Effects of Error Messages on Students' Ability to Understand and Fix Programming Errors

Harsha B. M. Kadekar
School of computing, informatics, & decision systems engineering
Arizona State University
Tempe, USA
harsha.kadekar@asu.edu

Sohum Sohoni
The Polytechnic School, Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, USA
sohum.sohoni@asu.edu

Scotty D. Craig
The Polytechnic School, Ira A. Fulton Schools of Engineering
Arizona State University
Tempe, USA
scotty.craig@asu.edu

***Abstract*—The role of error messages in the context of teaching programming, specifically assembly language programming to students who have limited prior programming experience was investigated. Assemblers and compilers provide feedback to a programmer in the form of error messages, and these error messages influence the programmer's mental model of computing. The current study investigated how an error message affects students' approach to understanding the error and fixing the error. Three error message types were developed – Default, Link and Example, to better understand the effects of error messages. The Default type provides an assembler-centric single line error message, the Link type provides a program-centric detailed error description with a hyperlink for more information, and the Example type provides a program-centric detailed error description with a relevant example. A think aloud programming exercise was conducted to capture the student programmer's knowledge model. Different codes were developed to analyze the data collected as part of think aloud exercise. After transcribing, coding, and analyzing the data, it was found that the Link type of error message helped to fix the error in less time and with fewer steps. Among the three types, the Link type of error message also resulted in a higher ratio of correct to incorrect hypotheses made by the participants, and a correspondingly higher ratio of correct to incorrect steps taken by them to fix the error.**

Keywords—component, formatting, style, styling, insert (*key words****)***

I. INTRODUCTION

A. Motivation

Assemblers and compilers act as feedback mechanisms which helps student programmers to test their mental models of programming [1]. Each time they use the assembler/compiler, it will evolve those mental models. However, assembler and compiler error messages are typically formed from the perspective of the programmer designing the compiler or assembler, rather than the perspective of the user of these tools [1]. Among the student programmers, novice programmers are the ones who are most affected by these error messages [2].

There are three steps in the process of fixing an error. First would be to identify the location of the error in the program. The next step is to understand the cause of the error based on the error message provided. The final step is to fix the error based on the information provided by the error message and the programmer's prior knowledge. Essentially, the error message generated by the assembler/compiler should help the programmer in locating, understanding, and finally fixing the bug in the code [3].

Quite often, an error message describes at what stage of assembling or compiling an error occurred rather than describing what could have caused the error, or what mistake on the programmer's side could have caused the error. For a student programmer who does not have any background on the inner workings of the assemblers/compilers, these messages appear to be cryptic and unhelpful [4][1]. Thus, usually the novice programmer ends up making random changes, following an unguided trial and error process hoping to get rid of the error. When the cryptic error messages persist, or increase during these attempts to fix the error, the programmer's frustration increases[5]. Hence, as many novice programmers treat these errors as personal failures, they feel demotivated, and their frustration and fear of programming increases [6]. As error messages are the direct feedback for the students' learning models, cryptic error message will become a barrier in the development of those learning models. However, useful error messages will not only help in correcting the error, but will also help to build a strong mental model of the programming language in the student's mind. They will help clarify concepts, remove misconceptions, and generally result in better outcomes for programming intense courses [1]. Thus, it is important to design error messages that support rather than frustrate novice programmers.

This paper explores the impact of different types of error messages on a student's ability to understand and fix errors. The study was conducted in the context of assembly language programs and errors that occur at assembly time (i.e. not run-time or logical errors).

B. Review of Literature

Of the many studies conducted over the last few decades to understand the effects of the error messages we focus on those that look at wording of the error message and its effects on the programmer's, ability to resolve the error.

Marceau, Fisler, & Krishnamurthi [7] helps quantify the effectiveness of error messages for a novice programmer. They explain that an error message is effective if a student reads it, can understand its meaning, and can then use the information to formulate a useful course of action. Based on this theory, various

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

codes were developed to analyze how students programmed a given an error message. The programming activity was coded, and this data was used to develop a rubric. The rubric was used to develop a formula which helps to quantify the students action on the program based on the presented error message. Our study, like theirs, uses human-factors research methods to explore the effectives of error messages. "Read->Understand->Formulate" theory is also used in our study to understand the students' programming activity.

