

# BACHELOR'S THESIS COMPUTING SCIENCE



RADBOUD UNIVERSITY NIJMEGEN

---

## Creating Better Error Messages by Improving Their Presentation

---

*Author:*  
Thomas Rhemrev  
s1045660

*First supervisor/assessor:*  
Dr. Mara Saeli

*Second assessor:*  
Prof. Erik Barendsen

## **Abstract**

First-year computer science students frequently encounter difficulties in programming. A major contributing factor to these challenges is the presence of unclear and vague error messages. These unhelpful error messages are an especially big problem for novice programmers, as they lack the necessary experience to immediately see what is going wrong and are therefore dependent on error messages to solve errors. In this thesis, we present several ways in which the presentation of error messages can be improved to make them more helpful for novice programmers. We then go on to implement and evaluate some of these methods. Finally, we provide evidence that indicates that they are an improvement upon standard error messages.

# Contents

<b>1</b>	<b>Introduction and related works</b>	<b>2</b>
1.1	Theoretical Background . . . . .	6
<b>2</b>	<b>Methods</b>	<b>8</b>
2.1	Survey . . . . .	8
2.2	Evaluation of the Enhancements . . . . .	9
2.2.1	Assignment . . . . .	9
2.2.2	Evaluation Interview . . . . .	10
<b>3</b>	<b>Analyzing the survey data</b>	<b>12</b>
3.1	Current Errors . . . . .	12
3.2	Enhanced Errors . . . . .	14
<b>4</b>	<b>Design</b>	<b>17</b>
4.1	DrJava . . . . .	17
4.2	Enhanced Error Messages . . . . .	18
<b>5</b>	<b>Results</b>	<b>21</b>
<b>6</b>	<b>Conclusion and Discussion</b>	<b>24</b>
6.1	Findings . . . . .	24
6.2	limitations . . . . .	27
6.3	future work . . . . .	27
6.4	conclusion . . . . .	27
<b>A</b>	<b>Survey</b>	<b>i</b>
<b>B</b>	<b>Assignment</b>	<b>x</b>
B.0.1	Class Main . . . . .	x
B.0.2	Class Rectangle . . . . .	xi
<b>C</b>	<b>Evaluation Interview</b>	<b>xiii</b>

# Chapter 1

## Introduction and related works

In a multi-institutional study, McCracken et al[15] showed that first-year computing science students are not nearly as proficient at programming as their teachers expect. One of the hurdles novices face when learning how to program is cryptic and unclear compiler error messages [4, 22]. The feedback compiler error messages give is especially important to novices, as they do not yet have enough experience to immediately understand what might be going wrong, so the error messages given by the compiler are often their primary guidance while debugging [3]. Barik et al[2] provide empirical evidence that students spend a substantial portion of their time (13-25%) on error messages, but find them difficult to understand and frustrating to deal with.

The process of fixing an error can be split up into 3 steps [14]. First, programmers have to identify the location of the error in the program. Second, they have to understand the mistake that was made, and finally, they have to fix the error. A good error message should help the programmer with all 3 of these steps.

Methods for improving error messages can be split into two main categories: changing their contents or changing their presentation [4]. The first of these methods focuses on improving the text of the error messages. This can, for example, be done by replacing technical jargon to make the messages easier to understand, or by making preliminary suggestions as to how to fix the error[3]. The second method keeps the wording of the errors the same but focuses on other ways they can be improved. This can for example be done by highlighting the part of the code where the error occurs, underlining important terms in the error message, or by linking to documentation.

Many authors have suggested improvements that focus on enhancing the contents of error messages. Becker [3] proposed an enhanced Java compiler called *Decaf*, and investigated its effectiveness in reducing student errors. *Decaf* provides a natural language interpretation of the error message, as well as a potential solution to the error if available. This enhanced message is shown together with the original message in order to allow the student to get used to standard error messages. When testing the results of the compiler, Becker found a small but significant decrease in the overall number of errors and the number of repeated errors for students using the *Decaf* editor when compared to a control group.

Pettit et al [17] provided enhanced errors that were similar to those made by Becker [3]. The compiler gives an explanation of the error in simple English and provides a potential solution. In contrast to Becker’s study, they found no evidence that enhanced error messages reduced the number of repeated errors committed by novices.

Denny et al [5] implemented enhanced error messages as a part of their *CodeWrite* program [6], a tool designed to help students practice writing Java code. These enhanced messages contain a detailed explanation of what is most likely causing the error. It also shows a piece of example code with an error of the same type, side-by-side with the correct implementation of that code snippet and an explanation as to how it was fixed. In an evaluation of the effectiveness of these enhanced messages, they again found no significant effects.

Watson et al [25] created *Bluefix*, a tool that automatically uses crowdsourced information to help students diagnose and fix errors. This tool was specifically aimed to help students learn, and gave more detailed and specific information if a student has been stuck on a certain error for a long time, so it can not be used as a general-purpose error message.

Also making use of crowd-sourced information, Thiselton and Treude [21] introduced *Pycee*, an extension that automatically queries Stack Overflow to generate more helpful error messages.

There has been comparatively little research into ways to improve the presentation of error messages. Many programming environments provide some presentational error cues to programmers. Some IDEs, such as Eclipse and VS Code, mark the offending code by underlining it with squiggly lines, which makes it easier to locate your mistake. VS Code also shows the error message in a pop-up box when hovering over a piece of erroneous code. However, little research has been done to test the effectiveness of these measures.

Another development environment that improves the presentation of error messages is BlueJ [13]. BlueJ is an environment specifically created to help novice programmers learn Java. It shows error messages in a pop-up box to ensure programmers will see them, similar to what VS Code does. It also only shows one error message at a time to help novices only focus on a single error at once. On top of this, it provides other improvements such as providing students with boilerplate code when creating classes/objects and using graphs to give students a clear overview of how different classes are related. Research indicates that BlueJ does improve the performance of students [8]. However, since it contains many other improvements on top of the error messages, it is not clear if the changes made to the messages were actually helpful for students. In fact, one of the student complaints mentioned in the paper is that “the error messages are not very helpful in the compiler”.

Because research into enhancing the content of error messages is inconclusive and has not produced the desired results, it might be valuable to focus on the presentation. Some tools have been developed that improve both the content and the presentation (see Table 1.1). For this research, however, we will focus solely on improving the presentation of error messages. The goal is to investigate the effect of the presentation on

the effectiveness of error messages, and the thesis will aim to suggest some methods in which this presentation might be improved. It will also test if these proposed methods improve student performance.

The research question that will be guiding this thesis is: “*What are characteristics related to error message presentation that can be used to improve the effectiveness of error messages for novice programmers?*”. For this research, novice programmers will be first-year computing science students. In particular, we will be investigating students who are currently following the first-year Object-oriented programming (Java) course at Radboud University.

The research will start by doing a literature review to come up with ways in which the presentation of error messages can be improved. Next, we will conduct a survey among first-year computing science students. This survey will help gather information on how they use error methods and what struggles they have with them. It will also allow us to get user feedback on the potential enhancements identified in the literature review. We will then use this data to design and implement a set of enhanced error messages. Finally, we will be conducting an experiment to evaluate the effectiveness of these enhanced error messages

The structure of this thesis will be as follows. The remaining part of chapter one will be used to provide a theoretical background and explain the theory behind the enhancements that will be investigated during this research. Chapter 2 will explain the methodology followed for our research. Chapter 3 will display the results of the survey. Chapter 4 will describe the set of enhanced error messages we created. Chapter 5 will show the results of our experiment. Finally, in Chapter 6 we will discuss our findings and draw conclusions.

	Content			Presentation		
	Readability	Examples	Suggestions	Highlight	Location	Amount
Decaf[3]	Displays a natural language interpretation alongside the original error	-	For a select number of errors	-	-	-
Codewrite[6]	Explains the error in simple terms	provides examples of similar errors	-	-	-	-
Athene[17]	Displays a natural language interpretation alongside the original error	-	Provides potential solutions for a select number of errors	-	-	Only shows one error at a time
Espresso[9]	Provides simple explanations for common errors not caught by a normal compiler	-	Yes	-	-	-
Pycee[21]	Shows explanations of the error by querying Stack Overflow	Provides Examples gathered from Stack Overflow	Provides solutions from Stack Overflow	-	-	-
BlueJ[13]	Gives shorter error messages	-	Provides suggestions for a select few errors	Underlines offending code in red	Shows error message in a pop-up box next to the offending code	Only shows one error at a time
Eclipse	-	-	provides hints when the user clicks the light-bulb icon	Highlights offending code with a red squiggly line	-	-
VSCode	-	-	provides a “Quick Fix” for select types of errors	Highlights the offending code in real-time	Shows error message in a pop-up when hovering over highlighted code	-

Table 1.1: Types of improvements provided by some error enhancement tools as well as some popular programming environments

## 1.1 Theoretical Background

The Cognitive Fit theory [24] proposes that performance on problem-solving tasks goes up if the problem is represented in a way that matches the task. The original study investigated the effects of using graphs vs tables, but the underlying theory might still be useful in creating better error messages.

