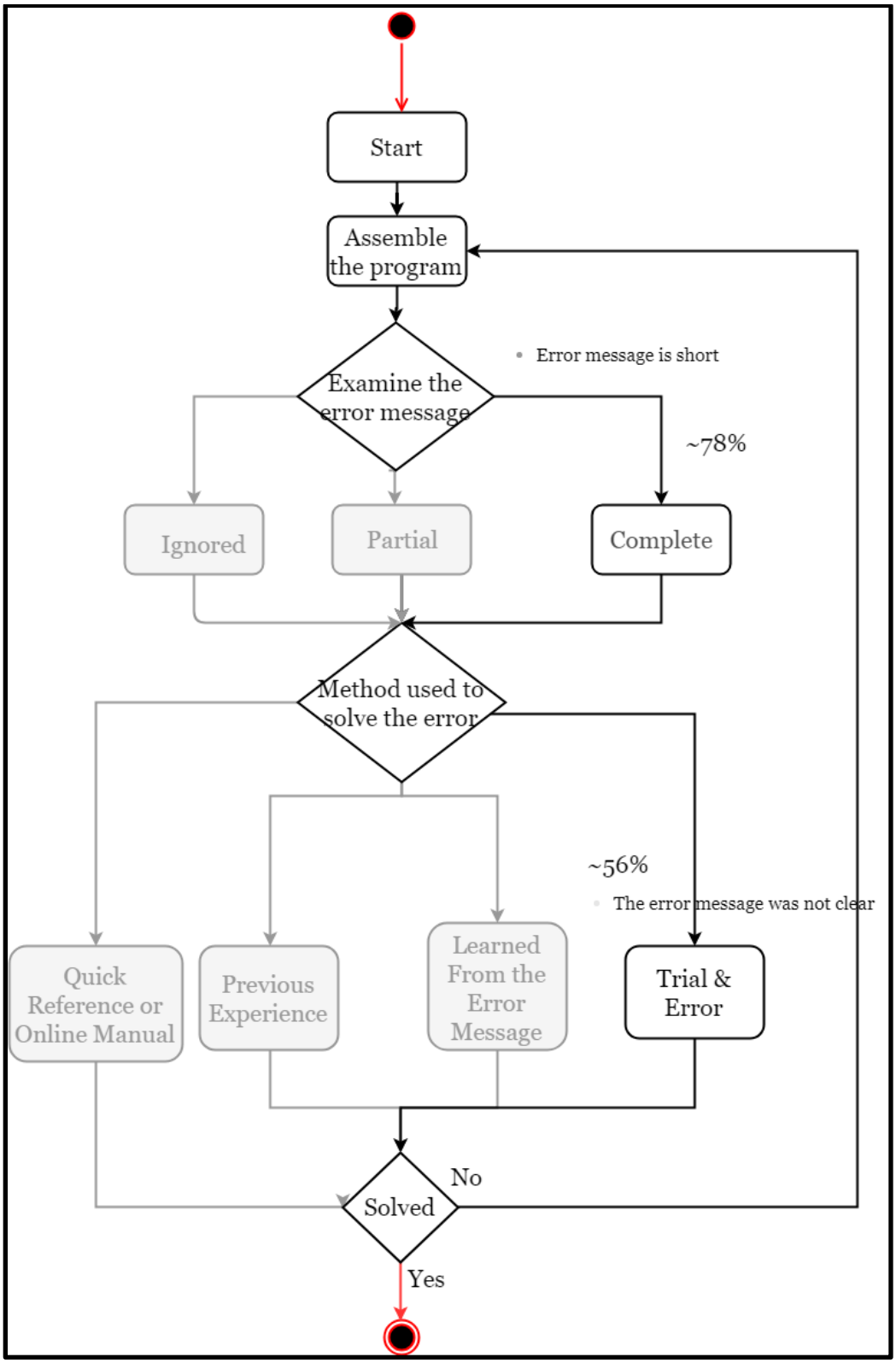

Figure 12- Percentage distribution of *Default type* on Student's process model

Quick reference	Previous Experience	Learned from the error	Trial & Error
1	2	1	5
11.11111111	22.22222222	11.11111111	55.555555556

Table 9- Percentage distribution of *Default type* error message

Student	Steps	Feedback
11	Solved while reading the program	Even though I did notice the error message, I did not read it. I just looked through the program to identify the error.
11	..okay I see the problem the exit statement doesn't have semicolon.	The error message helped me to fix the error by telling me what line in the program I needed to go to in order to fix the error.
23	so I assume & we have to use s7 because s is probably stop at 7	It told me that the register is invalid. This led me to believe that the $s does not go past 7.
23	so umm I guess one solution is just to line 28 we can just write a0 since we are at the end of the program & see how that works	It barely helped since I knew I needed to use a different register but had no idea which one to use. I forgot what registers were safe to use with jump and link. Since I could not remember what register I could use I re purposed an existing register instead. I know there is probably a better way but I always have to consult my notes on what each register is for. I knew what the problem was but had no good way of solving it.
36	s8 i guess invalid register	It was similar to the previous error. It did not help me to identify the problem on my own, but led me to a similar solution to the previous program.
36	why would v2 be an invalid register	It didn't help particularly much since I couldn't recall if it was that there was no register with the name $v2 or if it was some issue with declaration that didn't let me store information for some reason.
42	Missing colon on line 20	The error message was poor. The syntax highlighting was more helpful.
42	Added colon	The only thing useful was the line number.

Table 10- Steps and feedback from the student for the Default type error message

Figure 13 shows the effects of *Link type* error message on the students' ability to understand and resolve the error. It shows that 58% times students read the complete error message and 50% times they chose to use trial & error approach. Table *11* shows percentage of times students chose 4 methods to resolve the error after receiving *Link type* error message. Table 12 and Table 13 shows steps performed, and feedback provided by all the students for "*Link type*". After giving the *Link type* error message to the students, the highest percentage of times students learned from the error. The reason for that could be observed from Table 12 and Table 13 feedback column. The main reason for this was students understood what the error is and where it occurred on the line. One student provided feedback that "The error message helped me to understand the error because as I read further down the error lines, I saw that there was a problem around the addu word. I found in the program where the addu word was, and found the problem described above and made the necessary changes to the program.". Similar feedback was provided by another student "This error message was better. Instead of a parse error, the assembler issued an error that stated that the opcode did not exist. After playing with a few logical ways or rephrasing the mnemonic, I used the instruction set reference, which reminded me that this instruction was not implemented. I then used a complementary instruction to complete the task."