Vessey [8] used verbal protocol analysis to compare the debugging processes employed by expert and novice programmers. The study found that even though both experts and novices used breadth-first approaches for debugging, novices are deficient in their ability to think in system terms. The novice is also unable to divide programs into logical units (chunking) when compared to expert programmers. This was one of the initial experiments to employ a think aloud experiment to capture the mental model of a programmer. Our study employs a similar think aloud programming activity to better understand the mental model of student programmers.

Nienaltowski, Pedroni & Meyer [2] studied three types of error message representations – short form, visual form, and long form. Students from two different universities took part in the study which tested hypotheses like "at a lower experience level, enhanced messages result in more correct answers, more information results in more correct answers, more information in the error messages results in shorter response time". The questionnaire had three types of errors in three different forms of error messages. The results showed that providing more information in the error message does not lead to more correct answers. It also showed that giving more information in the error message did not reduce the response time. The other hypothesis which proved wrong was type of error determines the number of correct answers. Our study also used three types of error message representations for three different errors.

Hartmann, et al. [9] developed a system which gives helpful suggestions to compiler and runtime error messages. This system called as HelpMeOut, tracks the source code development from its error state to final correct state. It stores all the compiler errors committed by the users in a central database; information such as what was the error message, what was the wrong code and how that was corrected. When a user asks for help to correct an error message, it will fetch relevant data from that central database. Suggestions include a description of possible corrections, explanation of the error and previous code examples of the error and how it was fixed. This was done for 2 programming languages – Java and C++. A total of 39 hours of programming data was collected out of which 178 times suggestions were provided. Among them 47% of them were useful. In our study, one of the error message type has similar feature of displaying a relevant example code, although the example is not derived from real-time collection of errors and error corrections.

Traver [1] investigated cryptic compiler error messages to understand why error messages make the work of programmers more difficult. According to this study's findings, current error messages are more compiler-centric rather than programmer-centric. It was observed that most of the error messages were lacking in clarity and they were context insensitive. Most of the error messages were lacking in constructive guidance, and did not provide valid suggestions for correcting errors. The author also observed that many of the error messages were not specific, which resulted in different diagnostics. In our study, error messages were designed to rectify the issues listed by [1].

C. *Problem Statement*

This paper studies the effect of an error message on different aspect of resolving an error. This is an exploratory study, which tries to understand the mental model of a programmer and how an error message affects that mental model. As part of the error message construction itself, the study tries to find out what error information is helpful to programmers. To better capture all the different features that needed to be tested as part of the study, three types of error message types were developed.

1) Default – a short assembler-centric message indicating the stage at which the assembler was, when the error occurred.

2) Link – a detailed error messages with a weblink address for the section of the online manual where more information can be obtained.

3) *Example* – a detailed error messages with sample code. The sample code shows an error similar to the one currently encountered by the participant along with the solution for that sample error.

D. *Research Questions*

By analyzing the programmer's interaction with the three types of error messages and their corresponding programs, this study tries to answer following two research questions.

1) What aspects of an error message help the programmer to understand the error?
2) What aspects of an error message help the programmer to fix the error?

II. IMPLEMENTATION

The programming language used for the study is an assembly language called the Progressive Learning Platform Instruction Set Architecture (PLPISA). The progressive learning platform (PLP) uses simulation and visualization to teach students how software and hardware interact [10][11]. The PLP suite consists of the Verilog description of the PLP CPU, which can be synthesized on an FPGA board such as the Nexys 3 from Digilent Corp (including all interfaces to the Nexys 3 peripherals), the PLPTool IDE [12], an online manual [13], and the open-source repositories for the hardware description and the IDE software[14].

PLPTool and the assembler present in that toolchain was modified as part of this study. A separate module for processing the assembler exceptions was created. Once the assembler discovers that an error exists in the code, this module is called instead of raising the exception.

The overall assembling process of the PLP language was studied, and based on its working and PLP language structure, errors were grouped into 4 types.

1) *Invalid Label* – This error has two sub groups: Duplicate Label, when the same label is used in two different contexts, and Invalid Target, when the program uses a label name which is not yet defined.

