UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Project
Final Paper 2022

Title: **Improving Text-Based Java Programming Error Messages for South African Students**

Author: **Danny Guttmann**

Project Abbreviation: **IMPROVEDERRMSGS**

Supervisor(s): **Dr Zola Mahlaza**

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	0
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	20
System Development and Implementation	*0*	*20*	10
Results, Findings and Conclusions	*10*	*20*	15
Aim Formulation and Background Work	*10*	*15*	15
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	**80**

Improving Text-Based Java Programming Error Messages For South African Students

Danny Guttmann
gttdan002@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Programming error messages are an important source of feedback for programmers. Due to their lack of detail and use of complicated jargon, many novices do not find them useful, and ultimately find programming inaccessible. To make Java programming more accessible to South African students, Java error messages were improved by applying guidelines for writing effective error messages from previous research in this area, and then translated into Afrikaans, the most commonly spoken language in the Western Cape province. First, a sample of popular error messages were obtained from the open source BlueJ Blackbox Project database. Then, error message templates were derived. These templates were shown to tutors with experience teaching Java programming. The tutors were asked to improve the templates with guidelines established from previous research in this area. Then software was written to convert the standard error messages into the improved messages. To evaluate the efficacy of the improved error messages, programming tasks were developed with intentional errors. First year computer science students were invited to participate in a lab session, in which some of the students were given the improved error messages (some in English and others in Afrikaans), and others the standard error messages. Two should students participated. A second evaluation was completed online, in which first year students were sent a link to an online form containing erroneous Java code and the error message given when the program was run and were asked to remove the errors from the code. The students with the improved error messages did not perform better on the programming tasks than the students with the standard error messages, but due to the low number of participants, further evaluation is needed to establish a meaningful conclusion.

KEYWORDS

Compilers, Error Messages, Novice Programmers, Enhanced Error Messages

1 INTRODUCTION

Computers understand *machine code*, a binary language which comprises ones and zeros. As machine code is a difficult language for humans to understand, many programmers write in high-level languages, which are closer to natural languages (like English). Due to this practice, there is a need for *compilers*, a special type of program that reads and translates programs written in high-level languages into equivalents written in low-level languages (like *assembly* languages), which can be assembled into machine code so that computers can run the programs.

Before translating the program, the compiler checks for any mistakes in the code. If it finds any, it alerts the programmer by displaying an error message [4].

These error messages are important to programmers, because without them, they would spend a great deal of time ascertaining the cause of their programs' inability to compile or run. Error messages therefore expedite faster and more efficient programming and also play an important role in educating novice programmers by alerting them to mistakes that they make [6]. Unfortunately, the error messages generated by the compiler are not always helpful to novices, because many error messages lack detail and use complicated terminology. This may be useful for expert programmers, who are more familiar with how the different types of errors are caused, but not for novices who would not have such knowledge [9]. Another difficulty experienced by novice programmers whose first language is not English is that error messages are usually written in English [16]. This is pertinent, particularly for South African students of programming, as English is not the first language of most South Africans [10].

To address this problem, this study considered how to improve the quality and nature of Java text-based compiler error messages so that they are more useful for South African novice programmers.

In this paper, related works will be considered to provide background to this area of research, as well as the motivation for this particular study. Then the methods and materials utilised in this study will be described. This will be followed by a presentation and discussion of the results obtained. Then conclusions will be drawn and suggestions for further research will be proposed.

2 BACKGROUND AND RELATED WORK

An examination of prior research into the area of error message improvement is necessary so as to identify gaps in the research and to establish a new path of investigation. The findings are divided into datasets comprising corpora collection, error message prevalence, error message improvement, and identified limitations.

2.1 Datasets comprising Corpora Collection

Different studies have used various datasets to obtain *corpora collection*. These datasets are discussed below.