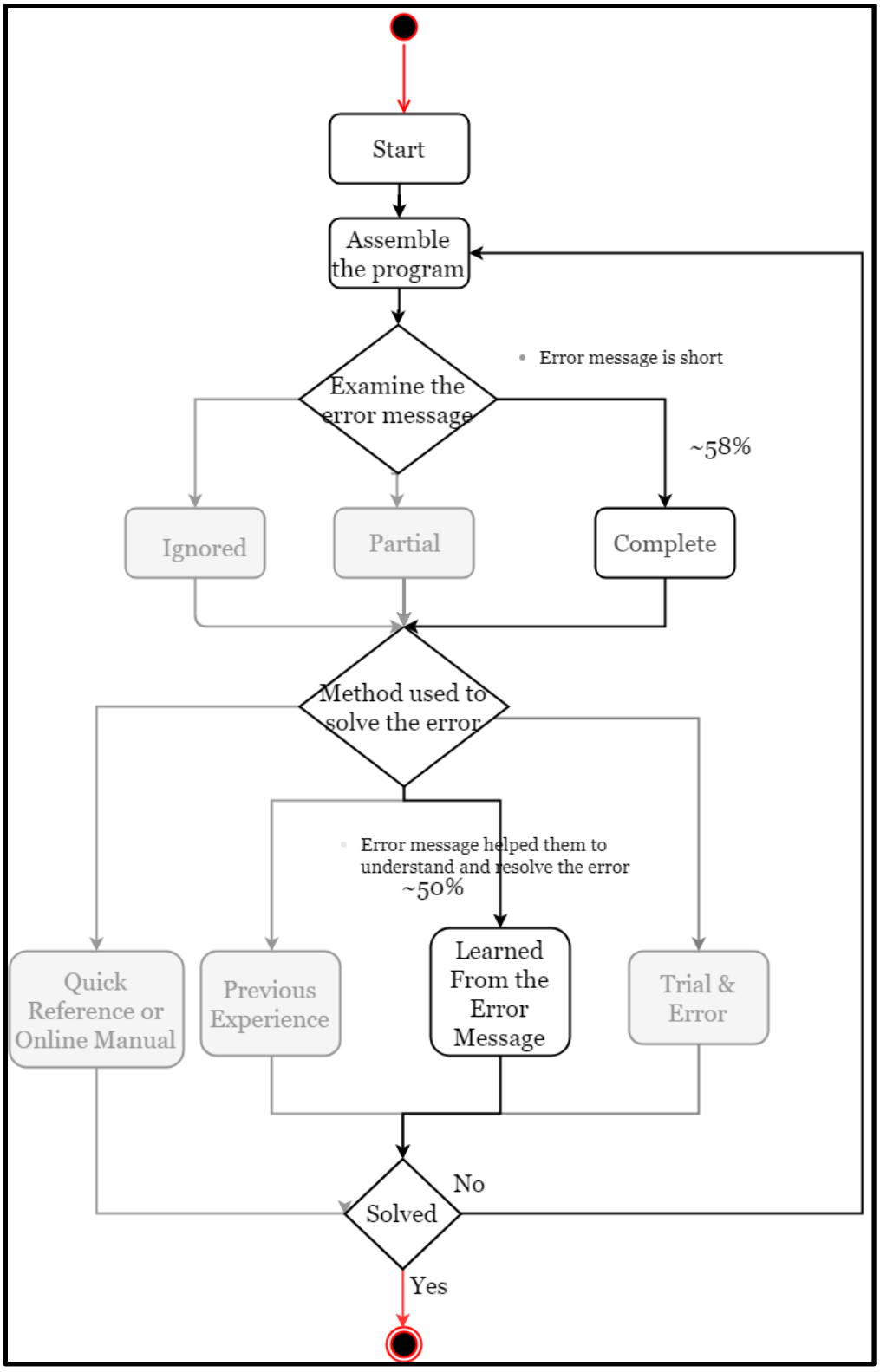

Figure 13- Percentage distribution of Link type on Student's process model

Quick reference	Previous Experience	Learned from the error	Trial & Error
1	1	7	5
7.142857143	7.142857143	50	35.71428571

Table 11- Percentage distribution of error solving method used in Link Type

Student	Steps	Feedback
11	ohh I think the problem let me ran it the problem might be the subtraction I don't think you have subiu I don't think you have immediate there...okay yeah the error at line 14 it say error occured around subiu...	The error message helped to confirm to me that what I thought was an error really was an error. If I did not have the error message, I would not be as confident that the only error was the subiu token.
11	..ohh because we are trying to add immediate value I need addiu...	The error message helped me to understand the error because as I read further down the error lines, I saw that there was a problem around the addu word. I found in the program where the addu word was, and found the problem described above and made the necessary changes to the program.
23	ohh it is saying subui doesn't exist because its subu which then gives error register not recognized	It pointed to the line and said what part was making an error. It also helped by saying what the function expected to receive. That helped by showing me what kind of number it wanted. It did not help try to figure out what function I meant to use and I had to remember that subu is used for subtraction.
23	umm press compile just to check wrong instruction line 22 addui instruction was not defined in plp umm error occured around word addui addui doesn't exist	It did not help much since the main issue was trying to remember what function adds a number to a register. Spelling the function was the hard part.

Table 12-Steps and feedback from the student for the Link type error message

Student	Read Error Message	Steps	Feedback
36	Complete	oh addition didn't have colon so it wasn't reading it as a lable	It pointed out the missing colon.
36	Ignored	I caught the error before assembling. There was a jump to exit label within the exit label. The exit label was entirely redundant. In addition the exit label was missing a colon that would have prevented it from assembling.	I did not have an error message because I fixed the program before running it.
42	Complete	This thing doesn't have immediate subtract. This architecture doesn't have an immediate subtract.	This error message was better. Instead of a parse error, the assembler issued an error that stated that the opcode did not exist. After playing with a few logical ways or rephrasing the mnemonic, I used the instruction set reference, which reminded me that this instruction was not implemented. I then used a complementary instruction to complete the task.
42	Complete	That's addiu	It gave me the line number and let me know that the opcode was incorrect. Once I knew the type of error, fixing it in this case was trivial.

Table 13- Steps and feedback from the students for the Link type error message (Continued)

Figure 14 shows the effects of *Example type* error message on the students' ability to understand and resolve the error. It shows that 46% times students chose to ignore the error message which is highest of all the error message types. Also 56% times students

read the error message partially and they never read the error message completely. Table *14* shows percentage of times students chose 4 methods to resolve the error after receiving *Example type* error message. Table 15 and Table 16 shows steps performed, and feedback provided by all the students for "*Example type*". After giving the *Example type* error message to the students, the highest percentage of times students used Trial & Error method to resolve the error. The reason for that could be observed from the Table 15 and Table 16 feedback column. The main reason for this was students got confused after reading the example part of the error message. One student provided feedback that "It did not help since it provided no information about the error except that it [sic]exsisted.". Similar feedback was provided by another student "It partially didn't. It suggested using sub instead, which would require the use of another register. If someone wants to use a subiu it implies they are using an immediate, which does not require a register. Therefore, recommending addiu but using the inverse of the immediate would be more useful.". One student while doing the think-aloud said that "So it says, well first thing it says is before correction and after correction. I am not sure about that. Line 28 register not recognized" which shows that the student got confused with the example.