2) *Invalid Token* – This has two sub groups: Invalid Instruction Type, when there is a spelling mistake in an instruction keyword, and Invalid Label, when the label is not declared.

3) *Invalid Number of Tokens* – This error is caused due to an invalid number of arguments to the instruction. It has two sub groups: Missing Tokens (fewer than expected, and Extra Tokens (more than expected).

4) *Invalid Operand* – This error is caused due to invalid operand to the instruction. It has four sub groups: Not Register, Not Number, Not String and Invalid address.

Once this categorization of errors was completed, an error information repository was created. For each type of error, the following information was stored in the repository.

1) Description – A detailed program centric explanation of the error.

2) Link – An http link to a section of the online PLP manual, where the programmer can get more useful information for understanding and fixing the error.

3) Example – A code sample that has two sub sections - before correction and after correction.

During the experiment, once and error in assembly occurs, the type of error will be identified and error information will be retrieved from the repository. Based on the type of error message to be displayed, some information will be discarded, and the rest will be formatted and displayed in the console window of PLPTool. For example, if the Link type of error message is to be displayed, then the example code part will be discarded. If the Example type of error message is to be displayed, then link will be discarded. If the Default type is chosen, then all the above information will be discarded and only an assembler-centric, single sentence error message will be provided.

Every error message will have an error number and error location link, which upon on clicking will highlight the line where the error occurred. The link will have the line number and file name where an error has occurred. The idea of error message with code examples was adopted from Hartmann, MacDougall, Brandt. & Klemmer [9], where they had developed a centralized repository which collects all the errors committed by the programmers and the corresponding fixes done to those errors by the same programmers.

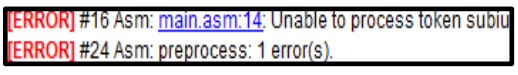

Fig. 1. The Default Type of Error Message

Fig. 1 shows an example of the Default type of error message. The instruction which is causing the error is *subiu $s1, $t3, 10*. For a novice programmer, who does not know the inner working of an assembler or compiler, the word "token" might be an unfamiliar word. As the error message says, "unable to process token "subiu.", a novice programmer might infer that the way instruction and its operands is written might be wrong. The actual reason for the error is that PLP does not have any instruction by name "subiu." Instead in PLP, subtract operation can be performed by a subu instruction. Fig. 2 shows the Link type of error message and Fig. 3 shows the Example type of error message.

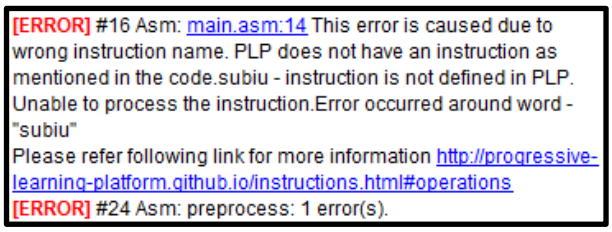

Fig. 2. The Link Type of Error Message

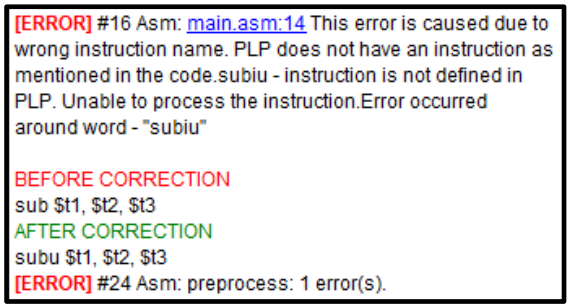

Fig. 3. The Example Type of Error Message

III. METHODS

A. Design

The current study implemented a block randomized repeated measures design in which learners received three problems with errors to correct. Each problem was randomly assigned a different type of error message. While all problems were successfully solved by the learners, a think aloud procedure and screen capture of problem solving were collected to understand the relationship between the error message and the problem solving. Seven steps of verbal data coding [15] were implemented.

B. Participants

PLPTool is used in Advanced Computer Architecture and Microcomputer Architecture and Programming courses at a large public university in the US. Any students who were enrolled in these two courses or had previously taken them were eligible to take part in the study. This study had 12 participants. Of the 12 participants, six participants were undergraduate students and six participants were graduate students.