You can apply this theory to error messages by matching the style of the error messages to the style of the code. For example, this could be done via matching font style and colors.

Another cognitive theory that might be helpful is the cognitive load theory [20]. According to this theory, there are two types of memory: long-term memory, which holds information for long periods of time, and working memory, which is highly volatile. This working memory has a finite capacity for efficiently processing input when problem-solving. By reducing the cognitive load of error messages, students have more capacity left for solving the errors.

Most of the previously mentioned studies did not account for cognitive load. The effects of high cognitive load in these enhancements have been given as a possible explanation for why “better” error messages, such as those created by Denny et al [5] or Pettit et al [17], do not improve student performance [18]. Hundhausen et al [10] provide 3 principles that can be used to reduce cognitive load in error messages:

- Reduce redundancy in the information presented so that the reader does not unnecessarily process the same information multiple times
- Use multiple modalities (i.e visual and auditory).
- Place information that is necessary for the programming task physically close to where the task is being completed.

These guidelines give us a few ways in which the cognitive load of error messages can be improved. The first of these guidelines, reducing redundancy, can be implemented by only showing one error message at a time, as is done in BlueJ [13]. Not only does this reduce the cognitive load, but it also helps the programmer focus on the error and prevents them from being overwhelmed. Additionally, consolidating error messages of the same type into one single message could also help in reducing redundancy, as suggested by Hundhausen et al[10].

Hundhausen et al[10] also gave a suggestion for how the second principle, presenting the information across multiple modalities, can be used to improve the presentation of error messages. Namely, they suggest that it can be done by having the machine read the error out loud, and by visually highlighting the error on the screen.

Finally, according to Hundhausen et al [10], the last of these guidelines, placing necessary information close to the task, can be done by physically placing the error message close to the offending code.

Placing information near the code might be helpful in more ways outside of reducing the cognitive load. The importance of providing context to error messages has been



brought up before in the literature, for example by Nienaltowski et al[16]. This context can include things like the location of the offending code or the variables involved in the error. Barik [1] has suggested that displaying the error message inside the text editor, right next to the line that caused the error, can help to provide this necessary context.

Most compilers provide unique message identifiers for each type of error message. These can be used to find additional information, including a more detailed explanation of the error, an explanation as to what might have caused it, and suggestions for how the error can be fixed. This information is usually hidden behind an online search or several clicks through documentation. Becker et al [4] suggest that easy access to this supplemental information might help novice programmers better understand and fix the error. This idea is further supported by Kadekar et al[12], who found evidence that linking to additional information in error messages reduces the amount of time it takes to fix an error.

Alternatively, Hundhausen et al [10] have suggested that placing this additional information next to the error message might help reduce cognitive load, as it would mean that the programmer no longer has to dedicate their processing capacity towards finding this information.

# Chapter 2

## Methods

As was mentioned in the introduction, this thesis will suggest some methods for improving the presentation of compiler error messages. This chapter will describe how these enhancements are designed and evaluated.

### 2.1 Survey

For this thesis, we will be designing error messages that are better suited for novice programmers. Before designing these enhanced errors, we want to gain some insight into how students interact with and think about error messages. For this, we will be using a survey. This section will describe what information we aim to obtain from this survey and also describes how we will learn that information. The full survey can be found in Appendix A.

The survey will be run on first-year computing science students following the course “Object Oriented Programming” at Radboud University. These students are currently learning the Java programming language. Since our error messages are designed to be used by novice programmers, the students targeted by the survey should be a good representation of our target audience.

First, we want to know how students normally interact with error messages. We are especially interested in how and when they use error messages. This will tell us what our error messages should be able to do. It also helps guide what areas our improvements should focus on. These insights will be gained by asking students to describe the process they go through when debugging compiler errors.

Second, we want to know what the students think of the current error messages. Knowing their opinion on error messages lets us identify problem areas with error messages and again helps guide our improvements. It also provides a point of comparison that will be used when evaluating our enhanced error messages. This information will be gained by asking the students to rate the effectiveness of current error messages in helping them find, understand and fix the errors. These are the three main goals of error messages, as described by Lazonder et al[14]. We also ask students to mention problems they might have with error messages, and ask them to come up with enhancements of

their own.

Finally, we want the student's opinion on some of the potential improvements as described in section 1.1. To do this, we create a mock-up for each suggested improvement and ask students to rate them. We also show a standard error message and ask the students to rate it. The ratings of the suggested changes and the standard error messages are compared to see if students prefer these enhancements over the standard error messages.

## 2.2 Evaluation of the Enhancements

Based on the results obtained from the survey, we will design a set of enhanced error messages. This section will describe how the error messages created in this study are evaluated. Because we only have a small number of participants for this part of the research, this is done using qualitative research. First, the participants are given a small programming assignment to complete using our enhanced error messages. The students are given 20 minutes to complete the assignment. They are then asked some questions about their experience with the enhanced error messages in an interview.

The experiment has three participants. Again, all participants are first-year computing science students following the object-oriented programming course at the Radboud university

The assignment is completed in a think-aloud study. This means that the participants are asked to verbalize their thought processes as they are working through the assignment. We also record their screen as they are programming, which provides us with additional information on their process. The idea of using a think-aloud experiment to capture a programmer's experience was first explored by Vessey[23] while studying the debugging habits of experts. In our case, we use a similar setup to get an idea of how novice programmers make their way through the assignment.

This setup allows us to get a good idea of how the participants interact with the enhanced error messages. It tells us if they actually interacted with the error messages and if the messages worked as intended. Knowing the participant's thought processes also shows us the effects of the error messages on this thought process. It also helps provide context to the participant's answers to the questionnaire.

### 2.2.1 Assignment

The assignment given to the students consists of a piece of broken code. Specifically, the program contains ten compiler errors. We have chosen to use this setup as it ensures that students will encounter error messages during the experiment. The code of the assignment, including the errors introduced, can be found in Appendix B.

The program is a simple application that sorts rectangles by their area. It consists of two classes: `Rectangle` and `Main`. The `Rectangle` class represents a rectangle object with its width and height as private variables. It provides methods to calculate the area of the rectangle and retrieve its dimensions.

Error Message	Mistake Introduced
“illegal start of expression”	Declare the static variable <code>rectangles</code> inside the static method <code>main</code>
“; expected”	Removed <code>;</code> at the end of the line
“) expected”	Added <code>;</code> after the updation in the for-loop
“invalid method declaration; return type required”	Removed return type from function definition
“cannot find symbol”	Removed the definition of <code>scanner</code>
“incompatible types”	Pass a character to <code>equalsIgnoreCase</code> , which requires a string input
“illegal start of expression”	Declare the static variable <code>n</code> inside the static method <code>sortRectanglesByArea</code>
“{ expected”	Removed opening brace after class definition
“incompatible types”	Define <code>Width</code> and <code>Height</code> as integers but try to pass a double to them.
“not a statement”	Removed the return before <code>width * height;</code>

Table 2.1: List of compiler errors introduced in the assignment

The `Main` class contains the `main` method and serves as the entry point of the program. It allows the user to input rectangle objects by providing their width and height. The rectangles are stored in a list. The program then prints the rectangles before sorting them by their area. After sorting, it displays the rectangles again to show the sorted order. The main class includes the `readRectanglesFromInput` method, which allows the user to input rectangles, the `sortRectanglesByArea` method, which sorts the rectangles by their area using a bubble sort algorithm, and finally the `printRectangles` method, which prints a list of rectangles.

As we said before, we introduce ten compiler errors into the program. These errors are chosen from the twenty most common errors among novice programmers, as found by Jackson et al[11]. The list of errors introduced can be found in Table 2.1.

Six of these error messages are unique, meaning that that specific error occurs only once in the program. The “illegal start of expression” and “incompatible types” errors occur twice in the program. This is done to test the effect of the enhanced error messages on repeated errors.

### 2.2.2 Evaluation Interview

After the participants have finished the assignment, we want to know what they thought of our enhanced error messages. To get this information, participants will be interviewed and asked questions on the effects of the enhancements we made to the error messages. The full list of questions used to guide the interview can be found in Appendix C.

We first want to know how effective the enhanced error messages are in performing the three main tasks of error messages[14]. This tells us what the strong and weak

points of our error messages are. To do this, we again ask the participants to rate the effectiveness of the error messages in helping them locate, understand and solve the errors. These questions were also asked about standard error messages in the first survey, which allows us to compare the results.

Next, we want to know what the participants thought of the enhancements we made. To get this information we ask them for their opinions on each of the individual improvements we have implemented. This will allow us to get gain insight into how useful each of them was in practice. It will also tell us something about the effect they had on the participants' error-solving process.

Finally, we want to know about any problems the participants might have with the enhanced error messages. Therefore, we ask them to mention any shortcomings our enhancements might have and ask if anything was missing. This can help in evaluating the error messages and might guide the design of any potential future iterations of the enhanced error messages.

## Chapter 3

# Analyzing the survey data

In this section, we will be analyzing the data generated by our survey. The analysis is split in two parts. First, we take a look at students' experiences and opinions on current error messages. Second, we analyze the rating given to the suggested changes and compare them to the rating given to current error messages.