The Blackbox Project is a large database that contains data collected from users of BlueJ, a Java development environment that is designed for programming students. The database contains erroneous code and the error messages that were given when the users tried to compile the code [15]. The Blackbox Project is a good source of data because it contains a large set of data from many thousands of users who have agreed to contribute their erroneous code for

research purposes [7]. Studies focused on identifying common errors encountered by novices would find the Blackbox Project to be a useful data source to consider because it provides a large sample of the different error messages encountered by novices. [12, 15].

Prutor is a system that stores each version of code that has been given to the compiler, as well as snapshots of the code at regular intervals. Prutor provides a way to track novices' progress [2]. Prutor is a good choice from which to obtain data for determining ways of enhancing text-based compiler error messages as Prutor is able to provide significant detail as to the nature of error messaging with which novices struggle as well as to the coding habits of novices.

GitHub is another *open-source* (freely available) data source containing the source code of many high-quality projects. The code available from GitHub is useful for training Machine Learning models and creating templates for working code thus enabling improvement in coding [1].

Apart from identify the coding habits of novices and generating improved code, Stack Overflow, a web-based question and answer forum, which allows users to ask and answer any programming related questions, is also a valuable data source. The vast number of questions and answers on Stack Overflow [18], enables the development of a system that queries its database so as to suggest fixes for error messages and to generate more detailed and more specific error messages.

For this paper, the BlueJ Blackbox Project database was chosen to obtain the corpus collection, due to its large supply of Java programming error messages encountered by novices. Of all the potential datasets mentioned above, the BlueJ Blackbox Project was the most appropriate for this paper.

2.2 Error Message Improvement

In order to enhance error messaging, studies either formulated a set of guidelines for the improvement of error messages based on their own or others' research [4], or applied known principles for effective error messaging [3, 13].

Guidelines formulated in these studies included increased readability; reduced cognitive load; provide context; use a positive tone; provide examples; offer hints or solutions [4, 7, 19].

The reviewed literature discussed various ways in which these guidelines have been applied. These included the creation of visual debuggers (such as Decaf [6]) which explain the cause of errors in a visual format [3, 13] or by rewriting the error messages in more natural language [6, 8, 9].

2.3 Identified Limitations

The sections above describe the various models and methodologies utilised in the reviewed literature to improve text-based compiler error messages and how these models and methods have been evaluated. To guide further research in this area, the findings from these models and methodologies as to what worked and what has not worked will be discussed .

Some studies categorise errors made by novices by examining the frequency with which a compiler returned certain error messages [12]. This approach has limitations because different errors can trigger the same message, and different compilers can return different messages relating to an identical error. Some studies have indicated that a better way to categorise errors is by manually examining them and classifying them based on the frequency with which they occur together with the difficulty involved in fixing them. This provides a clearer identification of the error messages create the most difficulties for novice programmers [15].

Another problem relates to taxonomies. Many studies considering the most common error messages encountered by novices form taxonomies to categorise them. Although these taxonomies can be helpful, there is a risk of creating too many categories, which may render the classification process less efficient. Some errors are not easily categorised and may require human intervention to distinguish them from others. Taxonomies may also need to be adjusted as new languages and features are introduced, and their effectiveness may be limited by the accuracy of the tools used to classify them [3].

Although most research has noted that certain errors are more commonly encountered than others,there is disagreement on which errors are most common [14, 15].

Significantly, studies on error frequency often focus on a specific programming language which render their results irrelevant to research which examines other programming languages [14].

Many studies on enhancing error messages conclude that shorter messages yield better results for understandability among novice programmers [3]. The more detailed an error message, the longer programmers will spend reading it and hence the more time they will need to code and debug errors [3].

Many studies admit their results have limited application, because the number of error messages they examine, or the sample size of participants involved is too small [1, 5, 7]. This is particularly true for studies that used the Blackbox Project for their research as the Blackbox Project only contains data from users of BlueJ [7, 12, 15].

3 RESEARCH QUESTIONS

This paper deals exclusively with Java error messages. As an object-orientated programming language, Java will generate error messages through objects known as Exception classes. This is unlike other programming language in which errors are not represented by objects [17]. The implications of this format with regards to enhancing error messages is discussed in detail hereafter.