While the number of participants is small, the number of subjects utilized in software engineering studies using think-aloud procedures tends to be small, with fewer than 30 protocols being collected. Usually a sub-set of these are used in the next stages of the research, often with 10 or fewer being prepared for encoding [16].

C. Materials

For each participant, three programs were provided with one error in each program. Each program had the program description in comments at the beginning of the program. The program description summarized the objective of the program. Following are the three programs provided to participant. Each participant faced these programs in the same order.

1) *Label Program:* This program had an invalid label declaration error.

2) *Instruction Program:* This program had an invalid instruction error.

3) *Register Program: This program had an invalid register error.*

These error types were chosen based on observations we had in an undergraduate microprocessors class. Based on discussion with other TAs, it was decided that above types of assembler errors were most common among the students. This study collected three types of data from participants.

Demographic Survey - As part of this survey, questions were asked to understand the proficiency of participants regarding PLP, assembly language, high level programming language and usage of Integrated Development Environment (IDE). Each question was given three answer choices - novice, intermediate and expert.

Think Aloud Programming Activity - This study aims to understand the knowledge structures student programmers use to solve an error and how those knowledge structures are affected by the error message, which was the input given to the students. The study is more interested in the process followed by the participants to solve the error rather than final result of whether error was fixed or not. As Vessey [8] mentions, verbal protocol or think aloud protocol is the preferred method of examining problem-solving process. Think aloud programming activities are used in many prior works to understand the mental model of the programmers [6][17][18]. In this study, 10 minutes time was given for the participant to understand the error and fix it for each of the three programs. The PLP program was opened in PLPTool, the IDE used in this study. After 10 minutes, an online feedback questionnaire was provided. This process was repeated for each of the three PLP programs.

Feedback Questionnaire - After each programming exercise completion, a feedback questionnaire related to the role of the error message in the programming activity was provided to the participants. The following questions were asked.

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?

These three questions were asked after completion of each PLP program. Participants were given 5 minutes minimum to complete this questionnaire, but they were free to take more than 5 minutes to fill out the questionnaire. The questionnaire was provided as an online form. The main reason for questionnaire was to capture those points from the subjects which were not verbalized during the programming activity itself. This provides another medium for the participants to express what helped to build their thought process. This is useful for the participants who did not verbalize much during the think aloud exercise.

D. Procedures

The participant were recruited from Advanced Computer Architecture and Microcomputer Architecture and Programming courses. Data was collected one participant at a time in a lab setting. After providing consent, participants took the demographic survey, followed by the think aloud programming activity, concluding with the feedback questionnaire. The whole process took approximately one hour to complete for one participant.

E. Data Capturing Procedures

All the participants' recordings were transcribed. Transcription involved both verbal utterances as well as screen activity, like changes in the program, searching in the online manual. Transcribing was done in multiple iterations. In the first pass, complete screen recording of that program was viewed along with the audio. In the second pass, just the audio of the recording was transcribed. Finally, in the third pass, the corresponding screen activities were transcribed and linked with the transcript of the audio recording. If any words were not audible in the recording, they were transcribed as [inaudible words]. If there were a few seconds of silence, then it was transcribed as '…'.Similarly, if there was a long pause, it was mentioned in the transcript as [long pause].

A coding scheme was developed inductively to identify basic steps for problem solving (Table I), and codes for correct and incorrect steps (Table II).

TABLE I. BASIC STEPS CODE

Code	Definition	Examples
Examine	Action involving reading program description, code, error message and reading searched information	"addition on v0 subtraction on v1 multiplication on v2"
Explore	Actions involving searching in internet, PLP Online manual, PLP Quick reference guide and error location link	"Goes to online manual. In that opens Register names and conventions section."
Hypothesize	A possible solution or an explanation given by the participant	"hmmm… so addiu is for sign extended addition but there is no I am not able to find any command for subtraction so I think we should use subu"
Repair	Doing the code changes in the program	"Line 14 is changed. Replaces subiu with subu. Now line 14 has instruction subu $s1, $t3, 10"
Evalute	An action taken to test the repair or hypothesis.	Assembles the program by clicking assemble button. Gets one error. [ERROR] #68 Asm: main.asm: 14 Invalid Register(s)