### 3.1 Current Errors

Looking at the data, we see that the majority of the respondents (roughly 85%) make use of error messages when trying to solve an error. However, this number does not paint the full picture. Almost half of these students do not actually read the error message. Instead, they only use them to find the location of the error. They then use other methods to actually try to find what was causing the error.

The most popular other method is looking up the error online. 18 out of the 21 responses mention the use of an online search engine or Q&A platform such as Stack Overflow as a part of their error-solving process. Worryingly, many students indicate that they simply copy-paste the error message into a search engine without even trying to read and understand it for themselves.

Another popular debugging method is using manual debugging. This usually involves putting print statements at various points in the code, and seeing what if the program prints out a result. This way, the students see what parts of the code are run and what kind of results this code produces. This information helps them to pinpoint the problem and provides important clues as to the nature of the error. survey results indicate that this method of manual debugging is used more often than the actual debugger that comes built-in with practically all IDEs.

Additionally, the results show that students see locating the error as an important job of the error message. This observation is further backed by results shown later on in this section, where students indicate that locating the error is one of the most helpful

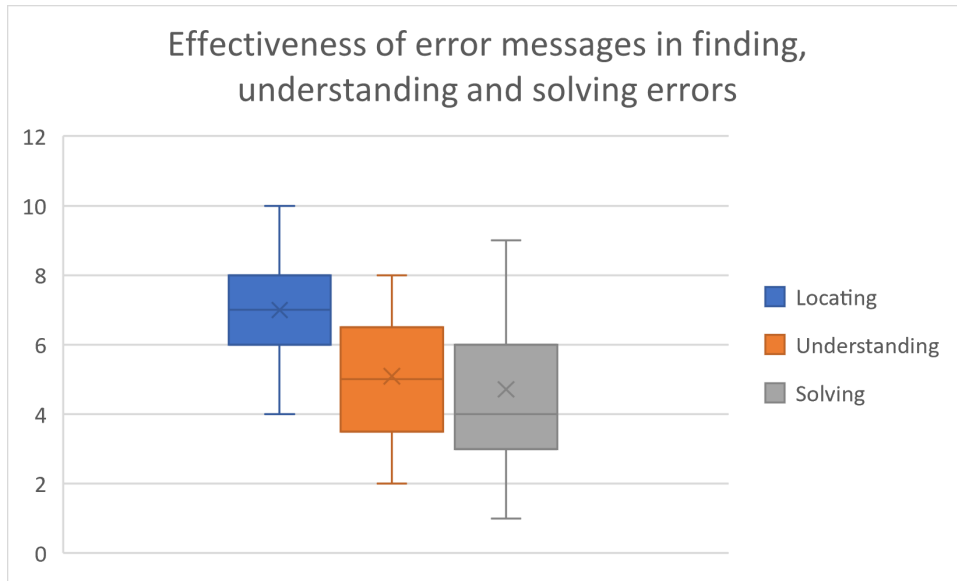


Figure 3.1: visualization of respondent’s ratings of the effectiveness of current error messages in helping them locate, understand and fix errors

aspects of error messages. On top of this, as we will see later on, enhancements that are focused on making it easier to locate the error are given the highest ratings by students. As described by Lazonder and van der Mei[14], error messages have 3 main jobs. They have to help the programmer find, understand and solve errors. According to the survey, most students think that error messages are good at helping to locate the error, giving them an average rating of 7 in this category (see Figure 3.1). However, students find that current error messages are ineffective in helping them understand and solve errors, rating them an average of 5,1 and 4,7 in these two categories respectively(again, see Figure 3.1).

This observation is further backed up by the questions where students were asked to mention the most and least helpful aspects of error messages. 14 out of the 21 respondents named helping them find the error as the most helpful aspect of error messages. On the other hand, one of the most common complaints was that error messages were vague, generic, or otherwise unclear. Many respondents mentioned that they often need to google the error message to understand what it means. Additionally, many of the enhancements suggested by the students include some form of rephrasing the error messages to make them easier to understand.

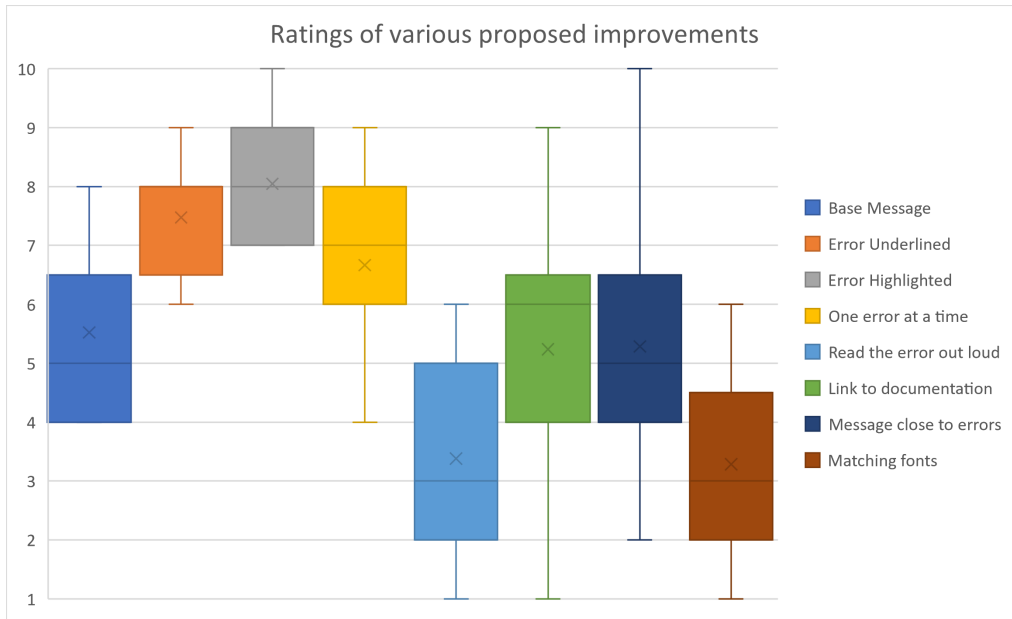


Figure 3.2: Ratings given by students to various proposed enhancements

## 3.2 Enhanced Errors

In the survey, we ask students to rate 7 potential improvements that could be made to the presentation of error messages. The improvements we ask about will be described below. Appendix A includes the mockups shown in the survey.

The first two improvements we included were meant to make it easier to locate the error. The first one underlines the mistake with a red squiggly line, similar to how text editors like Microsoft Word underline spelling mistakes. This method of underlining mistakes should be familiar to students. The second one highlights the entire line in a bright color and highlights the specific token that is causing the error in red.

The next few improvements are meant to reduce cognitive load and are based on the 3 design principles for reducing cognitive load as described by Hundhausen et al [10].

The third potential improvement included in the survey is one where we only show one error at a time to the programmer. This improvement is based on the principle of limiting the amount of information shown to the user. The fourth proposed improvement involves making the program read the error message out loud and is based on the principle of using multiple modalities. The fifth improvement provides a link to relevant documentation. This follows the principle of placing relevant information nearby where the task is being completed and is further supported by research conducted by Kadekar et al[12].

The sixth suggested improvement places the error message physically close to the erroneous code. This improvement is also based on the principle of placing relevant information close to where the task is being completed. On top of this, placing the error close to the code also helps provide context, as suggested by Barik[1].



The 7th and final proposed enhancement is based on the cognitive fit theory[24]. It involves displaying the error message in the same style as the rest of the code.

Finally, we also ask students to rate a normal unenhanced error message, which will be used as a point of comparison to the seven proposed improvements.

The ratings given by the responding students can be seen in Figure 3.2. The “Error Highlighted” Enhancement is a clear favorite among respondents. Students indicate that they like that it made it easier to quickly locate the mistake. Students prefer it over the “Error Underlined” enhancement because the highlighting makes it stand out more and is therefore easier to find when compared to the underlining. However, a minority of responses indicate that they prefer the simplicity of the underlining.

The “One error at a time” enhancement is also preferred over standard error messages. Students who rated it highly mentioned that it helps them focus on the problem at hand.

Both the “Message close to errors” and “Link to documentation” enhancements have an average rating that is roughly equal to the standard error messages. The spread in ratings given to these enhancements is however much wider when compared to the other proposals. This indicates that they were somewhat controversial, as many students either gave them very high or low ratings.

For the “Message close to errors”, many of the people who like it mention that it helps them in locating the error message. This is an unforeseen additional benefit of the enhancement on top of providing context and reducing cognitive load. On the other hand, the respondents who dislike the proposal mainly dislike the fact that it breaks up the flow of the code, thereby actually making it more difficult to see the full context of the error.

In the case of “Link to documentation”, many of the students who like it say that it makes it easier to understand what is going wrong. It also saves them time, as many of them also indicated that they already use external resources such as Google as an integral part of their error-solving process. Linking the relevant information directly in the error messages saves them a few clicks. The people who dislike it mention that such functionality would make the error message too busy. Some also say that such a feature would not be useful, as websites such as Stack Overflow do a much better job of providing relevant information than the standard documentation.

