Usability of Error Messages for Introductory Students

Paul A. Schliep
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
schli202@morris.umn.edu

ABSTRACT

Error messages are an important tool for programmers to help find and fix errors in their code. When an error message is unhelpful it can be difficult to find and fix the mistakes. Error messages are especially critical for introductory programmers in understanding problems with their code. Not all error messages are beneficial for helping novice programmers. This paper discusses how well error messages can help introductory students resolve mistakes in their programs and what aspects make an error message more user-friendly for introductory programmers. After that, we discuss the analyses of syntax, compiler, and exception errors and their results. We then discuss several methodologies and programs developed to help improve the experience a novice programmer has when attempting to understand causes for errors and their results.

Keywords

Novice programmers, usability, error messages, usability studies, compiler errors, syntax errors

1. INTRODUCTION

One of the most important foundations of computer programming is the communication between the system and the user, specifically in the error messages produced by the system. These error messages are especially important for introductory-level computer science students to help them resolve issues in their program because the error messages are the primary source for understanding what is wrong and according to Marceau et al., "[students] lack the experience to decipher complicated or poorly-constructed feedback" [5]. The first rule of good message design is to be sure that the error does not add confusion [3]. Difficulties in understanding error messages often leads to frustration because the error message was either too complicated to understand or led them down the wrong path [6], which can sometimes introduce new errors [1].

Several studies have been conducted on modern programming languages' error messages to study the effectiveness in helping novice programmers debug their program and help learn the concepts and programming languages. The results have shown that students struggle with compiler and syntax error messages [1] [8] (which we will discuss in detail in Section 2) and the general vocabulary of the error messages along with IDE-specific features such as source highlighting can be bothersome for introductory computer science students [6].

Several tools and heuristics are being developed to help address issues in error message usability and its development. The goals of these methodologies are to help introductory programmers learn the language and concepts easier. The goal of this paper is to discuss the analyses of error message design and its usability for introductory students in a class setting (meaning students' interactions with programming in a lab setting and at home), and how these developed methodologies help improve the user experience with error messages.

This paper is divided into five sections. In Section 2 we discuss usability studies, define compiler, syntax, and exception error messages, and discuss imperative and functional programming. In Section 3 we will focus on analyses of the usability of exception messages and compiler messages for introductory students and how those analyses were performed. In Section 4 we discuss the results of those analyses. In Section 5 we introduce three methodologies developed to help improve the error message usability. The first we will cover is Traver's heuristics for compiler message design [8]. Then, we will examine Marceau et al. recommendations for error message design [6]. Lastly, we will explore Denny et al. syntax error enhancement tool [1].

2. BACKGROUND

In order to discuss the analyses of error messages, we need to understand several concepts related to error types and usability. These concepts include compiler errors, syntax errors, runtime errors, usability studies, and Human Computer Interaction. We will then be discussing the programming languages and tools used the analysis of the error messages. They include Racket, C++, and Java programming languages and an overview of Integrated Development Environment (IDE) and DrRacket.

2.1 Human-computer interaction and methods of usability analysis

The study of Human-Computer Interaction, or HCI, is

focused on how computer technology is used, specifically on the interfaces between the user and the programs of the computer. As Traver notes, "HCI is a discipline that aims to provide user interfaces that make working with a computer a more productive, effective, and enjoyable task" [8]. Much of the research presented in this paper is from an HCI point of view, rather than a more technical approach. Thus, the areas we are looking at in error message design is the language used in the message, accuracy and precision of the messages, as well as interface elements such as text highlighting.

In order to analyze these messages in an HCI perspective and attain qualitative and quantitative information about error message usability, a usability test or case study may be performed. A usability test is a technique often used in HCI studies to evaluate a program or product by testing it on users. A case study is a research method that closely studies a group of participants, in the case of this paper, introductory students in a class room, and collects data about participants by observations and interviews. Many of the studies performed on error messages analyzed in this paper are using a case study and usability test design.