TABLE II. CORRECT AND INCORRECT STEPS CODE

Code	Definition
Expected	Action taken by the participant was expected by us
Gaming	Error was fixed using the error message, but participant did not understand the error or solution
T&E	Trial and Error
Correct Independent	A correct action was taken independent of the error message
Incorrect Interpretation	Incorrect action was taken based on the error message
Incorrect Independent	An incorrect action was taken independent of the error message
Silly Mistake	Interpretation was correct but made a mistake while fixing it

IV. RESULTS

A. Cohen's Kappa Coefficient for Inter-Rater Agreement

Once the coding was done, for the purpose of measuring interrater reliability, 3 participants' data was chosen randomly which covered all the three scenarios presented in Table I. Two raters coded this set independently. After coding Cohen's Kappa was calculated on the coded data to measure interrater reliability. For the basic steps coding that is for marking of 'Examine/Explore/Hypothesize/Repair/Evaluate', we got a score of 0.8736 which indicates near perfect agreement. For the expected and unwanted steps, we got a score of 0.62 which indicates moderate agreement. The remainder of the results section presents data from all 12 participants.

B. Time to Resolve the Error

The time to resolve the error is the time from when the error message was received to when the error was fixed. Usually, even after the error was resolved, it was observed that a participant would perform other activities like simulating (running) the program. Time spent on those activities is not considered. Table III provides the average time it took for the participants to fix an error for the different error messages types.

TABLE III. TIME TAKEN TO RESOLVE THE ERROR

	Average (seconds)	SD
Default	184.34	148.77
Link	136.67	74.05
Example	200.25	101.62

As can be seen from the table, programs which are given the Link type of error message took less time when compared to the other types of error message. Not only did the Link type take less time, the time was also more consistent when compared to other types of error message. To our surprise, the Example type took more time to fix than the Default type.

C. Number of Steps Taken to Resolve the Error

For every program, we measured the number of steps taken to fix the error. In every program, we had segmented into different steps or activities where each step is coded as one among Examine/Explore/Hypothesis/Repair/Evaluate. Table IV gives the statistics related to steps taken to fix the error.

The same pattern appears in Table IV as was observed in Table III. The Link type not only takes participants less time to fix but also takes fewer steps when compared to the other two types of error messages. Again, the Link type is more consistent. The Example type is takes more steps to fix when compared to the other two.

D. Correct and Incorrect Steps

The Explore-Hypothesize-Repair steps were categorized as correct steps or incorrect steps. Here correct steps include those which are coded as Expected or Correct Independent and incorrect steps include those which are coded as Silly Mistake, Trial & Error, Gaming, Incorrect Interpretation and Incorrect Independent. Table V provides how many of these steps (Explore-Hypothesize-Repair) were correct and incorrect steps for each error message type.

TABLE IV. NUMBER OF STEPS TAKEN TO RESOLVE THE ERROR

	AVERAGE	SD
Default	14	11.3
Link	9.8	4.17
Example	15.83	10.77

TABLE V. CORRECT AND INCORRECT STEPS

	CORRECT STEPS (%)	INCORRECT STEPS (%)
Default	59.34% (54)	40.65% (37)
Link	89.83% (53)	10.17% (6)
Example	71.56% (73)	28.43% (29)

Based on Table V, programs with the Link type of error message have fewer incorrect steps. Thus, the Link type error message results in less time to fix, fewer steps to fix and among the steps taken, a higher percentage of correct steps.

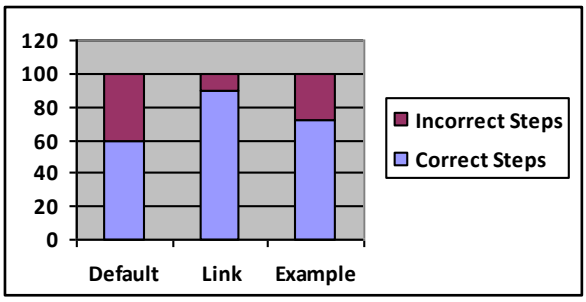

Fig. 4. Correct and Incorrect Steps

E. Correct or Incorrect Understanding of Error Message

After participants read the error message, they hypothesize the solution or express the error in their own words. If participants can understand the error message, then they hypothesize the correct solution or clearly express the error in their own words. Table VI provides the percentage of correct and incorrect hypotheses immediately after reading an error message.