Based on the background and related works examined, this paper sought to answer two research questions: (1) Will the improved error messages generated from the expertise of programming tutors with experience teaching Java and guidelines obtained previous research, help first year computer science students at UCT debug code faster than standard Java error messages; (2) Will the improved, Afrikaans error messages help Afrikaans speaking first year computer science students at UCT to debug code faster than standard error messages written in English?

4 METHODS AND MATERIALS UTILISED

To answer the research questions, a process of improving text-based Java programming messages was completed in four stages. These stages are described below and depicted visually in Figure 1.

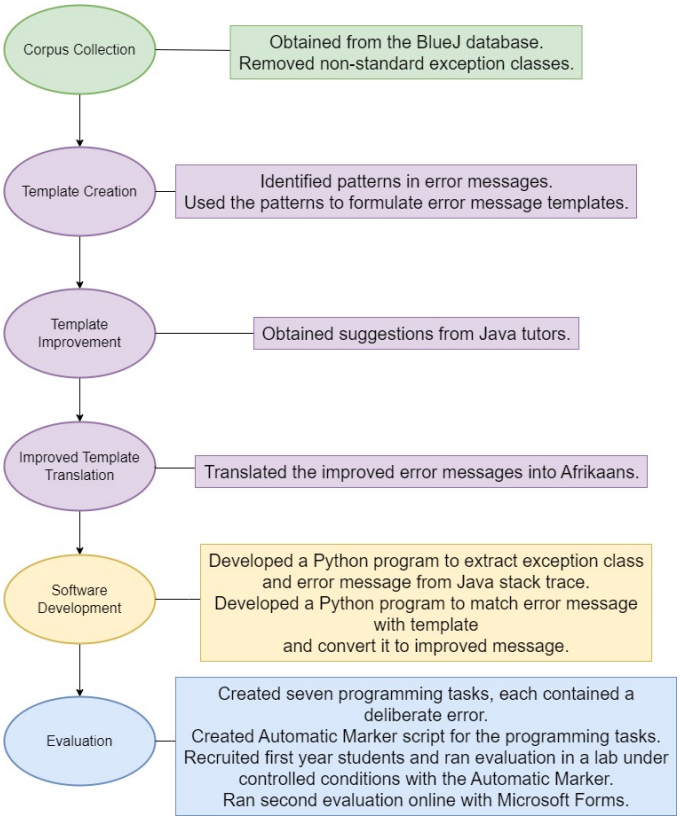

Figure 1: Visualisation of Methodology

4.1 Corpus Collection and Template Creation

As a vast number of Java exception classes exist [17] and given the scope and time limitations of this paper, it was deemed practical to improve only a limited subset of Java error messages .

To obtain this subset, access to the BlueJ Blackbox Project database was requested for research purposes. From the database, a list of the 100 most common exception classes and their associated error messages was extracted. The data was placed in an Excel spreadsheet, and a filtration process was applied. The filtration process involved removing error messages returned by exception classes that are not part of the standard Java library. This excluded 55 programmer-defined exception classes created for specialised purposes [11]. Examples of such excluded exception classes are: *twitter4j.TwitterException*, *IllegalDropZoneException*, *shed.mbed.MBedStateException*.

The remaining 45 exception classes were examined to determine whether the error messages followed specific patterns, and if they did, they were arranged into separate template formats. For example, all error messages associated with the exception class, java.lang.ArrayIndexOutOfBoundsException class, were expressed as: "Index **<int>** out of bounds for length **<int>**" (where the first **<int>** always corresponds to the index of the element number that the program attempted to access, and the second **<int>** corresponds to the length of the array).

Of the 45 exception classes, 25 error message were formulated into templates. The remaining 20 error messages could not be formulated in the same way, because their datasets did not follow a recognised pattern. This can be attributed to these error messages relating to data from different versions of the Java Development Kit or custom written error messages. For example, the error messages in the dataset for the java.util.NoSuchElementException included: "No next element", "There aint nothin left", "Out of list", as well as many others that did not follow a pattern. The Java documentation was therefore used to form error message templates for these exception classes.