2.2 Compiler and Runtime errors

In this paper, we discuss the general usability of error message design, but in order to do so, we will need to define compiler and runtime errors. A compiler is a program that processes source code written in a programming language into a machine language that a computer's processor reads so that instructions can be executed. A compilation error refers to a state when a compiler fails to compile a piece of program source code. A program will not run if there is a compiler error because the compiler will not be able to create executable code to run if there are errors the compiler finds. Then, we examine syntax errors, which is a type of compiler error that occurs when the code does not conform to the syntactical order expected by the parser. Often, these errors occur from variables that are not defined or missing/extra punctuation. The parser, is a program (usually part of a compiler) that receives input as program instructions and builds it into a structural representation of the input while checking for correct syntax of the input. As Kummerfield and Kay note [4], "The usability of compiler errors are important because syntax error correction is the first step in the debugging process. It is not possible to continue program development until the code compiles. This means it is a crucial part of the error correction process." Below is an example of a compiler error. Here, the programmer is defining `seven` to be `2+5`, but forgot to close the parenthesis. The compiler caught the syntax error, so the program did not execute.

```
int seven = (2 + 5;
```

```
error: ')' expected
```

After a program has successfully compiled without any errors, the system will then move to the runtime or execution time phase where it will attempt to execute the program. A runtime error is when there is an error detected during this phase. Runtime errors often indicate problems in the program itself such as running out of memory and can be harder to find and debug. Below is an example of a runtime error in Java. Here, the user wanted to print out a part of the string, "Hello World" but had the wrong bounds in the substring

command (which takes a substring of a string). The error is telling the programmer that the wrong bounds are given and are out of the range of the string "Hello World" where the programmer should have given the bounds of (6,11).

```
String string = "Hello World";
System.out.print(string.substring(6,12));
```

```
java.lang.StringIndexOutOfBoundsException:
String index out of range: 12
```

Not all languages compute programs the same way, however. Some languages, such as Racket (which we define in the next subsection), are considered dynamic in that many actions performed during compilation in other languages are performed during the runtime phase in a dynamic language. Such actions that are done at compile time are checking object types, where in a dynamic language this is done at runtime. This means that a dynamic language will handle some errors differently. We discuss these differences in subsection 2.3.

2.3 Overview of programming languages and tools analyzed

In order to discuss the research and studies done on error messages, we need to define the languages and programs used. In subsection 3.1, we discuss a study performed by Marceau et al. that analyzes the error messages in the Racket programming language and DrRacket integrated development environment. Racket is a member of the Lisp family of programming languages and is designed for students new to programming. Racket is a functional language, which means it uses a programming style of building elements of programs while retaining immutable data structures and without directly manipulating memory or changing state. Functional languages generally work well in teaching programming concepts to students since functional approaches emphasize core computer science concepts such as recursion. Racket is a dynamically typed language, which means type checking is performed at runtime rather than compile time and thus it is possible to bind a name to objects of different types during the execution of a program since every variable name is bound only to an object. Since the interpreter of a dynamically typed language deduces type and type conversions, a programmer does not have to worry as much about type declaration.

An integrated development environment, or IDE, is an application that has packaged several other programs typically consisting of a text editor, compiler, and other programs used to debug code. DrRacket is an IDE meant for writing programs in Racket commonly geared toward introductory students. DrRacket offers useful tools for introductory programmers in debugging such as highlighting the offending line when an error occurs.

In subsection 3.2, we discuss an analysis on compiler errors in C++ and in subsection 4.3, we discuss a system meant to enhance syntax error messages in the Java programming language. C++ and Java are widely used programming languages not designed for introductory programming, but are more designed toward system programming.. However, C++ and Java are occasionally taught in introductory computer science classes. C++ and Java are both imperative languages, which is a programming style that, as opposed to functional programming, uses a sequence of

statements to build a program using memory manipulation and changing the state of objects in a program. These languages both fall in the style of Object-oriented programming, or OOP, which is a method of programming based on class hierarchy and is based around creating objects, which are data structures that contain a set of routines called methods. Java and C++ are statically typed languages, which means type checking is done at compile-time rather than runtime. This means that when programming in statically typed languages, a programmer must pay attention to type assignment, but provides benefits such as earlier detection of programming mistakes.