V. Discussion

A. Descriptive Analysis

While the limited sample size prevented any meaningful quantitative group differences to be analyzed, the Link type of error message system provided decreased errors (Cohen's d = .41) and steps to resolve errors (Cohen's d = .49) compared to default. Because quantitative statistics could not be performed, the effect size was provided so that the magnitude of the reported differences could be observed. It should be noted that these effect sizes are on the low end of medium. The magnitude of the effect provide evidence for the plausibility of an effect and provide estimates for determining larger sample sizes in future studies. The explanation based error feedback also performed better than the default messaging condition with fewer incorrect steps taken. This is promising for the Link and Example error messages and indicated that there is a potential for improve performance in the system. However, follow-up evaluations are still needed to determine if these are real differences.

Feedback that is low levels of specificity is hurtful to the learning process [19][20][21]. This can be seen in the standard error message which can be seen in Fig. 1, This message gives a specific, but very archaic message that does not provide directive instructions. This makes the Default message very vague. This put the overall findings for the study in line with the literature showing the negative effects of error feedback.

The Link error message (See Fig. 2) is an example of directive feedback. This type of feedback indicates what needs to be fixed and gives specific information on how to fix the error [22]. This type of formative feedback facilitates the learning process[23]. The Link error message indicates that there is an error and links directly to information on how to correct the problem.

TABLE VI. CORRECT OR INCORRECT UNDERSTANDING OF ERROR MESSAGE

	Correct Hypothesis (%)	*Incorrect Hypothesis (%)*
Default	12 (66.67%)	6 (33.33%)
Link	20 (90.90%)	2 (9.1%)
Example	18 (78.26%)	5 (21.74%)

The other type of formative feedback is facilitative [22]. Facilitative feedback provides comments, examples and suggestions to help guide the revision of the error [23]. The Example error message (See Fig. 3) fits this type of feedback in that it shows an example of the error and a correct solution. This requires the learner to compare the two to find the discrepancy and see the relationship between the example and the error.

Humans are not normally strong in generalizing from an example to a specific problem even when these problems are highly analogous [24]. Having to make these comparisons may have increased uncertainty. During problem solving, uncertainty distracts attention from task performance and decrease efficiency in task strategies [25]. This is the exact pattern seen in learner's behaviors with the link conditions having fewer steps overall, fewer incorrect steps and less time when compared to the explanation error condition. It might also explain why the explanation condition has similar levels to the default error message condition overall. However, the guidance provided the example effort feedback does appear to have decreased the number of incorrect steps make while problem solving as would be expected from formative feedback [23].

However, uncertainty, and more specifically confusion, during the learning process is not always negative and has been linked to increased learning [26][27]. The struggle exhibited during the learning process can help to build a stronger understanding on the content because it causes the information to be processed at a deeper level and more thoroughly [28]. This is known as productive confusion which is the process of struggling through confusion to the point of resolving the confusion [29][30]. In the current study, all Example condition problems were solved. So, when confusion was introduced it was eventually resolved as indicated by the correct solution being reached. This would argue that this condition could provide deeper understanding of the material and would show up in more complete mental models of the problem space. However, this still needs to be verified in future studies.

B. Reflection on Experiment

All the participants could fix errors in all the programs given to them. Though during the process some of them committed other errors due to incorrect changes to the program. Once recording had been stopped i.e. once experiment was over, participants tended to informally discuss the programs, the errors they encountered and error messaging types. They gave valuable feedback during that time. A trial run of the experiment was conducted before the recruitment advertisement for participation was made public. This helped to identify few gaps in the experiment setup like environment setup for the experiment which helped to conduct experiment with other participants smoothly.