The “Matching Fonts” and “Read the error out loud” both receive scores that are much lower than the standard base error message. For Matching fonts, participants report that it is too similar to normal output which makes it difficult to see that there even is an error. In the case of reading the error out loud, they report that it would just be distracting to them, while not providing any real benefits.

In general, a significant portion of the respondents stressed that they rate simplicity and subtlety above all else. These people dislike changes that introduce extra distractors, such as reading the error out loud and putting the message next to the error. On the other hand, they like the underlining and showing one error at a time, as they make things simpler, or provide big benefits without adding additional distractions. Cutting down on unnecessary stimuli and distractors massively decreases the cognitive load,

which in theory should improve student performance.

Many of the most well-liked improvements focus on making it easier to find the location of the errors. This is noteworthy, as students also indicate that helping you locate the error is already one of the strong points of current error messages. Locating the mistake is already seen by many students as the most important job of error messages, so it makes sense that improvements that make this easier are rated highly.

# Chapter 4

## Design

As a part of our research, we implement error messages with enhanced presentation. This section will describe the enhanced error messages made for this study. It will show the design choices we made and explain the rationale behind them.

### 4.1 DrJava

In order for us to implement the enhanced error messages, we need an IDE. Since creating an IDE from scratch is very time-consuming, we have instead chosen to use an existing IDE as a base to build our enhanced error messages on top of. For this base, we have chosen to use DrJava[7]. This section will describe the DrJava environment.

We have chosen to use DrJava as a base because it is designed to be relatively lightweight. DrJava is designed to be as simple as possible so that students do not have to spend valuable time learning and getting used to the complexities of other popular IDEs. For us, this simplicity means that the environment is a blank slate on which we can add our own enhancements. The lack of many complicated features also means that there are few outside variables influencing our experiment. We do not want participants to base their opinion of our enhancements on features that are built into the IDE, but on the presentational enhancements made for our experiment.

The DrJava interface consists of two main panes: The interactions pane and the definitions pane. In the definitions pane, the user can enter and edit class definitions. In the interactions pane, the user can input Java expressions and statements and immediately see their results.

The interactions pane is built around a Read-Evaluate-Print Loop(REPL)[19]. It takes single user inputs, executes them, and immediately returns the results to the user. A REPL facilitates incremental development.

For our purposes, the definitions pane is more important, as that is where most of the coding happens, and where most of the presentational changes are displayed. The DrJava editor supports a small number of features. Like most other IDEs, it has automatic indentation and keyword highlighting.

One less common feature DrJava has is parenthesis matching. When the user clicks on or in between a set of parentheses, it will highlight the area in between them. This helps the user quickly find where the parentheses open and close. Unfortunately, we had to disable this feature for our implementation. This is for two main reasons. First of all, it is a feature that is not present in most other IDEs and introduces an additional outside variable in our experiments. When we let students use our implementation, we do not want them to factor this feature into their ratings of the experience. Secondly, it clashes with some of the presentational enhancements, namely the error highlighting, that we have implemented.

## 4.2 Enhanced Error Messages

For this research, we have implemented a number of presentational changes to the error message. These enhancements are based on prior research described in section 1.1 and on the data obtained from the survey as laid out in section section 3.

Of the potential improvements mentioned in section 1.1, we have chosen to implement 2: Highlighting the error and only showing one error at a time. These are the suggestions that have the most support among novice programmers according to our survey.

In the survey, many students indicate that locating the code is the most important job of an error message. Respondents were asked to rate two suggested changes that are meant to make it easier for the user to locate the error. The first one marks the error by underlining it with a squiggly red line. The second one highlights the line where the error occurs. Of these, the second one is the most popular, which is why we chose to implement it in our enhanced error messages.

When an error occurs, the line of code that is causing the error is highlighted in orange. Additionally, the token that is causing the error is highlighted in red. When you click elsewhere in the editor, the highlighting disappears. It appears again when the user either clicks on the line that is causing the error or on the error message. This highlighting can be turned off by toggling the “Highlight source” button in the bottom right.

As mentioned in section 3, we found that many students do not use error messages in the way they are intended. Many only use them to locate the error or immediately copy the error into an online search engine. They do not meaningfully interact with the error message. One of the aims of our enhanced errors is to make students actually read and pay attention to the error messages.

One major reason students do not actually read the error messages is that they find them vague and confusing. This problem can be tackled using the cognitive load theory. Reducing the cognitive load generated by error messages may make it easier for students to interpret and understand error messages. Therefore, the enhancement of showing only one error message at a time should make it easier for students to understand the error, as it aims to reduce the cognitive load.

In the survey, we ask students to rate a total of 5 suggested changes that aim to reduce the cognitive load produced by error messages. Most of these are given low or

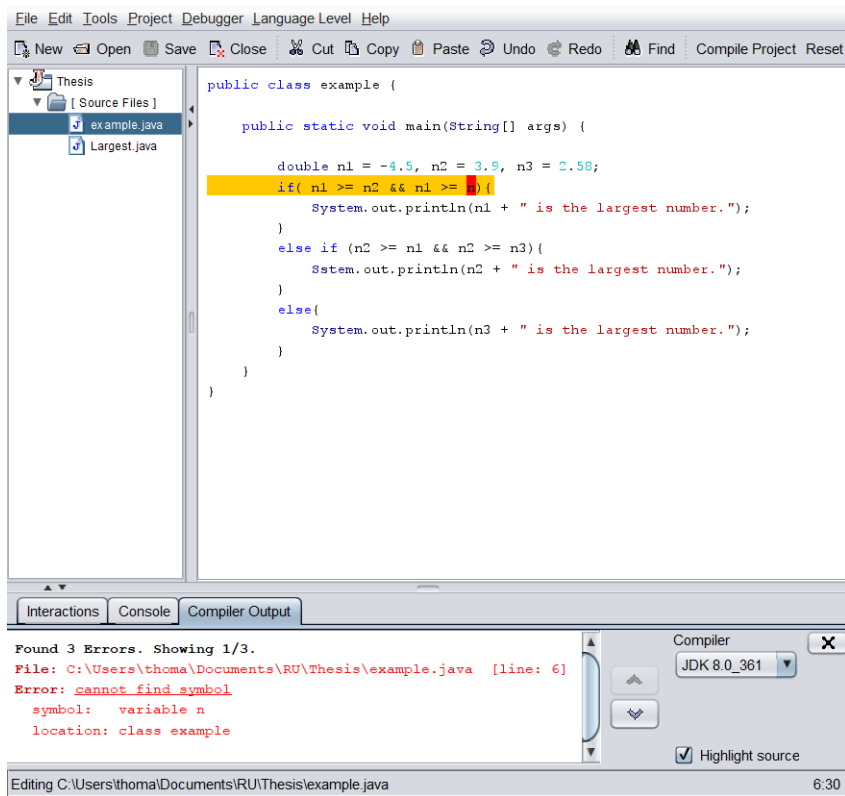


Figure 4.1: A screenshot showing our enhanced error messages

middling ratings. However, one of them, showing only one error message at a time, has a lot of support from students.

As you can see in Figure 4.1 the console now only displays a single error at a time. You can switch between the errors by pressing the up and down buttons to the right of the console. Switching to a different error also highlights this new error. The first line of the console tells you how many errors there are in the program, and shows you which error you are currently focused on.

Additionally, we have made some changes to the styling of the error messages to make students more likely to meaningfully interact with them. The beginning of each line is bolded and has been given a darker color, which makes it easier for students to differentiate between the line that tells you the error location and the rest of the error message. On top of this, the part of the message that explains what the error is is underlined, which immediately draws the user's attention.

Finally, the survey found that many students use a roundabout way of manually debugging, instead of using the built-in debugger that is already present in practically every IDE. To help solve this, we have implemented a feature that is meant to point the user to the built-in debugger.

When the program sees that the user gets the same error more than 3 times in a row, it shows a popup reminding the programmer to use the debugger. We chose to only show this message after 3 consecutive errors, as it is a sign that the user is either stuck on the error or is trying to manually debug.

## Chapter 5

# Results

This section will describe the results obtained in the experiment described in section 2.2. The experiment was performed on 3 first-year computing science students. Two of these students managed to complete the entire assignment. The third student got through most of the assignment, but got stuck on one particular error.

In the experimental error messages, we included 4 different enhancements. First, we have the program show only one error message at a time. This feature saw heavy use during the experiment. We often saw that, when students got stuck at a certain error, they used the feature to move on to the next error, thereby splitting the large problem into several smaller ones. This change seemed to be well received, with one participant specifically saying “*This switching around is quite handy*”.

The main goal of this feature was to reduce the cognitive load produced by the program. The effects of the enhancement on cognitive load can be seen in the experiment. On a few occasions, the participants mention how it helped them focus on the problem. In particular, during the following interviews, one participant stated that the enhanced error messages felt “*better than normal*” when solving errors. When asked why, they answered “*I think maybe that you only show one error lets me focus more on that error*”. Another participant said that “*it really helped me focus on the errors and not get [...] distracted*”

On top of this, all of the participants said that splitting the assignment into smaller tasks stopped them from feeling overwhelmed by the large number of errors. One of them also mentioned that “*it is easier to solve the problems separately*”.