4.2 Improvement of Error Messages and Translations

Once the 45 error message templates were formulated, suggestions for improvement of these templates were sought from tutors with experience in teaching Java. Pursuant to this, a survey form was prepared using Microsoft Forms. Each question within the survey included reference to: the name of a Java exception class; a description of what would cause the exception to be raised; and the derived error message template associated with that exception class. Respondents were asked to apply the guidelines, relating to error message improvement derived from the previous research [4, 6, 13] described above to each template. These guidelines included: no jargon, complete sentences, word economy, and simple vocabulary.

To explain to respondents what was expected from their answers, the survey form provided an example of how the template for the StackOverflow error message could be improved. The standard error message returned when this error occurs is is either "stack overflow" or the empty string, "". The example provided, for improving this error message by incorporating the guidelines, was as follows "The program has caused the computer to run out of memory space. This could be due to a method or a loop running too many times". The form was distributed over various WhatsApp groups, used by Computer Science students, as it was known that members of these groups were likely to have Java tutoring experience. Approximately 100 people received the form, all of which were either Computer Science students or graduates. Two responses to the survey were received using this method. One of the respondents had tutored the introductory Java programming course at UCT (CSC1016S) and the other had tutored Java privately. These responses were used to formulate the improved error message templates included in the corpus collection. Seven of the error messages were selected for the evaluation. These seven improved error message templates were then translated into Afrikaans the most commonly spoken languages in Cape Town and the Western Cape Province of South Africa [20] to evaluate whether Afrikaans speaking programmers would respond better to such messages than they do to default error messages typically written in English [16]. The messages were translated by a third-year computer science student at UCT, whose first language is Afrikaans.

4.3 Software developed to improve Standard Error Messages

Thereafter, software was developed to replace the standard Java error messages with the improved and translated messages, so as to present the improved error messages to programming students in a way that is similar to how standard error messages are generally returned to them in a *stack trace*.

A program was written in Python that uses a Java stack trace as input. The program worked to extract and output the name of the exception class and the error message contained in the stack trace, if present.

Another Python program was written that takes two strings as input: the name of a Java exception class, and an error message. The program then matches the error message with one of the error message templates associated with the given exception class. If a match is made, the program extracts specific data from the error message, such as: line numbers; index numbers; file names; method names; class names; or identifier names. These values are assigned to the corresponding improved error message template, which is then returned as output.

Python's **re** module was used to form regular expression representing the improved error message templates. Each derived error message template and the Java stack trace were rewritten as regular expressions in the appropriate **re** format. Regular expressions were used for efficient string manipulation and template matching.

A text file is used to specify whether the output should include the standard error messages or the improved error messages, as well as the language of the error messages (English or Afrikaans) if the improved error messages are selected. The programs access this text file to determine what the output should be.

The program was tested by running it with each of the evaluation programs' (see Evaluation of the Improved Error Messages section below) output.

The diagram depicting the architecture of the system to convert the standard error messages to the improved error messages is depicted in Figure 3.

4.4 Evaluation of the Improved Error Messages

After the software to convert a Java stack trace into an improved error message was developed, a task was designed to evaluate the efficacy of the improved error messages. The task comprised a number of Java programs. Each program had been written with a deliberate error, or had deliberately been written to cause an error when given certain inputs. Seven programs were written, so the task would evaluate seven error messages. The reason for this was that it would not have been practical to evaluate all 45 improved error messages in a controlled setting given the time limitations for this paper. These seven error messages were chosen based on the practically of developing a working Java program that would raise an exception for some input, but for others. These seven error messages and their corresponding templates are displayed in the table in Figure 2. The participants were given a description of what each of the seven programs were meant to do. Most involved a fictional hiking-club, and the programs needed to process data relating to members. One of the programs involved division, and needed to avoid diving by zero.