Since Java and C++ are statically typed languages, a programmer can receive type errors during compile time. However, the same error will not occur in a dynamically typed language since type checking is done during runtime. Consider the following example:

```
personName = "Francis"
personName = 7
```

This sequence of statements is illegal in a statically typed language since we are binding a string to `personName`, then an integer to`personName`. This statement would then throw an error during compile time. This is a legal statement in a dynamically typed language, however, since the interpreter deals with types during the runtime phase.

3. ANALYSES

This section will discuss two different studies performed on the usability of error messages and what the results of those studies are. The first analysis will discuss how well the error messages in Racket and DrRacket help introductory students debug their programs. The second analysis will discuss the effectiveness of compiler error messages in the C++ programming language.

3.1 Analysis of error messages in Racket and DrRacket

Marceau, Fisler, and Krishnamurthi helped design DrRacket error messages so that they can be more helpful to beginner programmers. However, Marceau et al. still noticed students struggling with debugging and understanding the error messages, so the authors were interested in seeing how their students responded to these error messages [6]. In the spring of 2010, Marceau, Fisler, and Krishnamurthi ran a case study on error messages in DrRacket. The study involved configuring DrRacket to save a copy of each program a student tried to run as well as the error message through six 50 minute lab sessions [5]. The authors were interested in which error messages are effective and how well DrRacket's text highlighting can help a student.

In order to measure effectiveness, the authors developed a rubric which determined whether the student made a reasonable edit in response to the error message [5]. The rubric was meant to distinguish how an error message would fail or succeed. They determined that an error message is effective if a student can read it, understand it, and use that information to figure out how to resolve the issue.

Figure 1 shows an example of an error message in Racket that is not effective for a student. The message is contradicting itself as it does follow an open parenthesis, but the parser thinks the `and` is an independent part from the `cond`.

```
(define (label-near? name bias word1 word2 word3)
  (cond
    (and (cond [(string=? name word1) "Name Located"]
               [(string=? bias word1) "Bias Located"])
         (cond [(string=? name word2) "Name Located"]
               [(string=? bias word2) "Bias Located"])
  "Mark")
))
```

✎ `and: found a use of `and' that does not follow an
open parenthesis`

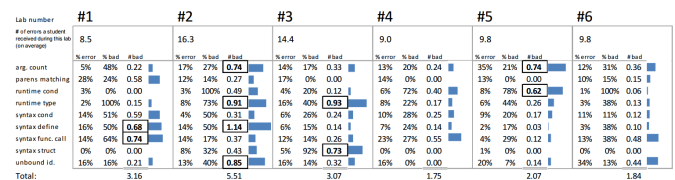**Figure 1: Example of an ineffective error message in Racket**

Figure 2: Results from DrRacket study

Marceau, Fisler, and Krishnamurthi grouped messages into nine most common error categories in their results from the study. Their rubric found the level of severity of the error messages. Through their data collection at the end of the study, they found the level of occurrence of each error type from each lab and which error messages were poorly responded to, indicated as *%error* and *%bad*, respectively. *#bad* shows the level of likelihood of recurrence of the respective error message. The values of interest are the values enclosed in a box, which are the higher level *#bad* values, as seen in figure 2.

The authors found from the data, that students have difficulties with certain errors at different points in the course. For example, in Lab 5 of the course, the authors found that the numerous argument count errors trace to mistakes in properly closing expressions. The error messages that the student saw from these mistakes did not match with the actual error the student encountered with their program.

> Rephrase paragraph

> fix image positions

3.2 Analysis of compiler messages in C++

Compiler error messages are often cryptic and difficult to understand for many programmers, especially for students who are new to programming. Unfortunately, as Traver notes, "most related disciplines, including compiler technology, have not paid much attention to this important aspect that affects programmers significantly, apparently because it is felt that programmers should adapt to compilers." [8] Not a lot of research has been conducted on compiler error message design. In the first semester of 2002-2003, Traver conducted a case study on students' work with compiler error messages in C++ in an introductory computer science course. The motivation of this study is to gain insight on what students are struggling with in the course and to help the professor's personal struggles with these error messages.

Traver gathered data from the students' interactions with C++ throughout the semester and wrote up analyses of the error messages received in 5 separate parts.

- *The error message* received from the compiler

- *The source code* that caused the original error

- *The diagnostic* of why the error occurred

- *An alternate error message* that may help lead more directly to the true diagnosis of the issue.