However, one of the students said that the setup was not very intuitive. Because it only showed one error, it looked that they were closer to completing the assignment than they actually were, and they found the navigation to be somewhat confusing. However, they did admit that this problem could be helped with more exposure to the setup, as you would naturally become used to only viewing one error at a once if you worked with it for a longer period of time. When asking students to rate the enhancement, they gave it an average rating of 8.

The second enhancement made to the experimental error messages was in the highlight-

ing. In the experiment, both the line and the token that caused the error are highlighted in different colors.

During the experiment, we saw that the participants used the highlighting to find the exact location of the errors. On a few occasions, the participants explicitly mentioned that the highlighting helped them with this. This can be seen in quotes said during the experiment such as “*that probably means that this static is wrong. You can tell by the highlighting.*” or “*it [the highlighting] really helped in finding the error*”.

The importance of highlighting could also be seen in cases where it was absent. On a specific point in the assignment, one of the participants turned the highlighting off because its color clashed with the keyword highlighting. After they had finished this section and moved on to the next error, they initially forgot to turn the highlighting back on. When trying to fix the next error they said “*where [has] the error gone?*”. They could only find it again after turning the highlighting back on. This indicates the importance of the highlighting when locating errors.

During the interviews conducted after finishing the experiment, participants praised the feature. One said that “*Especiallly the highlighting was very clear, you could quickly see where the error was*”, while another mentioned that “*it showed exactly where the error was*”. Overall, this feature received a rating of 8, and one of the participants explicitly said that it was better than normal.

The third change we made to the experimental error messages was in the styling. We changed the color, bolded the start of each sentence, and underlined the most important part of the error message. This was meant to have two effects. First, it was supposed to encourage the user to read the error messages.

The effects of the enhancements on reading error messages can be seen during the experiment. We see a lot of cases where participants said “*it says illegal start of expression*” or “*so it is Illegal start of expression*”. In fact, all participants consistently read the error message out loud before working on it.

This can also be seen in cases of repeated errors. For example, participants said things like “*that is the same problem*”, showing that they recognize the error and remember seeing it before. They were then able to use this information to quickly solve the repeated error.

The second goal of this enhancement was to prevent the number of times participants use the internet. During the experiment, the three subjects used the internet a combined 5 times. These internet searches can be split up into two categories. First, all participants used the internet to find out how to define a scanner in Java. Second, two of the participants used the Internet to clarify how some of the methods used in the assignment worked. One of them searched “*Methods for list*”, while another looked up “*parseDouble output*”.

During the subsequent interviews, the participants mentioned that especially the underlining of important parts of the error message helped. One of them said “*it is useful, because you are immediately focused on the problem. The rest is only additional information*”, while another said “*I think the underlining because it draws your attention*”.



*to the most important parts*". This student also mentioned that the styling "*makes the error stand out and it grabs your attention*". The final student however said that "*it was better than normal, but I don't think it had a big effect*". Overall, the participants gave the styling an average rating of 7,5. They also said that, of the changes made to the styling, the underlining was the most helpful. This is because it draws attention to the most important part of the error message.

The fourth and final enhancement present in the experimental error messages was that it points the user towards the debugger when it detects they are stuck on a certain error. All three of the participants initially reacted to this enhancement with confusion. After further questioning, it was revealed that they never learned what a debugger was and that they did not know how to use it.

In the interviews conducted after the experiment, we asked the participants to rate how effective they thought the error messages were in locating, understanding, and solving the errors respectively. The effectiveness in locating the error message received an average rating of 9. All three participants mentioned how especially the highlighting was very helpful in locating the error messages. The understanding received a rating of 7.6, but two of the participants mentioned that they thought that the error messages were too vague at times. Finally, the effectiveness of solving the error messages received an average rating of 7.3. Again, participants noted that sometimes the error messages were too vague or unclear to be able to solve them quickly.

When asked about problems or shortcomings of the enhanced error messages as presented in the experiment, two of the students still mentioned that the error messages were not always clear or easy to understand. There were also some mentions of implementation issues. One participant mentioned that the highlighting seemed to be off at times, and all students noticed that the color of the highlighting sometimes clashed with the keyword highlighting.

There were also some mentions of things that could have been implemented differently. One of the students said that it would be better if the highlighting was automatically turned off while typing, as it was distracting. Another participant suggested making the highlight color different for different categories of errors, to allow the user to quickly differentiate between them. Finally, one student suggested creating keyboard shortcuts to toggle highlighting and switch between errors.

Besides these smaller quality-of-life improvements, two other large-scale design changes were suggested. First, one student mentioned linking to documentation in the error message. Second, someone suggested adding examples to the error message.

## Chapter 6

# Conclusion and Discussion

In this section, We first present our main findings based on the results presented earlier. We then discuss limitations in this research and potential future research directions. Finally, the chapter will end with a conclusion.

### 6.1 Findings

Overall, It seems that the changes made to the error messages had a positive effect on students. In the end, this resulted in 2 of the 3 participants being able to complete the assignment within the time limit. The third participant got stuck on one particular error but handled the other ones with relative ease.

During the experiment, we asked students to rate the effectiveness of the enhanced error messages in helping them locate, understand and solve the errors. In the survey conducted earlier, students were asked to rate current standard error messages in these same categories. A comparison of these two results can be found in Figure 6.1.

As you can see, the enhanced error messages received higher ratings across the board. There was one student who both filled in the first survey and participated in the experiment. When comparing the ratings this student gave to both the standard en enhanced error messages, we see that they gave the same rating for their effectiveness in locating the error, while they rated the enhanced error messages higher in the other two categories. While this data seems to indicate that the enhanced error messages are an improvement upon standard error messages, it is important to keep in mind that we only had 3 participants for the experiment and that the data is therefore not conclusive. For the rest of this section, we will be providing more qualitative evidence to back up this claim.

In the experimental error messages, we made four major changes. For three of the four changes made, we can immediately see that they had their intended effect.

For the feature where the program only displays one error message at a time, its main goal was to reduce the cognitive load produced by the program. As you can see in the results, the participants mentioned that it helped them focus, with one of them even specifically mentioning that it felt better than normal.

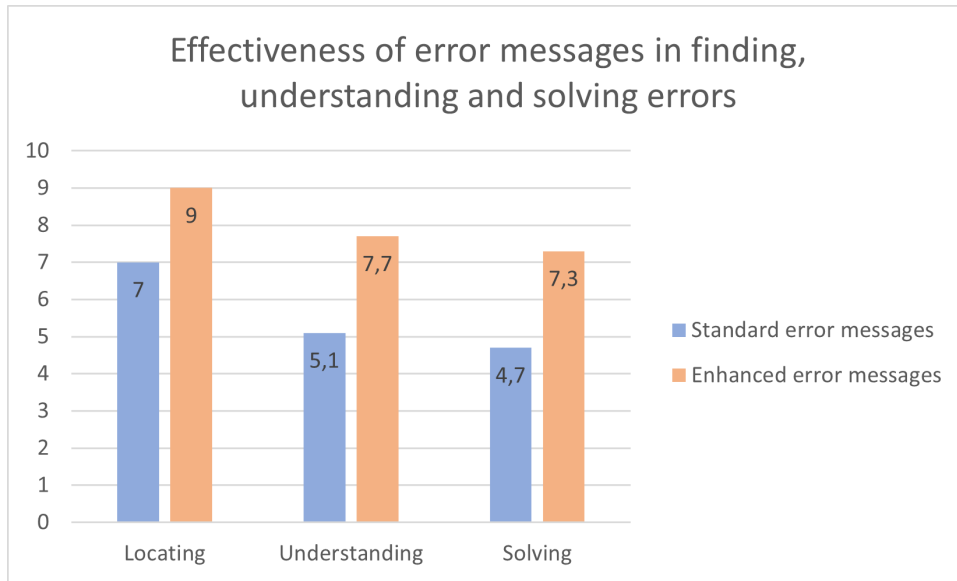


Figure 6.1: Comparison of the effectiveness ratings given to standard and enhanced error messages

The idea was that, by reducing the cognitive load produced, students would be better able to understand the error messages, and thereby have an easier time solving them. This understanding of error messages was found to be a big problem for students in earlier studies [4, 22]. The results indicate that the enhanced error messages performed better in these aspects than standard error messages, but there were still problems. While participants did mention that it felt better than normal, they still found that the error messages were unclear or vague at times.

Besides its small effects on cognitive load, participants also found that the feature helped them divide the assignment into smaller sub-problems, which made it easier to solve and prevented them from being overwhelmed.

For the highlighting, we again see that the participants found them to be very useful. In the first survey, students indicated that locating the error was one of the most important jobs of the error messages, which is made easier by this highlighting. As shown in the results, the participants made heavy use of the highlighting when locating errors. This was also confirmed in the subsequent interviews, where students again praised the highlighting for making it easy to locate the error.

The goal of the updated styling was twofold. First, its aim was to make participants read the error messages. During the initial survey, we found that many students don't actually read the error message, and instead only use it to find the error's location. The idea behind the enhancement was to draw the user's attention toward the error message by underlining the most important part.