Therafter, the *Automatic Marker* was chosen to evaluate the efficacy of the improved error messages. The *Automatic Marker* is a program that is integrated into the learning management systems of computer science courses at UCT. It is used to mark programming assignments, by running the programs that students have submitted with different inputs and then comparing the output to an expected output. This is achieved with Unix terminal commands. The Automatic Marker was chosen for this study, as it was relatively quick and easy to configure given the available time. It also keeps timestamps of when programs are submitted, which was beneficial for data collection and analyses purposes. The first year students are familiar with the Automatic Marker.

First year students were recruited, and invited to attend an evaluation session in computer labs at UCT. 2 out of 669 students arrived and both were given the same seven programs to debug. They were not permitted to use an IDE, compile or run the code that they had been given. They were allowed to use a basic text-editor to view and edit the code, and were instructed to submit the code to the Automatic Marker, which would compile and run the code as well as provide error messages. One student was given the standard error messages, and the other was given the improved error messages that had been translated into Afrikaans. Due to the low response rate, no student was given the English improved error messages. Both students were given instructions on what each program was meant to do and sample input and output were provided. Before attempting the programming task, they completed a quiz containing five multiple-choice questions about basic Java programming constructs (the *pre-survey*). For example, some of the

questions included: "which is the best type of loop to use in Java when the number of iterations is unknown, but you want the loop to run at least once?"; "what will happen when the following code is run?"; "which of the following is true about the basic, primitive array structure in Java?". After completing the programming task, they completed a short survey (the *post-survey*) in which they gave feedback about their experience with the programming task.

Due to the low number of responses, a second evaluation was completed remotely on a Microsoft Form (the Automatic Marker was not used). This form contained four questions about basic Java programming constructs and six programming tasks. The number of questions was reduced from the previous evaluation to encourage more participation. Three forms were created to evaluate each of the datasets: standard error messages; improved error messages in English; and improved error messages in Afrikaans. Each of the programming task questions contained Java code; input that had been given to the program; a description of the expected behaviour of the program; and the error message that the program produced. The respondents were asked to rewrite the code in the answer field so that the program would behave as expected and not give any errors. Participants were permitted to use any IDE of their choosing, as a restriction on this would not have been controllable under the conditions. An invitation sent out to first year computer science students, and 3 out of 669 responded. The first respondent was given a link to the form with the standard error messages; the second respondent was given a link to the form with the improved error messages in English; and the third was given a link to the form with the error messages in Afrikaans.

The results of the evaluation are presented and discussed below.

5 RESULTS

The results of the evaluation are depicted visually in Figure 4, with green representing a correct answer, white representing an incorrect answer, and grey representing where there was no question asked. *P1* through *P5* represents each of the questions about basic Java programming. *T1* through *T7* represents each of the programming tasks. An explanation of the results is given below.

5.1 Pre-Survey Results and Task Completion

In the first evaluation, the participant who was given the standard error messages (*Participant A*), answered four out of the five questions correctly in the pre-survey. The participant who was given the improved error messages that were translated into Afrikaans (*Participant B*), answered all five questions correctly in the pre-survey. Participant A attempted the programming task and gave up after approximately 60 minutes. At which point, the participant had not managed to debug any of the programs. Participant B attempted the programming task and gave up after approximately 70 minutes. At which point, the participant had managed to debug five out of the seven programs correctly. In the second evaluation, of the four questions asked in the pre-survey, the participant with the standard error messages (*Participant C*) answered four correctly; the participant with the improved error messages in English (*Participant D*) answered one correctly; and the participant with the improved error messages in Afrikaans (*Participant E*) answered two correctly. Participant C completed all six programming tasks correctly. Participant D completed five of the six programming tasks correctly. Participant E completed five of the six programming tasks correctly.