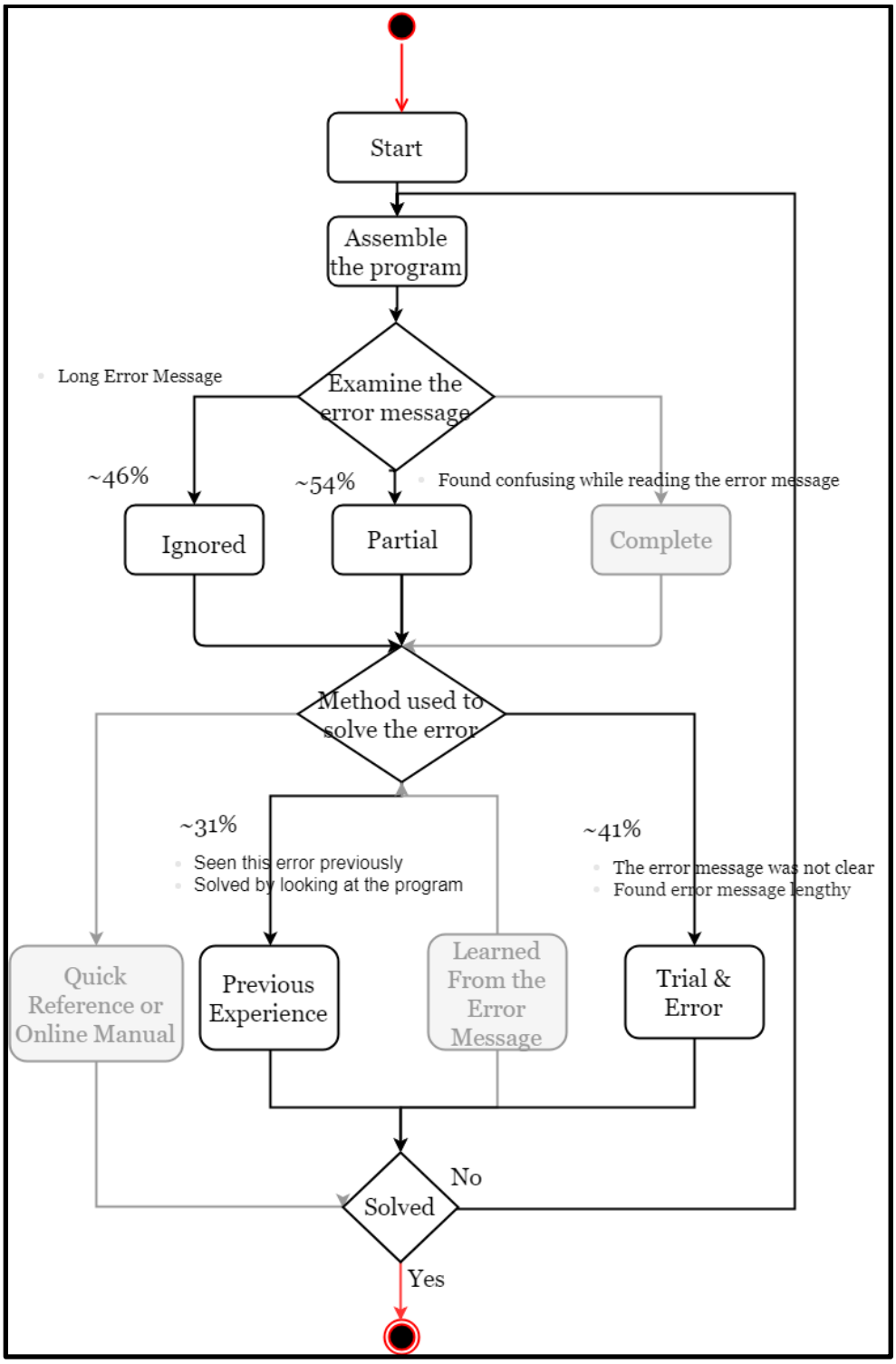

Figure 14- Percentage distribution of Example type on Student's process model

Quick reference	Previous Experience	Learned from the error	Trial & Error
1	4	2	6
7.692307692	30.76923077	15.38461538	46.15384615

Table 14-Percentage distribution o Example type error message

Student	Steps	Feedback
11	So it says,well first thing it says is before correction and after correcction. I am not sure about that. Line 28 register not recognised .	The error message helped me to understand what line the error was on, and what the problem was with that line. Specifically, it told me that there was something wrong around the $v2 register, which I understood to mean there was something wrong with using $v2 in that specific line of code.
11	so it says the error is on 24 something around safe word..s8 is not recognized I am guessing its kind of similar problem s8 is not a register	The error message helped me to see what line the error was at, but I did not really need to see what register specifically was causing the problem, it just helped confirm to me what I thought the problem already was. I had come to this conclusion based on what I have learned in class about the PLPTool and invalid registers, and that there was a problem with an invalid register in the last program.
23	but yeah it need to say something to go that zero t or s or v so that would be	It did not help since it provided no information about the error except that it exsisted.
23	umm so okay we have check the register I guess t1 look good t2 t3 s1 t2 t3 s2 t3 t2 umm exit and then jump to exit	It provided no hint as to what was the problem. It just said it existed.

Table 15- Steps and feedback from the student for the Example type error message

Student	Steps	Feedback
36	what is it there is not sub immediate in plp	It partially didn't. It suggested using sub instead, which would require the use of another register. If someone wants to use a subiu it implies they are using an immediate, which does not require a register. Therefore recommending addiu but using the inverse of the immediate would be more useful.
36	does not have oh its typo	It reccommended addiu instead
42	Stayed silent for 45 seconds then changed v2 to v1	The error message was excessively verbose. It told me that there was an invalid register name, then provided a bunch of unhelpful nonsense after the actual error. This was confusing as it did not relate to the problem i was having.
42	Ohh okay there are only seven s registers	It helped me to understand that the register was incorrect. It had a lot of extraneous information that was confusing and made the program more difficult to debug by scrolling other errors out of the limited error window. This added at least 30 seconds to the amount of time required to fix the program.

Table 16- Steps and feedback from the student for the Example type error message (Continued)

Chapter 5

CONCLUSION and FUTURE WORK

This study used think-aloud protocol to understand the process used by the students to resolve the errors in assembly language programming. The experiment conducted on four students helped to design student's process model. The process model helped to understand different strategies used by students to resolve the errors.

Based on the analysis of feedbacks and steps taken by the students, it shows the impact error message types had on the students process to resolve the error, the study shows that the *Link type* error message was more helpful and engaging than *Default type* and *Example type*. The *Link type* error also helped to increase confidence level in the students. The study shows that *Default type* error message was too short and did not help students to resolve the error. On the other hand, the *Example type* was too lengthy, and students avoided to read the complete error message. In case of, *Default type* and *Example type* students prefer to use trial & error method to resolve the error, instead of understanding the error. The default type and Example type did not facilitate learning as evidenced from the feedback and steps coded. In case of *Link type* error message students understood the error which not only helped then to fix it but also helped them to learn from it, which indicates that the Link type facilitated learning from understand the error message. The study also shows that in case of *Example type* error message student were more confused by the example in the error message. Though the detailed description of the error was similar in *Link type* and *Example type*, the study shows that because *Example type* was lengthy students avoided to read the error message.

The impact of error messages on student learning is an extremely important and interesting topic to explore. To gauge this impact, however, a study needs to be conducted at a much larger scale, with longitudinal follow up. Ideally, students in three sections of

the assembly programming course will be presented with three different version of PLP-one with the default error message, one with the example type, and one with the link type error messages. Class averages on various projects, quizzes and exams, can serve as data points throughout the semester, and the study can gauge if there are trends associated with an error message type. For example, do students in one section achieve project milestones quicker, or score more on an average than students in another section. Focus group interviews can determine students' usage of resources like the quick reference and the manual, to see if one of the error message types drives more or better usage of these resources. The study can provide data on why students might learn better through one of the error message types.