As you can see in the results, the enhancements succeeded in accomplishing this goal. All three of the participants consistently read the error messages when solving the

errors. It also became clear that the students actively remembered the error messages, as they were able to recognize repeated error messages. They were also able to use the knowledge gained while working on the first instance of the error to quickly solve the repeated error.

The second goal of this enhancement was to reduce the users' reliance on the internet. During the survey, we found that many students simply copied the error message into an internet search engine, again without reading the error message. As shown in the results section, we counted a combined total of 5 internet searches during the experiments. Notably, none of these searches were cases where the user copied the entire error message. This is a noticeable improvement upon the behavior described by students in the first survey

The final enhancement made to the error messages, pointing to the debugger when the program detects the user is stuck on a certain error, did not work as well as the others. This was due to the fact that the students had never learned how to use a debugger, and could therefore not really make good use of the feature.

During the first survey, we found that students rarely made use of the debugger, instead using a method we referred to as "manual debugging". At the time, we assumed that this was because they forgot about it during programming, which we aimed to fix by suggesting they use the debugger when stuck at an error. However, it has now become clear that this lack of usage was instead caused by a gap in their education. Pointing to the debugger in the error message might still be helpful, as it encourages the user to learn about the debugger. However, we did not have enough time for this during the experiment, so participants were told to ignore it.

While the participants were generally positive about the enhanced error messages, they did still find some shortcomings. As mentioned before, they still found that the error messages were not always clear or easy to understand. This was already a common complaint with standard error messages. In the enhanced error messages, we tried to make the errors easier to understand using cognitive load, but it seems like this was not enough.

As mentioned in the results, there were also some mentions of things that could be implemented better. These were smaller quality-of-life improvements that would be nice to have, but their absence does not really take away from the overall quality of the error messages.

Finally, there were two larger improvements that could be made to the error messages mentioned in the interviews. First, one student mentioned linking to documentation in the error message. This feature was included in the first survey but was not implemented in the enhanced error messages because it only received middling support. Second, someone suggested adding examples to the error message, which is a change in content and therefore out of scope for this study.

## 6.2 limitations

For our implementation of the enhanced error messages, we chose to use the DrJava IDE[7] as a base. This decision was made due to time constraints, as implementing an IDE from scratch is extremely time-consuming. While we deliberately chose to use a relatively lightweight and barebones IDE as a base, it still contained some features that could have influenced the participants' opinions of the enhanced error messages

There were more outside factors that might have affected the results of the experiment. The fact that participants were aware of the fact that we were investigating error messages, combined with the fact that the researcher was on a call with the participant while completing the assignment, might have influenced their behavior. For example, it could have made them pay more attention to the error messages than they normally would.

## 6.3 future work

While the findings seem to indicate that the enhancements improved participants performance, it is important to keep in mind the experiment only has a small number of participants. A larger-scale study, similar to those performed by Becker [3] or Pettit et al[17], is needed to confirm these conclusions. This is left for further research.

It is also important to note that, while the results look promising, it also looks like it is not enough to solve all problems students have with error messages. During the experiment, participants still said that they found the error messages to be difficult to understand at times. It seems like a combination of both content and presentation enhancements is needed to fully solve this problem. Studying the effects of the presentational enhancements presented here combined with content enhancements is also left as future work.

In this study, we have chosen to investigate only four different enhancements in depth. Some of the enhancements that were mentioned during this thesis, but that were not included in the final product, could also prove to be useful changes. In particular, the "Linking to documentation" and "Placing the error message close to the error" seem to be good candidates for further investigation. Both of these received middling support in the survey, but saw high variance in their ratings. the "Linking to documentation" enhancement was also mentioned as a potential improvement by a student during the experiment. Researching the effects of these enhancements is also left for future research.

## 6.4 conclusion

First-year computing science students often struggle with programming. One of the biggest reasons for this is unclear and frustrating error messages. In this thesis, we aimed to find out how presentational enhancements could be used to improve compiler error messages. In the end, we have identified 3 enhancements that seemed to have a positive effect on users.

First, we found that highlighting the line and token that caused the error will help the user to quickly and effectively locate the mistake. Second, we found that updating the styling of the error messages can encourage users to actively read the error messages. Finally, we found that only showing one error at a time can help the user focus and prevent them from being overwhelmed by large numbers of errors.

# Bibliography

- [1] Titus Barik. “Error Messages as Rational Reconstructions.” In: (2018-03-29). URL: <http://www.lib.ncsu.edu/resolver/1840.20/35439>.
- [2] Titus Barik et al. “Do Developers Read Compiler Error Messages?” In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 575–585. DOI: 10.1109/ICSE.2017.59.
- [3] Brett A. Becker. “An Effective Approach to Enhancing Compiler Error Messages”. In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE ’16. Memphis, Tennessee, USA: Association for Computing Machinery, 2016, pp. 126–131. ISBN: 9781450336857. DOI: 10.1145/2839509.2844584. URL: <https://doi.org/10.1145/2839509.2844584>.
- [4] Brett A. Becker et al. “Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research”. In: *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. ITiCSE-WGR ’19. Aberdeen, Scotland Uk: Association for Computing Machinery, 2019, pp. 177–210. ISBN: 9781450375672. DOI: 10.1145/3344429.3372508. URL: <https://doi.org/10.1145/3344429.3372508>.
- [5] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. “Enhancing Syntax Error Messages Appears Ineffectual”. In: *Proceedings of the 2014 Conference on Innovation Technology in Computer Science Education*. ITiCSE ’14. Uppsala, Sweden: Association for Computing Machinery, 2014, pp. 273–278. ISBN: 9781450328333. DOI: 10.1145/2591708.2591748. URL: <https://doi.org/10.1145/2591708.2591748>.
- [6] Paul Denny et al. “CodeWrite: Supporting Student-Driven Practice of Java”. In: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’11. Dallas, TX, USA: Association for Computing Machinery, 2011, pp. 471–476. ISBN: 9781450305006. DOI: 10.1145/1953163.1953299. URL: <https://doi.org/10.1145/1953163.1953299>.
- [7] Robert Cartwright Eric Allen and Brian Stoler. “DrJava: A lightweight pedagogic environment for Java”. In: (2001).
- [8] Dianne Hagan and Selby Markham. “Teaching Java with the BlueJ environment”. In: *Proceedings of Australasian Society for Computers in Learning in Tertiary Education Conference ASCILITE*. 2000.

- [9] Maria Hristova et al. “Identifying and Correcting Java Programming Errors for Introductory Computer Science Students”. In: *SIGCSE Bull.* 35.1 (Jan. 2003), pp. 153–156. ISSN: 0097-8418. DOI: 10.1145/792548.611956. URL: <https://doi.org/10.1145/792548.611956>.
- [10] C. D. Hundhausen, D. M. Olivares, and A. S. Carter. “IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda”. In: *ACM Trans. Comput. Educ.* 17.3 (Aug. 2017). DOI: 10.1145/3105759. URL: <https://doi.org/10.1145/3105759>.
- [11] J. Jackson, M. Cobb, and C. Carver. “Identifying Top Java Errors for Novice Programmers”. In: *Proceedings Frontiers in Education 35th Annual Conference*. 2005, T4C–T4C. DOI: 10.1109/FIE.2005.1611967.
- [12] Harsha B. M. Kadekar, Sohuni Sohoni, and Scotty D. Craig. “Effects of Error Messages on Students’ Ability to Understand and Fix Programming Errors”. In: *2018 IEEE Frontiers in Education Conference (FIE)*. 2018, pp. 1–8. DOI: 10.1109/FIE.2018.8658629.
- [13] Michael Kölling et al. “The BlueJ System and its Pedagogy”. In: *Computer Science Education* 13.4 (2003), pp. 249–268. DOI: 10.1076/csed.13.4.249.17496. eprint: <https://doi.org/10.1076/csed.13.4.249.17496>. URL: <https://doi.org/10.1076/csed.13.4.249.17496>.
- [14] Ard W. Lazonder and Hans van der Meij. “Error-information in tutorial documentation: Supporting users’ errors to facilitate initial skill learning”. In: *International Journal of Human-Computer Studies* 42.2 (1995), pp. 185–206. ISSN: 1071-5819. DOI: <https://doi.org/10.1006/ijhc.1995.1009>. URL: <https://www.sciencedirect.com/science/article/pii/S1071581985710099>.
- [15] Michael McCracken et al. “A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students”. In: *SIGCSE Bull.* 33.4 (Dec. 2001), pp. 125–180. ISSN: 0097-8418. DOI: 10.1145/572139.572181. URL: <https://doi.org/10.1145/572139.572181>.
- [16] Marie-Hélène Nienaltowski, Michela Pedroni, and Bertrand Meyer. “Compiler Error Messages: What Can Help Novices?” In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’08. Portland, OR, USA: Association for Computing Machinery, 2008, pp. 168–172. ISBN: 9781595937995. DOI: 10.1145/1352135.1352192. URL: <https://doi.org/10.1145/1352135.1352192>.
- [17] Raymond S. Pettit, John Homer, and Roger Gee. “Do Enhanced Compiler Error Messages Help Students? Results Inconclusive.” In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’17. Seattle, Washington, USA: Association for Computing Machinery, 2017, pp. 465–470. ISBN: 9781450346986. DOI: 10.1145/3017680.3017768. URL: <https://doi.org/10.1145/3017680.3017768>.