A few valuable lessons were learned and some just-in-time fixes were applied during the main experiment. In the demographic survey, most of the participants chose PLP proficiency as intermediate. It would have been better to have a 4-scale or even a number ranking scale as options for the questions in demographic survey. During the think aloud, many of the participants would become silent and get completely involved in solving the error mentally. In that scenario, they were prompted to "Keep talking", so that they get back to the think out aloud setup. For the feedback questionnaire, as the questions were phrased as "How did the error message help in understanding/fixing the error?", participants tended to only give positive feedback about the error message. It would have been better to tell the participants to provide both positives and negatives about the error message. Most of the participants finished the feedback within 2 minutes by writing just one or two

lines. Participants had to be reminded that until minimum 5 minutes is over, they could go on to next segment. Participants were reminded to use that extra time to rephrase and add new information to the feedback.

C. Future Work

There is still lot of data from this experiment which needs to be further analyzed. One interesting aspect to look for is the confidence with which a participant makes the changes to the code. We need to understand what factors help to increase the confidence of a programmer. To understand this, we need to look at the data as a set of impasses and their resolutions. We should also further analyze the incorrect steps taken by the participants. Analysis on what type of incorrect steps are taken more commonly, and once an incorrect step is taken, how the participants correct it. This could help in understanding features of the programming language and teaching techniques that negatively impact student programmer's mental model. There is also need to analyze the flow of steps taken by the participants – is there any particular pattern in debugging? We can also understand what part of the program is examined more and is there a pattern in that as well.

The effects of error message types on novice programmers needs to be investigated. In the current experiment, participants were selected in general without any PLP proficiency level precondition. It is difficult to analyze how these error messages would affect the novice programmer's performance. So, in a future experiment we need to have a participant pool with equal representation of novice and expert programmers. This will help to clearly understand what factors of error messages help novices and expert and find out whether it differs. As every participant in this experiment could solve the programs, we need to come up with better programs which can test more factors of the error message.

As for the tool and error message system itself, the current experiment is just a preliminary step towards the development of an intelligent IDE. These IDE's should act as coding assistant [31][32] which improve the performance of the programmers by reducing the unnecessary steps taken by programmers.

VI. Conclusion

This exploratory study used a think aloud programming exercise to understand the effects of an error message on the overall debugging process of student programmers. As part of this study, three different error message types were developed. The effect of each of these error message types was analyzed from the perspective of understanding of an error and resolving the error. In the study, it was observed that an error message type which describes the error in a program-centric way rather than an assembler-centric way helped increase understanding of the error. An error message type which has a hyperlink to a relevant section in an online manual, appeared to help in fixing the error by reducing the incorrect interpretation of the error and increasing the understanding of the error.

The Link type error message was associated with participant's ability to fix the error in less time and with fewer steps. It also assisted the student programmers' search for information in the online manual and quick reference in a more focused manner. On the other hand, the study showed that giving a relevant example code as part of the error message may lead to confusion among the student programmers. The error message which provides little description, or an assembler-centric error description also creates confusion. The confusion and lack of information in the error messages led to more steps and more time to resolve the error. This lack of clarity in the error message led to less understanding of the error which in turn led to more incorrect steps in resolving the error. While additional research with a larger sample size is needed to verify these findings, the Link type error message which has a program-centric enhanced error description and a hyperlink to relevant online manual content, appears to provide improved problem solving behavior.

References

[1] V. J. Traver, "On compiler error messages: what they say and what they mean.", Advances in Human-Computer Interaction, 2010.

[2] H. Nienaltowski, M. Pedroni and B. Meyer, Compiler error messages: What can help novices? In ACM SIGCSE Bulletin, Vo 40, No. 1, March 2008, pp 168-172.

[3] W. Lazonder, and H. van der Meij, "Error-information in tutorial documentation: supporting users' errors to facilitate initial skill learning", International Journal of Human-Computer Studies, 42(2), 1995, pp.185-206.

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

[5] M. M. T. Rodrigo, and R. S. Baker, "Coarse-grained detection of student frustration in an introductory programming course.", In Proceedings of the fifth international workshop on Computing education research workshop, ACM, August 2009, pp. 75-80.

[6] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murphy, B. Simon, L. Thomas, and C. Zander, "Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers." Computer Science Education, 18(2), 2008, pp.93-116.