Exception Class	Standard Error Message Template	Improved Error Message Template	Translated Error Message Template (Afrikaans)
java.lang.ArrayIndexOutOfBoundsException	Index <int> out of bounds for length <int>	Array elements begin at 0. If an array has a length of <int>, it only has elements ranging from 0 to <int>	Skikking elemente begin by nul. As 'n skikking 'n lengte van <heelgetal> het, het dit net elemente wat wissel van 0 tot <heelgetal>.
java.lang.NullPointerException	*null*	You are attempting to use data from a variable that does not contain any data.	Jy probeer om data van 'n veranderlike te gebruik wat geen data behels nie.
java.util.InputMismatchException	*null*	The input data obtained on the last line is in the incorrect format. Perhaps, a number was expected, but text was provided.	Die invoer data verkry op die laaste lyn is in die inkorrekte formaat. Daar was dalk 'n nommer verwag, maar teks is verskaf.
java.lang.StringIndexOutOfBoundsException	begin <int>, end <int>, length <int>	You are attempting to extract characters between positions <int> and <int> from text that only has <int> character(s).	Jy probeer om karakters te onttrek tussen posisies <heelgetal> en <heelgetal> vanaf teks wat net uit <heelgetal> karakter(s) bestaan.
java.lang.ArithmeticException	<arithmetic operation>	The mathematical operation "<operation>" resulted in a mathematical error.	Die wiskundige bewerking "<bewerking>" het gelei tot 'n wiskundige fout.
java.lang.ClassCastException	class <className> cannot be cast to class <className>	It is not possible to covert a <className> into a <className>, as they are different.	Dit is nie moontlik om 'n <klasNaam> na 'n <klasNaam> te omskep nie, omdat hulle verskillend is.
java.lang.NegativeArraySizeException	<int>	You are attempting create an array with a negative number of elements. An array may not have a length less than 0.	Jy probeer om 'n skikking te skep met 'n negatiewe hoeveelheid van elemente. 'n Skikking mag nie 'n lengte minder as nul hê nie.

Figure 2: Table Displaying the Improved Error Messages used in the Evaluation

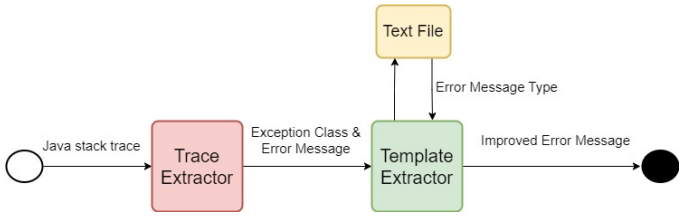

Figure 3: Architecture Diagram representing the Software Interaction

5.2 Analysis of Automatic Marker and Online Form Submissions

Participant A made 11 submissions to the Automatic Marker over a 60 minute period. The participant did not attempt all seven tasks. The code submitted code did not score any more points than the original code. The ninth submission contained a syntax error, which prevented the code from compiling.

Participant B made 19 submissions to the Automatic Marker over a 70 minute period. The submission score increased from the default value of 29 to 39 on the seventh submission. It increased to 54 on the eighth submission, and continued to increased gradually until it reached 86 by the 14th submission. The 17th submission contained a syntax error, which prevented the code from compiling. The participant made two more submissions before giving up.

With regard to the second evaluation, Participant C took 52 minutes to complete the evaluation. Participant D took 31 minutes. Participant E took 861 minutes (it is likely that the participant did not complete the evaluation in one sitting). Participant F completed the evaluation in 16 minutes.

This data does not suggest any correlation between performance and time taken to complete the evaluation.

5.3 Post-Survey Results

After the first evaluation, the two participants were asked to describe their experiences. Both participants agreed that their participation in the study was beneficial to their studies as it gave them an opportunity to practise programming. Participant B said, "I gained some valuable experience that will assist me with the rest of computer science studies". Participant A expressed feelings of fatigue and general academic related stress, as well as the unfamiliarity of the basic text editor as opposed to the JGrasp IDE (which is UCT's recommended IDE for introductory Java programming). Participant A said in the Post-Survey, "it was a nice study, but I found it very hard". Participant B expressed that the lack of JGrasp was uncomfortable at first, but quickly became accustomed to it. With regards to the Afrikaans error messages, Participant B said in the Post-Survey, "although I am able to communicate and understand Afrikaans, mapping that to programming was quite difficult to adjust to at first since I'm so used to programming in English".