REFERENCES

Becker, B. A. (2016, February). An effective approach to enhancing compiler error messages. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (pp. 126-131). ACM.

Chabert, J. M., & Higginbotham, T. F. (1976, April). An investigation of novice programmer errors in IBM 370 (OS) assembly language. In Proceedings of the 14th annual Southeast regional conference (pp. 319-323). ACM.

Munson, J. P., & Schilling, E. A. (2016). Analyzing novice programmers' response to compiler error messages. Journal of Computing Sciences in Colleges, 31(3), 53-61.

Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., & Parnin, C. (2017, May). Do developers read compiler error messages?. In Proceedings of the 39th International Conference on Software Engineering (pp. 575-585). IEEE Press.

Pettit, R. S., Homer, J., & Gee, R. (2017, March). Do Enhanced Compiler Error Messages Help Students?: Results Inconclusive. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (pp. 465-470). ACM.

McCall, D., & Kölling, M. (2014, October). Meaningful categorisation of novice programmer errors. In Frontiers in Education Conference (FIE), 2014 IEEE (pp. 1-8). IEEE.

Prather, J., Pettit, R., McMurry, K. H., Peters, A., Homer, J., Simone, N., & Cohen, M. (2017, August). On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In Proceedings of the 2017 ACM Conference on International Computing Education Research (pp. 74-82). ACM.

Barik, T., Witschey, J., Johnson, B., & Murphy-Hill, E. (2014). Compiler Error Notifications Revisited.

Traver, V. J. (2010). On compiler error messages: what they say and what they mean. Advances in Human-Computer Interaction, 2010.

Hristova, M., Misra, A., Rutter, M., & Mercuri, R. (2003, February). Identifying and correcting Java programming errors for introductory computer science students. In ACM SIGCSE Bulletin (Vol. 35, No. 1, pp. 153-156). ACM.

Nienaltowski, M. H., Pedroni, M., & Meyer, B. (2008). Compiler error messages: What can help novices?. ACM SIGCSE Bulletin, 40(1), 168-172.

Hartmann, B., MacDougall, D., Brandt, J., & Klemmer, S. R. (2010, April). What would other programmers do: suggesting solutions to error messages. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1019-1028). ACM.

Ebrahimi, A. (1994). Novice programmer errors: Language constructs and plan composition. International Journal of Human-Computer Studies, 41(4), 457-480.

Radosevic, D., & Orehovacki, T. (2011, June). An analysis of novice compilation behavior using Verificator. In Information Technology Interfaces (ITI), Proceedings of the ITI 2011 33rd International Conference on (pp. 325-330). IEEE.

Van Den Haak, M., De Jong, M., & Jan Schellens, P. (2003). Retrospective vs. concurrent think-aloud protocols: testing the usability of an online library catalogue. Behaviour & information technology, 22(5), 339-351.

Chi, M. T. (1997). Quantifying qualitative analyses of verbal data: A practical guide. The journal of the learning sciences, 6(3), 271-315.

Sohoni, S. (2014, June). Making the hardware-software connection with PLP. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 324-324). ACM.

Kadekar, H. (2016). Effects of error messages on a student's ability to understand and fix programming errors. ASU.

APPENDIX A

PROGRAMS, ERRORS AND FIXES

As part of this study, six programs were used in the experiments. In this appendix, all the programs, their errors and how those errors can be fixed are provided.

1. Label Program:

    a. First label program:

This was the first program given to the participant in the experiment. The program performs addition or subtraction based on the value in register $t1 and stores the result in register $s1. *Figure 15* shows the program. This program has an error at line number 20. Here word 'addition' is a label. It is missing a ':' to declare it as label.

main.asm <main program>

```
1
2   #Problem - perform either addition or subtraction operation.
3   #              Final result should be stored in register $s1.
4   #              0 - Addition
5   #              1 - Subtraction
6
7   .org 0x10000000
8
9   li $t1, 0
10  li $t2, 45
11  li $t3, 60
12
13  beq $t1, $0, addition
14  nop
15
16  subu $s1, $t3, $t2
17  j exit
18  nop
19
20  addition
21     addu $s1, $t2, $t3
22
23  exit:
24     j exit
25     nop
```

Figure 15- Program 1 label program

Figure 16 shows the error provided as a *Default type* to the participants. *Figure 17* shows *Link type* error provided for the first program to the participants. *Figure 18* shows the error message provided to the participant as *Example type*. To fix this error it was important to add a ':' after the label addition.

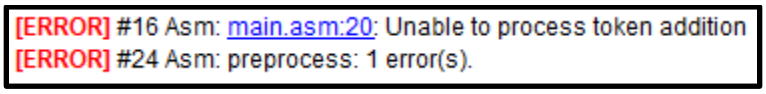

Figure 16- *Default type* Error Message for the Program 1 Label Program

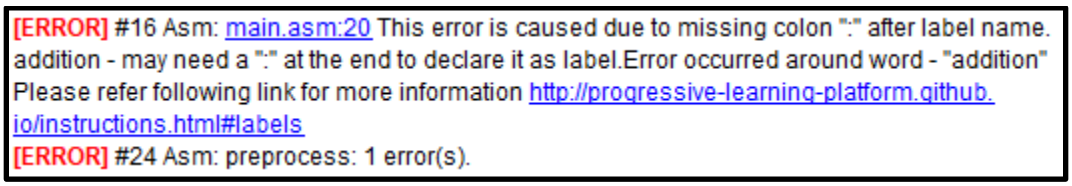

Figure 17- *Link type* Error Message for the Program 1 Label Program

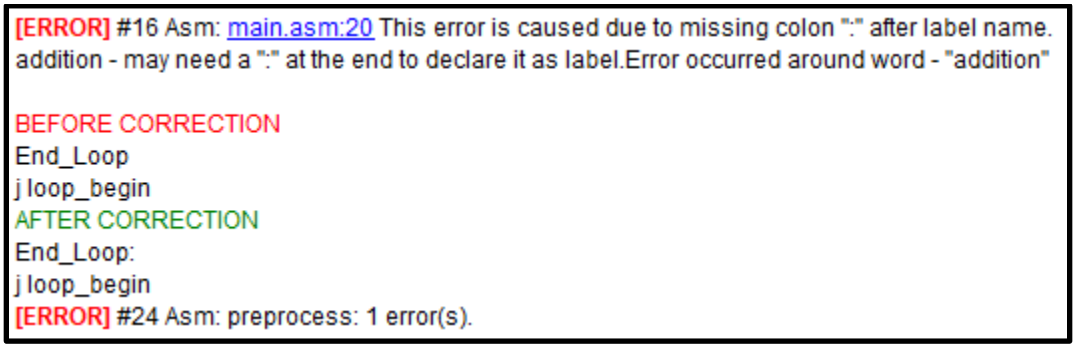

Figure 18- Example Type Error Message for the Program 1 Label Program

b. Second label program

Figure 19 shows the second program given to the participants. This program performs addition and subtraction operations. The result of the addition is stored in register $s1 and the result of the subtraction is stored in $s2. This program has an error at line number 18. Here the word 'exit' is a label. It is missing a ':' to declare it as the label.