- [18] James Prather et al. “On Novices’ Interaction with Compiler Error Messages: A Human Factors Approach”. In: *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ICER ’17. Tacoma, Washington, USA: Association for Computing Machinery, 2017, pp. 74–82. ISBN: 9781450349680. DOI: 10.1145/3105726.3106169. URL: <https://doi.org/10.1145/3105726.3106169>.
- [19] Erik Sandewall. “Programming in an Interactive Environment: The “Lisp” Experience”. In: *ACM Comput. Surv.* 10.1 (Mar. 1978), pp. 35–71. ISSN: 0360-0300. DOI: 10.1145/356715.356719. URL: <https://doi.org/10.1145/356715.356719>.
- [20] John Sweller. “Cognitive load during problem solving: Effects on learning”. In: *Cognitive Science* 12.2 (1988), pp. 257–285. ISSN: 0364-0213. DOI: [https://doi.org/10.1016/0364-0213\(88\)90023-7](https://doi.org/10.1016/0364-0213(88)90023-7). URL: <https://www.sciencedirect.com/science/article/pii/0364021388900237>.
- [21] E. Thiselton and C. Treude. “Enhancing Python Compiler Error Messages via Stack”. In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Los Alamitos, CA, USA: IEEE Computer Society, Sept. 2019, pp. 1–12. DOI: 10.1109/ESEM.2019.8870155. URL: <https://doi.ieeecomputersociety.org/10.1109/ESEM.2019.8870155>.
- [22] V. Javier Traver. “On Compiler Error Messages: What They Say and What They Mean”. In: *Advances in Human-Computer Interaction 2010* (Jan. 2010). DOI: 10.1155/2010/602570.
- [23] Iris Vessey. “Expertise in debugging computer programs: A process analysis”. In: *International Journal of Man-Machine Studies* 23.5 (1985), pp. 459–494. ISSN: 0020-7373. DOI: [https://doi.org/10.1016/S0020-7373\(85\)80054-7](https://doi.org/10.1016/S0020-7373(85)80054-7). URL: <https://www.sciencedirect.com/science/article/pii/S0020737385800547>.
- [24] Iris Vessey and Dennis Galletta. “Cognitive Fit: An Empirical Study of Information Acquisition”. In: *Information Systems Research* 2.1 (1991), pp. 63–84. ISSN: 10477047, 15265536. URL: <http://www.jstor.org/stable/23010613> (visited on 02/13/2023).
- [25] Christopher Watson, Frederick W. B. Li, and Jamie L. Godwin. “BlueFix: Using Crowd-Sourced Feedback to Support Programming Students in Error Diagnosis and Repair”. In: *Advances in Web-Based Learning - ICWL 2012*. Ed. by Elvira Popescu et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 228–239. ISBN: 978-3-642-33642-3.

# Appendix A

## Survey

This appendix will list the questions asked in the survey

**1.) Please describe the process you go through when trying to debug an error**

**2.) How effective do you think current error messages are in helping you identify the location of the erroneous code?**

This question asked students to give a rating between 1 and 10

**3.) How effective do you think current error messages are in helping you understand what is going wrong?**

This question asked students to give a rating between 1 and 10

**4.) How effective do you think current error messages are in helping fix the error?**

This question asked students to give a rating between 1 and 10

**5.) What aspects of error messages do you find the most helpful? Please explain your answer.**

**6.) What aspects of error messages do you find the least helpful? Please explain your answer.**

Below, we show a program that finds the largest of 3 numbers. The code contains 2 errors. In the following questions, we will show different ways of presenting the resulting error messages. Please rate each of them on a scale from 1 to 10.

## 7.) Base Message

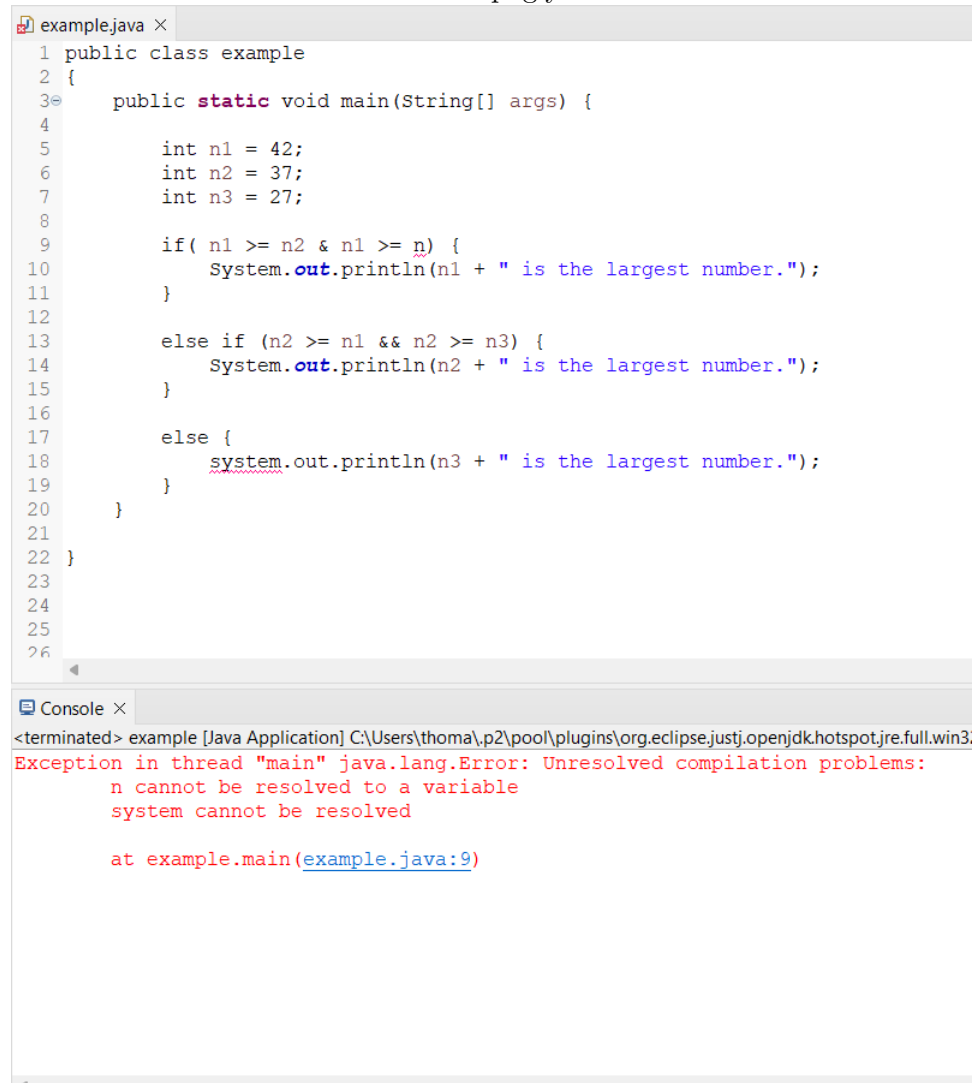
```
example.java ×
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10            System.out.println(n1 + " is the largest number.");
11        }
12
13        else if (n2 >= n1 && n2 >= n3) {
14            System.out.println(n2 + " is the largest number.");
15        }
16
17        else {
18            system.out.println(n3 + " is the largest number.");
19        }
20    }
21 }
22
23
24
25
26
```

```
Console ×
<terminated> example [Java Application] C:\Users\thoma\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    n cannot be resolved to a variable
    system cannot be resolved

    at example.main(example.java:9)
```

## 8.) Error Underlined

We underline the error with red squigly lines.



```
example.java ×
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10            System.out.println(n1 + " is the largest number.");
11        }
12
13        else if (n2 >= n1 && n2 >= n3) {
14            System.out.println(n2 + " is the largest number.");
15        }
16
17        else {
18            system.out.println(n3 + " is the largest number.");
19        }
20    }
21 }
22 }
23
24
25
26
```

```
Console ×
<terminated> example [Java Application] C:\Users\thoma\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    n cannot be resolved to a variable
    system cannot be resolved
    at example.main(example.java:9)
```

## 9.) Error Highlighted

We highlight both the error and the line in which the error occurs

```
example.java ×
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10            System.out.println(n1 + " is the largest number.");
11        }
12
13        else if (n2 >= n1 && n2 >= n3) {
14            System.out.println(n2 + " is the largest number.");
15        }
16
17        else {
18            system.out.println(n3 + " is the largest number.");
19        }
20    }
21 }
22 }
23
24
25
26
```

```
Console ×
<terminated> example [Java Application] C:\Users\thoma\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    n cannot be resolved to a variable
    system cannot be resolved

    at example.main(example.java:9)
```