6 DISCUSSION

With regard to the first research question (will the improved error messages generated from the expertise of programming tutors with experience teacher Java and guidelines obtained previous research, help first year computer science students at UCT debug code faster than standard Java error messages), the following is noted:

Based on the results from the pre-survey alone, there is insufficient data to suggest any significant correlation between a student's understanding of the basic Java programming constructs and performance on the evaluation task of this study. Observations made during the evaluation and the results of the post-survey, demonstrate a potential correlation exists between the participant's enthusiasm and their performance on the evaluation task of this study.

With regard to the second research question (will the improved, Afrikaans error messages help Afrikaans speaking first year computer science students at UCT to debug code faster than standard error messages written in English?), the following is noted:

The data obtained from the evaluation demonstrates no correlation between the improved error messages and debugging performance. More data would be needed to establish a correlation of this nature.

During the first evaluation, Participant B noted a problem with Automatic Marker, as it was not giving Participant B any feedback after an hour into the study. Upon inspection, it was identified that the Automatic Marker had been configured to report run-time errors, but not compilation errors. The reason why the Automatic Marker was not giving Participant B any feedback at this point in the evaluation was because Participant B had submitted code with syntactic errors, resulting in the code being unable to compile.

7 CONCLUSIONS AND LIMITATIONS

This paper formulated improved templates for 45 Java error messages, and translated seven of them into Afrikaans. The seven translated error message templates were evaluated, and no significant difference between debugging performance of participants who received the standard error messages and the participants who received the improved error messages (in English or Afrikaans) was found. Due to the small sample size of participants and the limited amount of time, insufficient data was gathered to draw any meaningful conclusions about the efficacy of the improved error messages.

8 FUTURE WORK

Even though this paper did not obtain sufficient data to draw conclusions, the findings indicate that removing jargon is conducive to improving error messages. It has demonstrated that error messages may intimidate novice programmers in creating the impression that their code has failed causing their programs to crash. Presenting them with friendlier, more descriptive messages without jargon,

Participant	Format	Error Messages	P1	P2	P3	P4	P5	T1	T2	T3	T4	T5	T6	T7
A	Lab	Standard												
B	Lab	Afrikaans												
C	Online	Standard												
D	Online	English												
E	Online	Afrikaans												
F	Online	Afrikaans												

Figure 4: Table depicting Evaluation Results

may encourage improved performance if the error messages explain how programs can operate more effectively.

Future research in this area can involve further evaluation of the seven improved and translated error messages as well as the improved error messages that were not included in the evaluation. Translating the improved templates into other natural languages is another possibility.

9 ACKNOWLEDGMENTS

- **Dr Zola Mahlaza**: for proposing the idea for this paper, his constant supervision, feedback and guidance.
- **Aslam Safla**: for granting permission to use the lab and arranging its availability for the evaluation session, and for allowing the Automatic Marker script to be deployed on the CSC1016S Amathuba site, and for his assistance with the recruitment of first year students.
- **Andreas Wilsnach**: for translating the improved error message templates into Afrikaans.
- **Stephan Jamieson**: for his explanation of the Automatic Marker and how to use it.

REFERENCES

[1] Shubham K Agrawal. 2016. Syntax errors identification from compiler error messages using ML techniques. *Iowa State University Digital Repository* (2016). https://doi.org/10.31274/etd-180810-5281

[2] Umair Z. Ahmed, Pawan Kumar, Amey Karkare, Purushottam Kar, and Sumit Gulwani. 2018. Compilation Error Repair: For the Student Programs, from the Student Programs. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training* (Gothenburg, Sweden) *(ICSE-SEET '18)*. Association for Computing Machinery, New York, NY, USA, 78–87. https://doi.org/10.1145/3183377.3183383

[3] Titus Barik, Jim Witschey, Brittany Johnson, and Emerson Murphy-Hill. 2014. Compiler Error Notifications Revisited: An Interaction-First Approach for Helping Developers More Effectively Comprehend and Resolve Error Notifications. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 536–539. https://doi.org/10.1145/2591062.2591124