Figure 20 shows the error message as displayed using the *Default type*. *Figure 21* shows the error message as displayed using the *Link type* and *Figure 22* gives the error message as displayed using the *Example type*. The correct way to fix this error is to place a ':' at the end of line 18 to make word exit as the label. So, line 18 will have 'exit:' after error correction.

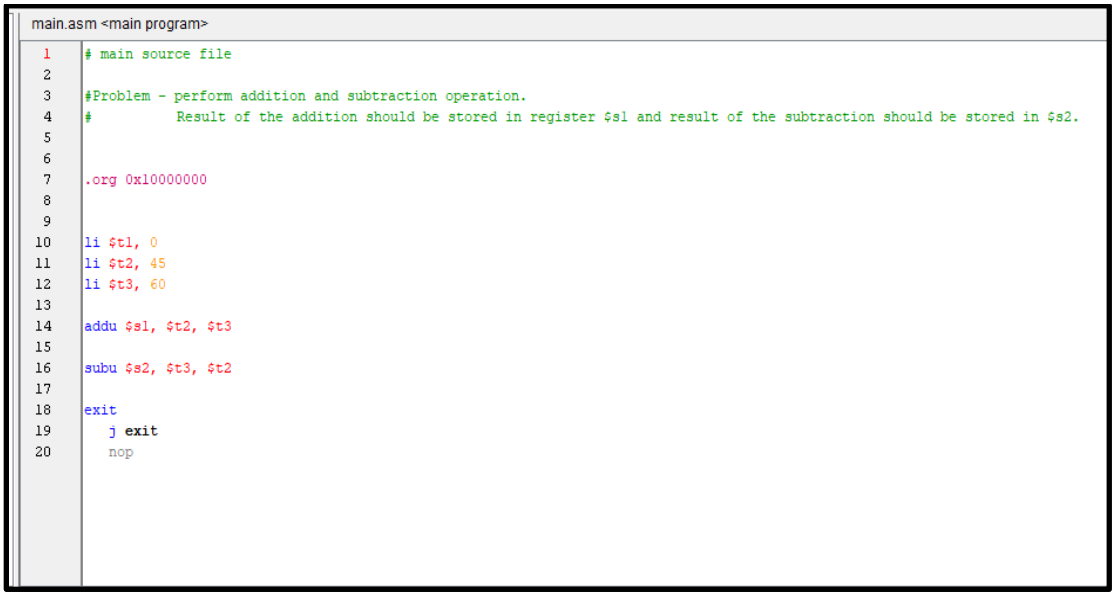

Figure 19-Program 2 label program

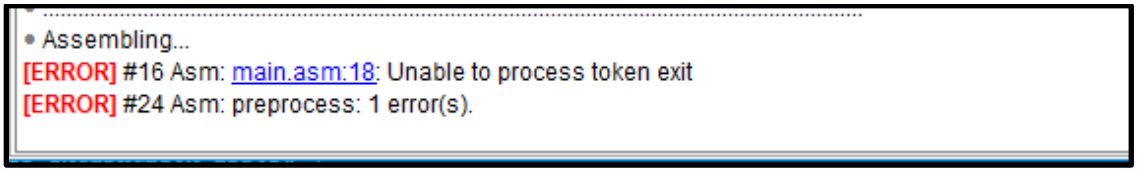

Figure 20-*Default type* Error Message for the Program 2 Label Program

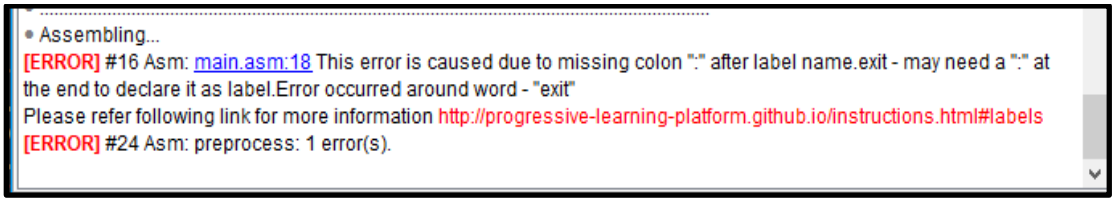

Figure 21-*Link type* Error Message for the Program 2 Label Program

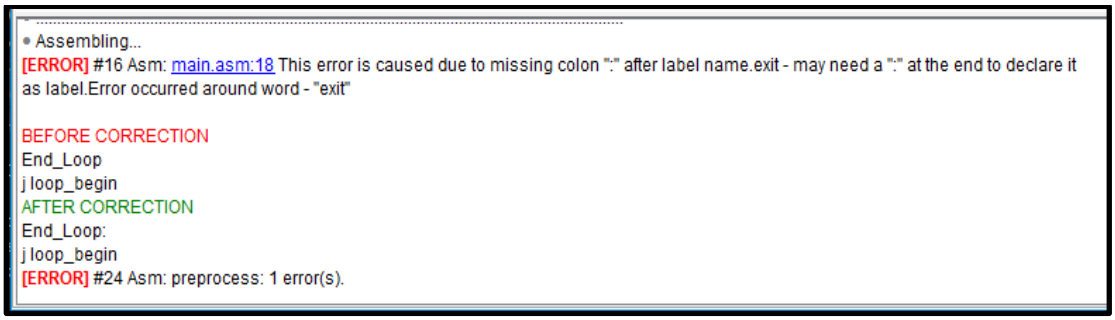

Figure 22-Example Type Error Message for the Program 2 Label Program

2. Instruction Program:

a. First Instruction program:

This third program provided to the participant was an instruction program. *Figure 23* shows the exact program provided to the participants. The program contains an invalid instruction on line 14. PLP instruction set does not contain an instruction "subiu". There is no instruction available which does the immediate value subtraction. To fix this issue there were two ways. The first one is to use "subu" with an additional register. In this method, the participant must create a new register with value 10 and use that register with "subu" instruction. Another way of solving this problem was to use "addiu" instruction with value -10. *Figure 25* shows the *Default type* error displayed for the program. *Figure 26* shows the link type error displayed for the program. *Figure 26* shows the *Example type* error displayed for the program.

main.asm <main program> Editor Mode

```
1
2   # Problem - Given a number (60), based on the value in register
    $t1, perform either increment by 10 or decrement by 10.
3   #               Final result is stored in register $s1.
4   #               0 - Increment Operation
5   #               1 - Decrement Operation
6
7   .org 0x10000000
8
9   li $t1, 1
10  li $t3, 60
11
12  beq $t1, $0, increment
13  nop
14  subiu $s1, $t3, 10
15  j exit
16  nop
17
18  increment:
19              addiu $s1, $t3, 10
20
21  exit:
22              j exit
23              nop
```

Figure 23- Program 3 Instruction Program

```
[ERROR] #16 Asm: main.asm:14: Unable to process token subiu
[ERROR] #24 Asm: preprocess: 1 error(s).
```

Figure 24- *Default type* Error Message for the Program 3 Instruction Program

```
[ERROR] #16 Asm: main.asm:14 This error is caused due to wrong instruction name. PLP does not
have an instruction as mentioned in the code.subiu - instruction is not defined in PLP. Unable to
process the instruction.Error occurred around word - "subiu"
Please refer following link for more information http://progressive-learning-platform.github.
io/instructions.html#operations
[ERROR] #24 Asm: preprocess: 1 error(s).
```