## 10.) One error at a time

We only show one error message at a time.

```
example.java ×
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10            System.out.println(n1 + " is the largest number.");
11        }
12
13        else if (n2 >= n1 && n2 >= n3) {
14            System.out.println(n2 + " is the largest number.");
15        }
16
17        else {
18            system.out.println(n3 + " is the largest number.");
19        }
20    }
21 }
22 }
23
24
25
26
```

```
Console ×
<terminated> example [Java Application] C:\Users\thoma\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    n cannot be resolved to a variable
    at example.main(example.java:9) | error 1/2 >>
```

## 11.) Message close to errors

We place the error message right next to the error

```
examplejava x
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10            System.out.println(n1 + " is the largest number.");
11            n cannot be resolved to a variable
12        }
13        else if (n2 >= n1 && n2 >= n3) {
14            System.out.println(n2 + " is the largest number.");
15        }
16        else {
17            system.out.println(n3 + " is the largest number.");
18            system cannot be resolved
19        }
20    }
21 }
22 }
23 }
24 }
25 }
26 }
```

## 12.) Matching fonts

We match the style and font of the error message with the style of the code

```
example.java ×
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10             System.out.println(n1 + " is the largest number.");
11         }
12
13         else if (n2 >= n1 && n2 >= n3) {
14             System.out.println(n2 + " is the largest number.");
15         }
16
17         else {
18             system.out.println(n3 + " is the largest number.");
19         }
20     }
21 }
22 }
23
24
25
26

Console ×
<terminated> example [Java Application] C:\Users\thoma.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    n cannot be resolved to a variable
    system cannot be resolved

    at example.main(example.java:9)
```




### 13.) Read the error out loud

We provide a button that makes the program read the error out loud

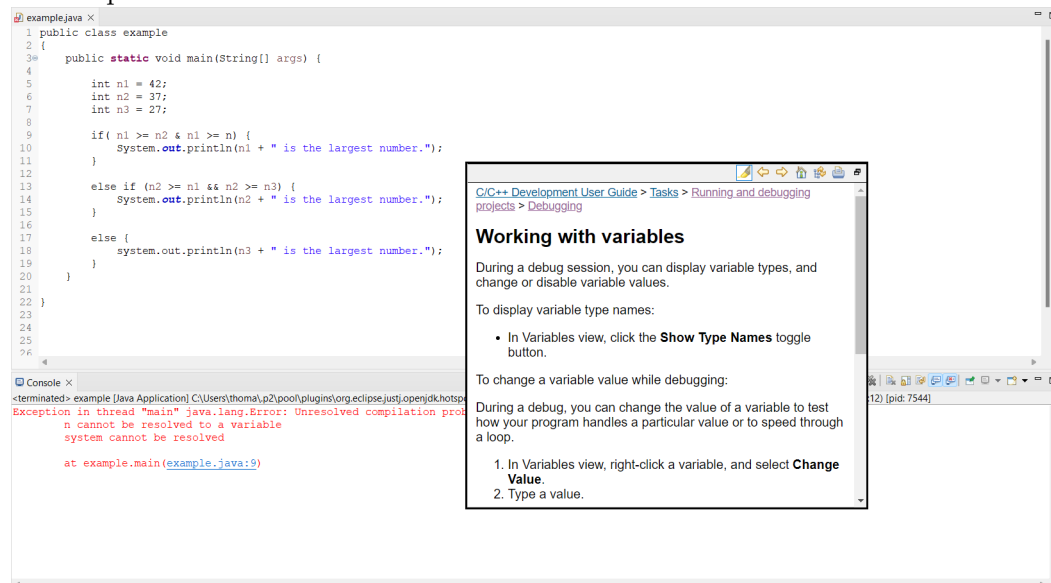
```
examplejava ×
1 public class example
2 {
3     public static void main(String[] args) {
4
5         int n1 = 42;
6         int n2 = 37;
7         int n3 = 27;
8
9         if( n1 >= n2 & n1 >= n) {
10            System.out.println(n1 + " is the largest number.");
11        }
12
13        else if (n2 >= n1 && n2 >= n3) {
14            System.out.println(n2 + " is the largest number.");
15        }
16
17        else {
18            system.out.println(n3 + " is the largest number.");
19        }
20    }
21 }
22 }
23
24
25
26
```

```
Console ×
<terminated> example [Java Application] C:\Users\thoma\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win3
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    n cannot be resolved to a variable
    system cannot be resolved
    at example.main(example.java:9)
```



#### 14.) Link to documentation

We provide a link to the relevant documentation



#### 15.) Please explain your reasoning for why you liked the improvement that you rated the highest

please also mention what change you rated the highest in your answer

#### 16.) Please explain your reasoning for why you disliked the improvement that you gave the lowest rating

please also mention what error you rated the lowest in your answer

#### 17.) Do you have any other suggestions on how error messages can be improved?

# Appendix B

## Assignment

This appendix will show the code of the programming assignment used in the evaluation of the error messages

### B.0.1 Class Main

```
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4
5 class Main {
6     public static void main(String[] args) {
7
8
9         // Read rectangles from user input
10        //Syntax Error 1: Define static variable inside static method
11        static List<Rectangle> rectangles = readRectanglesFromInput();
12
13        System.out.println("Before sorting:");
14        printRectangles(rectangles);
15
16        // Sort the rectangles by area from smallest to largest
17        sortRectanglesByArea(rectangles);
18
19        //Syntax Error 2: removed ;
20        System.out.println("\nAfter sorting:");
21        printRectangles(rectangles);
22    }
23
24    public static void printRectangles(List<Rectangle> rectangles) {
25        //Compiler Error 3: Added ; after updation
26        for (int i = 0; i<rectangles.size(); i++) {
27            System.out.println("Width: " + rectangles.get(i).getWidth() + ", Height: " +
28                ↪ rectangles.get(i).getHeight() + ", Area: " +
29                ↪ rectangles.get(i).getArea());
30        }
31    }
32    //Compiler Error 4: removed return type
```

```

31     public static readRectanglesFromInput() {
32         List<Rectangle> rectangles = new ArrayList<>();
33         System.out.println("Enter rectangles (format: width, height), one per line
34         ↪ (Enter 'q' to finish):");
35         while (true) {
36             //Compiler Error 5: Use scanner without defining it
37             String input = scanner.nextLine().trim();
38             //Compiler Error 6: define q as a character instead of a string
39             if (input.equalsIgnoreCase('q')) {
40                 break;
41             }
42             try {
43                 String[] dimensions = input.split(",");
44                 double width = Double.parseDouble(dimensions[0].trim());
45                 double height = Double.parseDouble(dimensions[1].trim());
46                 Rectangle rectangle = new Rectangle(width, height);
47                 rectangles.add(rectangle);
48             } catch (Exception e) {
49                 System.out.println("Invalid input format. Please try again.");
50             }
51         }
52
53         return rectangles;
54     }
55
56     public static void sortRectanglesByArea(List<Rectangle> rectangles) {
57         //Compiler Error 7: Define static variable inside static method
58         static int n = rectangles.size();
59         for (int i = 0; i < n - 1; i++) {
60             for (int j = 0; j < n - i - 1; j++) {
61                 if (rectangles.get(j).getArea() > rectangles.get(j + 1).getArea()) {
62                     Rectangle temp = rectangles.get(j);
63                     rectangles.set(j, rectangles.get(j + 1));
64                     rectangles.set(j + 1, temp);
65                 }
66             }
67         }
68     }
69 }

```

## B.0.2 Class Rectangle

```

1 //Compiler Error 8: removed { in class declaration
2 public class Rectangle
3     private double width;
4     private double height;
5
6 //Compiler Error 9: Constructor takes int instead of double
7 public Rectangle(int width, int height) {
8     this.width = width;
9     this.height = height;
10 }

```

```
11
12     public double getArea() {
13         //Compiler Error 10: removed return statement
14         width * height;
15     }
16
17     public double getWidth() {
18         return width;
19     }
20
21     public double getHeight() {
22         return height;
23     }
24 }
```

# Appendix C

## Evaluation Interview

This appendix will list the questions asked in the evaluation interview held with students after they have completed the assignment using the enhanced error messages

**1.) How effective do you think the enhanced error messages were in helping you identify the location of the erroneous code?**

This question asked students to give a rating between 1 and 10

**2.) How effective do you think think the enhanced error messages were are in helping you understand what is going wrong?**

This question asked students to give a rating between 1 and 10

**3.) How effective do you think think the enhanced error messages were in helping you solve the error?**

This question asked students to give a rating between 1 and 10

The next 4 questions will describe 4 enhancements made to the error messages of the IDE you just used. Please give me your opinion on each of them.

**4.) Highlighted the error**

In the enhanced error messages, the line of the error was highlighted. Additionally, the token that caused the error is highlighted in red.

**5.) One error at a time**

In the enhanced error messages, we only showed one error message at a time.

**6.) Error Styling**

In the enhancements, we changed the styling of the error messages. We bolded the beginning and underlined the most important part of the message.

**7.) Pointing to the debugger**

In the enhanced error messages, we count how often you repeat the same error. If we find that you are stuck on an error(get the same message 3+ times in a row), we point the user toward the built-in debugger

**8.) What do you think were the biggest weak spots of the enhanced error messages**

**9.) Do you think that anything was missing in the enhanced error messages**