[7] G Marceau, K. Fisler, and S. Krishnamurthi, "Measuring the effectiveness of error messages designed for novice programmers.," in Proceedings of the 42nd ACM technical symposium on Computer science education, March 2011 , pp. 499–504.

[8] I, Vessey. "Expertise in debugging computer programs: A process analysis." International Journal of Man-Machine Studies 23, no. 5, 1985, pp. 459-494.

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

[10] S. Sohoni, D. Fritz, and W. Mulia, "Transforming a Microprocessors Course through the Progressive Learning Platform." In ASEE Midwest Conference, September 2011, Russellville, AR.

[11] Fritz, W. Mulia, S. Sohoni, B. Gordon, K. Kearney, M. Mwavita, "The Progressive Learning Platform for Computer Engineering", ASEE Annual Conference and Expo (ECE Division), June 2011, Vancouver, Canada.

[12] PLPTool 5.2 Download, accessed May 2018. https://github.com/Progressive-Learning-Platform/progressive-learning-platform/releases

[13] PLPTool Manual, accessed May 2018. http://progressive-learning-platform.github.io/plptool.html

[14] PLP Project Repository, accessed May 2018. https://github.com/Progressive-Learning-Platform

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

[16] Hughes, and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research", Behaviour & Information Technology, 22(2), 2003, pp.127-140.

[17] S. Letovsky, "Cognitive processes in program comprehension", Journal of Systems and software, 7(4), 1987, pp.325-339.

[18] R. Jeffries, "A comparison of the debugging behavior of expert and novice programmers." In Proceedings of AERA annual meeting, March 1982.

[19] R. Butler "Task-involving and ego-involving properties of evaluation: Effects of different feedback conditions on motivational perceptions, interest, and performance." Journal of educational psychology, 79(4), 1987, 474-482.

[20] N. Kluger, and A. DeNisi, "Feedback interventions: Toward the understanding of a double-edged sword." Current directions in psychological science, 1998, 7(3), 67-72.

[21] Williams, "Keeping learning on track: Classroom assessment and the regulation of learning." In F. K. Lester Jr. (Ed.), Second handbook of mathematics teaching and learning (pp. 1053-1098). 2007, Greenwich, CT: Information Age Publishing.

[22] P. Black, and D. Williams, "Assessment and classroom learning. Assessment in Education: principles, policy & practice" 5(1), 7-74, 1998.

[23] J. Shute, "Focus on formative feedback. Review of educational research", 2008, 78(1), 153-189.

[24] M. L. Gick, and K. J. Holyoak, "Analogical problem solving. Cognitive psychology" 1980, 12(3), 306-355.

[25] R. Kanfer, and P. L. Ackerman, "Motivation and cognitive abilities: An integrative/aptitude-treatment interaction approach to skill acquisition." Journal of applied psychology, 1989, 74(4), 657-690.

[26] S. Craig, A. Graesser, J. Sullins, and B. Gholson, "Affect and learning: an exploratory look into the role of affect in learning with AutoTutor. Journal of educational media" 2004, 29(3), 241-250.

[27] Lehman, S. D'Mello, and A. Graesser, "Confusion and complex learning during interactions with computer learning environments" The Internet and Higher Education, 2012, 15(3), 184-194.

[28] L. Craik, and R. S. Lockhart, "Levels of processing: A framework for memory research." Journal of verbal learning and verbal behavior, 1972,11(6), 671-684.

[29] S. D. Craig, "Confusion's impact on learning" In Encyclopedia of the Sciences of Learning, 2012, (pp. 766-767). New York:Springer.

[30] S. D'Mello, and A. Graesser, "Dynamics of affective states during complex learning." Learning and Instruction, 2012, 22(2), 145-157.

[31] Layman, L. Williams and R. St. Amant, "Toward reducing fault fix time: Understanding developer behavior for the design of automated fault detection tools", In Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium, IEEE, September 2007, pp. 176-185.

[32] M. Layman, L. A. Williams, and R. St. Amant, "MimEc: intelligent user notification of faults in the eclipse IDE", In Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering, ACM, May 2008, pp. 73-76.