Figure 25- Link Type Error Message for the Program 3 Instruction Program

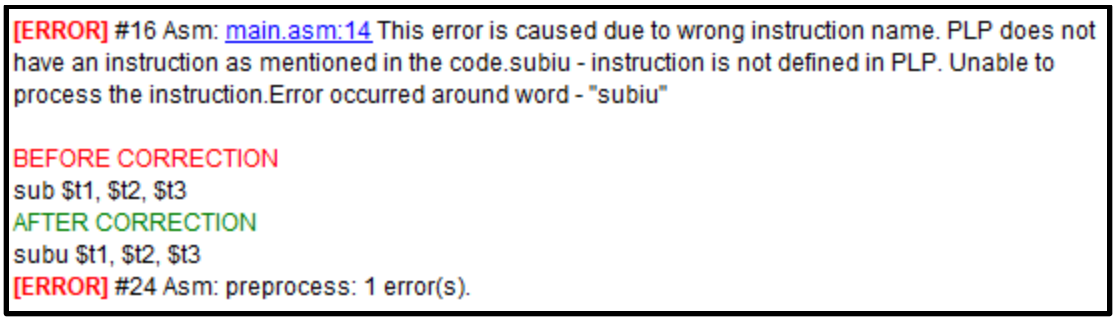

Figure 26- Example Type Error Message for the Program 3 Instruction Program

b. Second Instruction program:

This was the fourth program given to participants during the experiment. *Figure 27* gives the Instruction program. This program has an error on line 22. Here program's intention is to increment value 10 from the register $t3 and store back the result into register $s1. For unsigned operations, we do have an addition instruction in PLP which is "addiu". So "addui" is not an instruction in PLP. *Figure 28* gives the *Default type* error message displayed for the program. *Figure 29* gives the *Link type* error message displayed for the program and *Figure 30* gives the error message displayed for *Example type* error message.

main.asm:26 <main program> Editor Mode

```
1   # main source file
2
3   # Problem -  based on the value in register $t1, perform either
    increment by 10 or multiply by 10.
4   #              Final result is stored in register $s1.
5   #              0 - Increment Operation
6   #              1 - Multiply Operation
7
8   .org 0x10000000
9
10
11  li $t1, 0
12  li $t3, 60
13  li $t4, 10
14
15  beq $t1, $0, increment
16  nop
17  mullo $s1, $t3, $t4
18  j exit
19  nop
20
21  increment:
22            addui $s1, $t3, 10
23
24  exit:
25            j exit
26            nop
```

Figure 27-Program 4 Instruction Program

```
• Assembling...
[ERROR] #16 Asm: main.asm:22: Unable to process token addui
[ERROR] #24 Asm: preprocess: 1 error(s).
```

Figure 28-*Default type* Error Message for the Program 4 Instruction Program

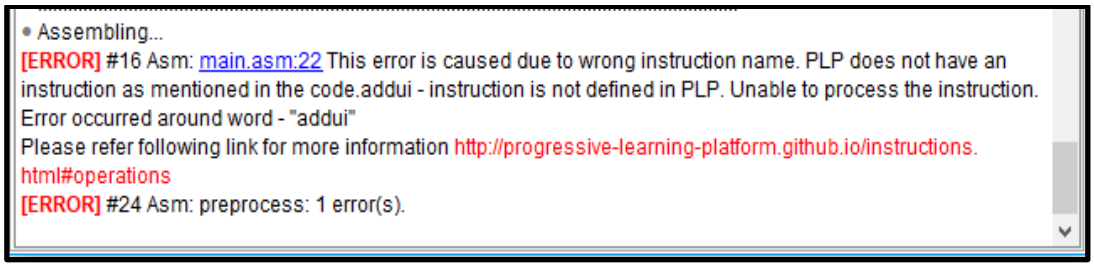

Figure 29-Link Type Error Message for the Program 4 Instruction Program

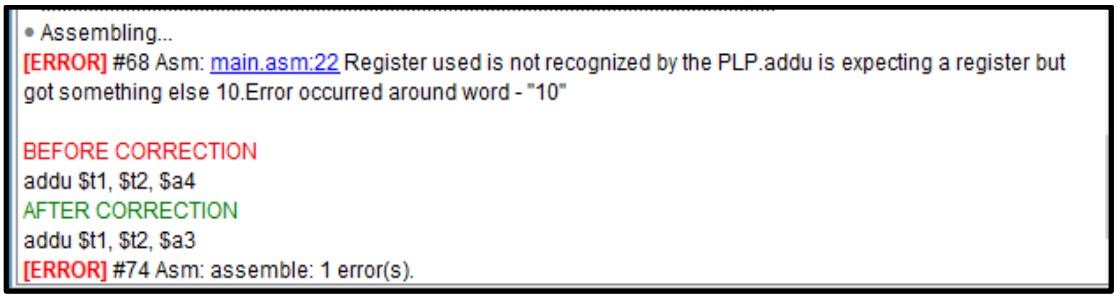

Figure 30-Example Type Error Message for the Program 4 Instruction Program

3. Register Program:

a. First register program:

Figure 31 shows the fifth program given to the participants. The intent of the program is to perform addition, subtraction, and multiplication on two registers $a0 and $a1. The values stored in the register $a0 is 100 and the value stored in the register $a1 is 200. The error is present on line 28. The register used in the program $v2 is not present in the PLP register set. To correct this error, it is very important to have good knowledge about the register sets present in the PLP assembly language. The solution to this error is the use of any valid register for example from register set s or v. In this program if v type register is used then it will overwrite the previous result which makes the program logically incorrect.

The *Default type* error message for this program is given in the *Figure 32*. The *Link type* error provided to the participants is given in the *Figure 33* and *Example type* error message provided to the participants is given in the *Figure 34*.

main.asm <main program> Editor Mode

```
1
2   #This program will perform addition, subtraction and multiplication
    operation on two registers $a0 and $a1.
3   .org 0x10000000
4   main:
5               li $a0, 100
6               li $a1, 200
7               jal addition_operation
8               nop
9               jal subtraction_operation
10              nop
11              jal multiplication_operation
12              nop
13  end:
14              j end
15              nop
16
17  addition_operation:
18              addu $v0, $a1, $a0
19              jr $ra
20              nop
21
22  subtraction_operation:
23              subu $v1, $a1, $a0
24              jr $ra
25              nop
26
27  multiplication_operation:
28              mullo $v2, $a1, $a0
29              jr $ra
30              nop
31
32
```