[4] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research *(ITiCSE-WGR '19)*. Association for Computing Machinery, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

[5] Brett A. Becker, Paul Denny, James Prather, Raymond Pettit, Robert Nix, and Catherine Mooney. 2021. Towards Assessing the Readability of Programming Error Messages. In *Proceedings of the 23rd Australasian Computing Education Conference* (Virtual, SA, Australia) *(ACE '21)*. Association for Computing Machinery, New York, NY, USA, 181–188. https://doi.org/10.1145/3441636.3442320

[6] Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (2016), 148–175. https://doi.org/10.1080/08993408.2016.1225464 arXiv:https://doi.org/10.1080/08993408.2016.1225464

[7] Brett A. Becker, Cormac Murray, Tianyi Tao, Changheng Song, Robert McCartney, and Kate Sanders. 2018. Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 634–639. https://doi.org/10.1145/3159450.3159453

[8] Arthur Chargué raud. 2015. Improving Type Error Messages in OCaml. *Electronic Proceedings in Theoretical Computer Science* 198 (dec 2015), 80–97. https://doi.org/10.4204/eptcs.198.4

[9] Paul Denny, James Prather, and Brett A. Becker. 2020. Error Message Readability and Novice Debugging Performance. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 480–486. https://doi.org/10.1145/3341525.3387384

[10] Saifaddin Galal. 2022. Statista. https://www.statista.com/statistics/1114302/distribution-of-languages-spoken-inside-and-outside-of-households-in-south-africa/. Accessed: March 21, 2023.

[11] Jang-Wu Jo, Byeong-Mo Chang, Kwangkeun Yi, and Kwang-Moo Choe. 2004. An uncaught exception analysis for Java. *Journal of Systems and Software* 72, 1 (2004), 59–69. https://doi.org/10.1016/S0164-1212(03)00057-8

[12] Ioannis Karvelas, Joe Dillane, and Brett A. Becker. 2020. Compile Much? A Closer Look at the Programming Behavior of Novices in Different Compilation and Error Message Presentation Contexts. In *United Kingdom amp; Ireland Computing Education Research Conference*. (Glasgow, United Kingdom) *(UKICER '20)*. Association for Computing Machinery, New York, NY, USA, 59–65. https://doi.org/10.1145/3416465.3416471

[13] Tobias Kohn and Bill Manaris. 2020. Tell Me What's Wrong: A Python IDE with Error Messages. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1054–1060. https://doi.org/10.1145/3328778.3366920

[14] Davin McCall and Michael Kolling. 2014. Meaningful categorisation of novice programmer errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Vol. 2015-. IEEE, 1–8.

[15] Davin McCall and Michael Kölling. 2019. A New Look at Novice Programmer Errors. *ACM Trans. Comput. Educ.* 19, 4, Article 38 (jul 2019), 30 pages. https://doi.org/10.1145/3335814

[16] Bertrand M. Roehner. 2015. Translation into any natural language of the error messages generated by any computer program. *CoRR* abs/1508.04936 (2015). arXiv:1508.04936 http://arxiv.org/abs/1508.04936

[17] Alfred Strohmeier and Stanislav Chachkov. 2001. A Side-by-Side Comparison of Exception Handling in Ada and Java. *Ada Lett.* XXI, 3 (sep 2001), 41–56. https://doi.org/10.1145/568671.568683

[18] Emillie Thiselton and Christoph Treude. 2019. Enhancing Python Compiler Error Messages via Stack Overflow. *CoRR* abs/1906.11456 (2019). arXiv:1906.11456 http://arxiv.org/abs/1906.11456

[19] V. Javier Traver. 2010. On Compiler Error Messages: What They Say and What They Mean. *Advances in human-computer interaction* 2010 (2010), 1–26.

[20] Daan P. Wissing. 2020. Afrikaans. *Journal of the International Phonetic Association* 50, 1 (2020), 127–140. https://doi.org/10.1017/S0025100318000269