- *A comment* about why the error message is not helpful

Below is an example of an error message in C++ analyzed in the study along with the source code [8] (in the interest of space, I have not included the other parts of the analysis):

Source Code:

```
class SavingAccount
friend ostream & operator<<
ostream &os, const SavingAccount &sA);
};
```

Error Message:

```
ANSI C++ forbids declaration
'ostream' with no type 'ostream'
is neither function nor method;
cannot be declared friend
parse error before '&'
```

`find a better code snippet`

In this case, the user forgot to include a header file (iostream.h), so the compiler does not know what iostream is. The error message however, did not suggest to include a header, a simple fix, where the original error message could easily confuse novice programmers. The author of the study noted that this type of error message should "convey a clear message that the programmer can quickly understand and that is useful for fixing the error", but the error message given to the user would not accomplish this for students who are still new to programming [8].

Traver found from the study, that there is an apparent lack of thought put into the usability of compiler error messages and that many students, especially those new to programming, will have a hard time understanding these errors. There was no quantitative data gathered on these error messages, but rather just a general observation of how students struggled working with the compiler error messages. Traver noted that "better [error messages] are possible" and was able to provide alternative error messages for each of the ones analyzed [8]. So, better error message design is possible, such that more efforts and time are put into compiler error message research.

4. METHODOLOGIES

In this section, we will be discussing three tools and methodologies meant to attempt to improve the usability of error messages or suggest improvements for error message design based on the results of the analyses discussed in section 3. The first methodology we will discuss is a set of recommendations for improving the usability of error messages. The

second approach discusses improving compiler error messages through a set of principles meant to increase usability of the messages. For the third approach, we will discuss an attempt on enhancing syntax error messages in Java and the how well these syntax error messages improve over the original.

4.1 Recommendations for error messages

This will use my source, Mind Your Language: On Novices' Interactions

`TODO`

4.2 Principles of compiler error design

This will use my source, On Compiler Error Messages: What They Say and What They Mean

`TODO`

4.3 Syntax error message enhancement and results

This will use my source, Enhancing Syntax Error Messages Appears Ineffectual

`TODO`

5. CONCLUSION

Discussion of the direction usability of error messages will be taken and how the methodologies will be applied for future work.

`TODO`

6. ACKNOWLEDGMENTS

`TODO`

7. REFERENCES

[1] DENNY, P., LUXTON-REILLY, A., AND CARPENTER, D. Enhancing syntax error messages appears ineffectual. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (New York, NY, USA, 2014), ITiCSE '14, ACM, pp. 273–278.

[2] HARTMANN, B., MACDOUGALL, D., BRANDT, J., AND KLEMMER, S. R. What would other programmers do: Suggesting solutions to error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2010), CHI '10, ACM, pp. 1019–1028.

[3] ISA, B. S., BOYLE, J. M., NEAL, A. S., AND SIMONS, R. M. A methodology for objectively evaluating error messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1983), CHI '83, ACM, pp. 68–71.

[4] KUMMERFELD, S. K., AND KAY, J. The neglected battle fields of syntax errors. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20* (Darlinghurst, Australia, Australia, 2003), ACE '03, Australian Computer Society, Inc., pp. 105–111.

[5] MARCEAU, G., FISLER, K., AND KRISHNAMURTHI, S. Measuring the effectiveness of error messages designed for novice programmers. In *Proceedings of the 42Nd*

ACM Technical Symposium on Computer Science Education (New York, NY, USA, 2011), SIGCSE '11, ACM, pp. 499–504.

[6] MARCEAU, G., FISLER, K., AND KRISHNAMURTHI, S. Mind your language: On novices' interactions with error messages. In *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (New York, NY, USA, 2011), Onward! 2011, ACM, pp. 3–18.

[7] MURPHY, C., KIM, E., KAISER, G., AND CANNON, A. Backstop: A tool for debugging runtime errors. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2008), SIGCSE '08, ACM, pp. 173–177.

[8] TRAVER, V. J. On compiler error messages: What they say and what they mean. In *Advances in Human-Computer Interaction* (2010).

Citing sources for references

[1] [2] [3] [4] [5] [6] [7] [8]