Figure 31- Program 5 Register Program

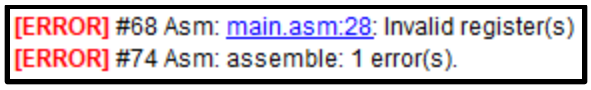

Figure 32- *Default type* Error Message for the Program 5 Register Program

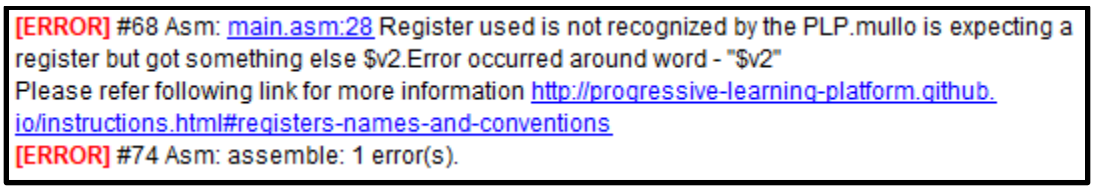

Figure 33- Link Type Error Message for the Program 5 Register Program

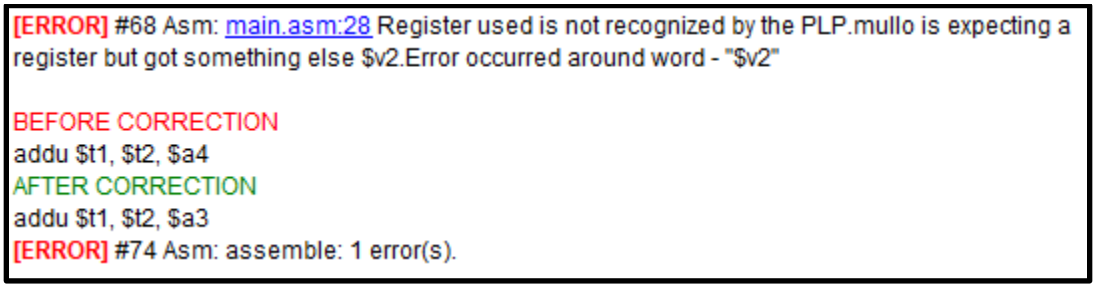

Figure 34- Example Type Error Message for the Program 5 Register Program

b. Second register program:

The sixth and last program provided to the participant is given in the *Figure 35*. The intent of the program is to turn on the LED at the least significant bit position and then the LED at the most significant bit position in the LED array. The error is present on line 23 and 24 in this program. The program is trying to use register $s8 which is not present in the register set in PLP. To correct this error, it is important to have a good understanding of the registers present in the register set. After changing $s8 with valid saved temporaries register the error disappears from the program. *Figure 36* shows the *Default type* error presented to the students. *Figure 37* shows the *Link type* error presents to the students. *Figure 38* shows *Example type* error given to the students.

main.asm <main program> Editor Mode

```
1
2   #This program will first turn on the LED at the least significant bit
    position, and then the LED at the most significant bit position in the LED
    array.
3
4   .org 0x10000000
5   main:
6           li $t1, 0xf0200000
7           jal lsb_LED
8           nop
9           jal msb_LED
10          nop
11
12  end:
13          j end
14          nop
15
16  lsb_LED:
17          li $s7, 1
18          sw $s7, 0($t1)
19          jr $ra
20          nop
21
22  msb_LED:
23          li $s8, 128
24          sw $s8, 0($t1)
25          jr $ra
26          nop
27
```

Figure 35-Program 6 register program

```
• Assembling...
[ERROR] #68 Asm: main.asm:23: Invalid register(s)
[ERROR] #68 Asm: main.asm:23: Invalid register(s)
[ERROR] #68 Asm: main.asm:24: Invalid register(s)
[ERROR] #74 Asm: assemble: 3 error(s).
```

Figure 36-*Default type* Error Message for the Program 6 Register Program

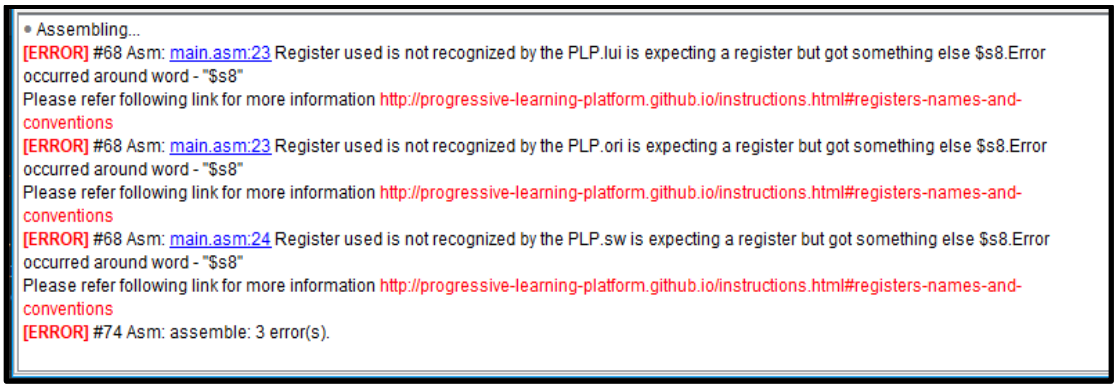

Figure 37-Link Type Error Message for the Program 6 Register Program

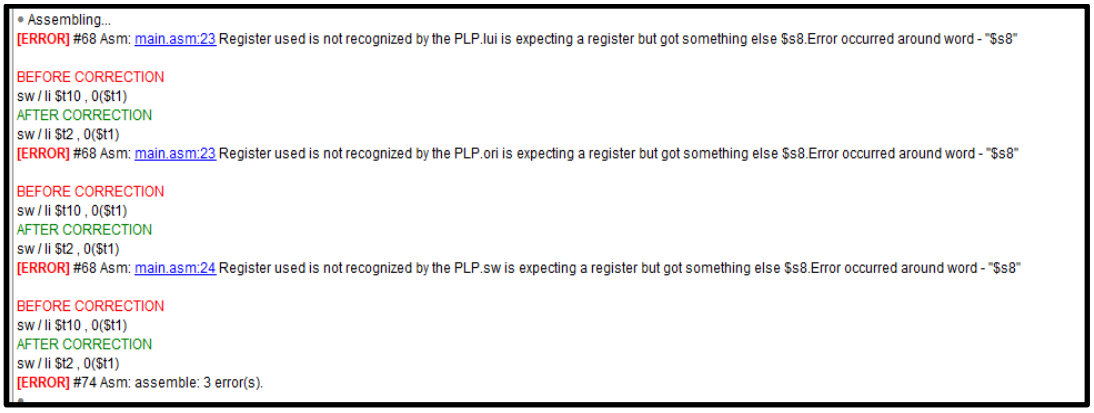

Figure 38-Example Type Error Message for the Program 6 Register Program

APPENDIX B

DEMOGRAPHIC SURVEY FORM

In the demographic survey, following questions were asked:

1. Please select your proficiency in the PLP assembly language.

   0 - No proficiency

   1 - Elementary proficiency

   2 - Limited working proficiency

   3 - Full professional proficiency

2. Please select your proficiency in programming language other than assembly. [Example – High level language like C, C++, Java, Python]

   0 - No proficiency

   1 - Elementary proficiency

   2 - Limited working proficiency

   3 - Full professional proficiency

3. Please select your proficiency in using an Integrated Development Environment (IDE).

   0 - No proficiency

   1 - Elementary proficiency

   2 - Limited working proficiency

   3 - Full professional proficiency

4. Please select the assembly language you have used other than PLP

   none

   MIPS

   Other (Example – x86/Motorola/ARM)

For this question if user selects MIPS then following question were asked:

5. Please select your proficiency in MIPS assembly language

   0 - No proficiency

   1 - Elementary proficiency

   2 - Limited working proficiency

   3 - Full professional proficiency

APPENDIX C

SAMPLE TRANSCRIBING, SEGMENTING AND CODING

The table 17 shows sample coding of student 11 for the *Default type* error message. The steps were coded as Examine, Explore, Hyporhesize, Repair and Evaluate.

Speaker	Audio	Screen	Examine Explore Hypothesize Repair Evaluate
##	so this problem is asking me to perform addition and subtraction operation so the result of addition should be stored in s1 then result of subtraction should be stored in s2		Examine
##	okay I have three load immediate operations the result being stored in t1 and 45 in t2 60 in t3		Examine
##	I do not see any error messages right now I am going to try to run it see what happens.		Examine
##		Clicked on Assemble button.	
##		asm:18 Unable to process token exit	
##	so error message it say main 18 unable to processes token exit.		Examine
##	okay I see the problem the exit statement doesn't have semicolon.		Examine
##	so I am gonna add semicolon and make sure everything else look okay...okay it should work	Added semicolon at end of exit label	Repair
##	yes result addition stored in s1 and result subtraction stored s2 so perform exit.		Examine
##		Clicked on Simulate button. No error reported.	
##	so I simulate it and there are no problems. I think I am done.		Evaluate

Table 17- Sample Coding for Basic Steps

Speaker	Audio	Screen	Expected Gaming T&E Incorrect-Interpretation Incorrect-Independent Correct-Independent Silly Mistake
##	Okay based on register t1 increment by 10 or decrement by 10 so adding or subtracting 10		Expected
##	currently it is set to decrement umm let's see so equals if t1 is zero increment otherwise takes immediate subtract 10 exit nop exit works		Expected
##	extra exit here	Deleted j exit loop statement	Incorrect Interpretation
##	increment has nothing wrong here		Expected
##		Clicked on Assemble button	Expected
##	error caused due to wrong instruction name instruction not on after correction	Received error of type example	Expected
##	except the problem with this subi cause subu is not what we are looking for either		Incorrect Interpretation
##		Changed subiu to subi	T&E
##		Received *Example type* error	
##		Changed subi to subu	Gaming
##	register is not recognized subu is expected	Received error of type example	Incorrect Interpretation
##		Opened the online PLP manual	
##	what is it there is not sub immediate in PLP		Correct Independent
##	alright I will just use add immediate	Changed to addiu from sub	Correct Independent
##	okay it assembles.		

Table 18- Sample Coding for Expected and Unwanted Steps

Speaker	Audio	Screen	Program-Description Error-Message Code Search-Information
##	So problem is to either do addition or subtraction final result should be stored 0 1 addition subtraction		Program Description
##	so load 0 45 60 branch equal 1 0 so its saying 1 is equal 0 it will go to addition branch		
##	okay so first branch is acting as if statement for addition or subtraction depending on what loaded in t1 determines addition or subtraction subu so it gets stored in s1 t3 minus t2 jump exit addition		
##	umm I don't think there is an error		
##		Clicked on assemble button.	
##	Let's try to caused due to missing colon after addition	Received Link Type error	Error Message
##		Added colon after addition label	Code
##	oh addition didn't have colon so it wasn't reading it as a label	Clicked on assemble button.	
	umm okay let's see alright so I have completed the first one		

Table 19- Sample Coding for Examine Steps

Speaker	Audio	Screen	Complete Partial Ignore
##	Based on value in register increment by 10 or multiply by 10		
##	this time branch equals increment nop increments same as before multiplying t1 t3 t4		
##	does not have oh its typo	Received *Example type* error.	Ignored
##	doesn't look like any more assembly errors	Changed addui to addiu	
##	oops oh wrong one		
##		Clicked on assemble button. Clicked on step button.	
##	start its going infinite loop	Removed j exit statement	
##	then run	Clicked on simulation and then run button	

Table 20- Sample Coding for Complete, Partial, Ignore

APPENDIX D

VERBAL, BLACKBOARD AND SLACK CHANNEL ANNOUNCEMENT SCRIPT

My name is Siddhant Tanpure and I'm currently undertaking a research study to investigate the effects of error messages in PLP (Progressive Learning Platform) language on student's ability to understand and fix errors in programs. The total amount of time you would participate for is approximately 1 hour.

Your participation will involve demographic survey; think-aloud programming exercise consisting of six PLP (Progressive Learning Platform) programs and feedback questionnaires about the PLP programs. You have the right not to answer any question, and to stop participation at any time. The computer monitor will be recorded (commonly referred to as screen recording). Entire session will be audio recorded. The recordings will be stored in secure password protected ASU dropbox with access only to research team. The recordings will be kept for a period of 5 years.

Your participation in this study is voluntary. If you choose not to participate or to withdraw from the study at any time, there will be no penalty. Your participation in this study will not affect your grade in CSE 230. You must be 18 or older to participate in the study.

Your participation in this study will in no way affect your grade in class [CSE 230]. Data collected during the study will be anonymous and confidential. Results will only be shared in the aggregate form. The results of this study may be used in reports, presentations, or publications but your name or other identifiable information will not be used.

If you are interested in participating in the study, you may email me at sbtanpur@asu.edu.

APPENDIX E

CONSENT AND PARTICIPANT RECRUITMENT FORM

Increasing the effectiveness of error messages in a computer programming and simulation tool

Welcome to the research study!

I am a graduate student under the direction of Dr. Sohum Sohoni in the Department of Engineering at Arizona State University. I am conducting a research study to examine the effects different forms of error messages in PLP (Progressive Learning Platform) language on student's ability to understand and fix errors in programs.

I am inviting your participation, which will involve demographic survey, think-aloud programming exercise consisting of six PLP (Progressive Learning Platform) programs and feedback questionnaires about the six PLP programs. The study should take you around 1 hour to complete. You have the right not to answer any question, and to stop participation at any time.

Your participation in this study is voluntary. You must be 18 or older to participate in the study. If you choose not to participate or to withdraw from the study at any time, there will be no penalty. Your participation in this study will not affect your grade in CSE 230. You have the right to withdraw at any point during the study, for any reason, and without any prejudice.

You will receive $10 worth of gift cards for your participation. There are no foreseeable risks or discomforts to your participation.

Your name and other identifiable information will not be collected. Results will only be shared in the aggregate form. The results of this study may be used in reports, presentations, or publications but your name or other identifiable information will not be used.

I would like to audio record this session. Everything visible to you on the computer monitor will be recorded (commonly referred to as screen recording). The screen and audio recording will not take place without your permission. Please let me know if you do not want the screen or audio to be recorded; you also can change your mind after the recording starts, just let me know.

If you have any questions concerning the research study, please contact the research team: Dr. Sohum Sohoni at (405) 338-8317 or Sohum.Sohoni@asu.edu, Siddhant Tanpure at (480) 570-9991 or sbtanpur@asu.edu . If you have any questions about your rights as a subject/participant in this research, or if you feel you have been placed at risk, you can contact the Chair of the Human Subjects Institutional Review Board, through the ASU Office of Research Integrity and Assurance, at (480) 965-6788.

By clicking the button below, you acknowledge that your participation in the study is voluntary, you are 18 years of age, and that you are aware that you may choose to terminate your participation in the study at any time and for any reason.