Improving Text-Based Programming Error Messages For South African Students

Research Proposal

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

Mandisa Tunzi
tnzman002@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Python and Java are popular programming languages, especially among novices. One of the significant challenges that novice programmers face is understanding the languages' error messages. As the primary source of feedback, error messages can be a barrier to learning if novice programmers struggle to interpret and respond to them effectively. There is significant literature on enhancing compiler error messages to improve their usability and effectiveness. These studies either produce or apply guidelines formulated by other researchers to improve error messages. The studies show varying effects on improving the usability of error messages, with some studies reporting improvements. This research has focused primarily on how to improve an error message's usability, without assessing its readability. While most researchers have emphasised the importance of error message readability, its impact on improving error messages remains unclear, the literature lacks in concrete criteria for achieving and assessing it. Furthermore, in the context of South Africa, where people speak different languages, there is a need to investigate the impact of using alternative languages to present error messages. This paper proposes a system to enhance the readability of Python and Java error messages. Reproducing a recent study, the system applies a set of recently guidelines, including simpler language, full sentences, and word economy, with the assistance of first-year Computer Science lecturers, TAs or tutors and natural language processing tools. Additionally, the system presents error messages in isiXhosa and Afrikaans for Python and Java, respectively, to assess the impact of providing error messages in alternative languages. The effectiveness of the guidelines is evaluated based on first-year Computer Science students' perceptions and statistical analysis of debugging time. The translations will be generated by a linguist and will be based on the already enhanced error messages. The goal of this research is to validate existing guidelines and to determine the impact of presenting error messages in different natural languages.

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 syntactic mistakes in the code. If it finds any, it alerts the programmer by displaying an error message [3].

Error messages also play an important role in educating novice programmers by alerting them to mistakes that they make [4]. 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 do not have such knowledge [7]. Becker et al. [2019], in their work synthesise the work done to enhance error messages in an attempt to improve their usability and effectiveness to novice programmers. After analysing the work of 107 papers in the literature, they concluded on a call for research based on identified gaps. This included:

- Determining a formal definition for readability as well as a way to assess it.
- Gathering empirical evidence in support of one guideline or related pedagogical practice.
- Determining whether the use of identifiers and external documentation could be used to improve the understanding of error messages.

Denny et al. [2021] present their efforts in response to the first point above: Readability. Their work establishes 4 readability guidelines that may be used to transform programming error messages. The proposed research continues from their work and applies these guidelines on Python and Java error messages in order to investigate whether they improve the readability of error messages.

Another difficulty is that error messages are almost always presented in English, which does not accommodate for people who speak other languages. This is pertinent, particularly for South African students of programming, as English is not the first language of or only language spoken by most South Africans [8]. As such, the proposed research considers the impact of presenting programming error messages in alternative languages that are spoken by programmers, instead of solely English, as presently done. Although preliminary, there have been some positive results of creating programming languages for non-native English programmers [6, 9, 15]. The proposed research aims to investigate the impact of this translated presentation for error messages.

The following research questions arise:

- RQ1: Will enhancing programming error messages using the expertise of first year Computer Science lecturers, TAs or tutors and natural language generation tools validate

a recently established set of guidelines for improving the readability of programming error messages?
- RQ2: Will these enhanced error messages allow first year computer science students at UCT to debug code faster than with default error messages?
- RQ3: Will enhanced messages written in an alternative language (Afrikaans or isiXhosa) spoken by first year computer science students at UCT allow them to debug code faster than with default error messages?

This paper is laid out as follows: In section 2 the Related Work is presented. This will be followed by the Procedures and Methods in section 3. Thereafter the Ethical, Professional and Legal Issues are presented in section 4. Section 5 discusses the anticipated outcomes and finally, section 7 presents the Project Plan.

2 RELATED WORK

This section provides a discussion on the past work that informs the proposed research.

In order to determine which error messages are encountered by novices the most, studies tend to obtain error message data from large data sets, such as the Blackbox Project, and perform a statistical analysis on the data [13]. The statistical analysis typically includes a category development process in which different categories of errors are decided upon and a category hierarchy is formed. A process of categorisation is then done, assigning each error message to a category. From here a data analysis can be completed by calculating error frequencies, levels of error message coverage and error difficulties and severity [12]. This process may differ among studies, where some may begin with a data filtering process and end by removing outliers [10] (Figure 1 provides a visualisation of this process). It may also use machine learning techniques to analyse data, such as Hierarchical Clustering, Support Vector Machines, Probabilistic Topic Modelling and/or Multi-Label Classification (whereby the process of hierarchy generation and categorisation is automated) [1].

Enhancing error messages, which refers to the process of intercepting standard programming errors and changing them before being presented to users, has been investigated by many researchers. In order to enhance error messages, studies either produce a set of guidelines based on their own or others' research [3] or they apply known principles for good error messaging and produce their own enhanced error messages [2, 11]. Becker et al. [2019] gathered ten general categories of guidelines for enhancing programming error messages using a corpus of 107 papers, which are as follows: Increased Readability, Reduced Cognitive Load, Provide Context, use a Positive Tone, Provide Examples, Offer Hints or Solutions, allow dynamic interaction, provide scaffolding, use logical argumentation, and report errors at the right time. Some of these studies discuss various ways in which these guidelines have been applied. Similarities in the approaches include the creation of visual debuggers which explain the cause of errors in a visual format [2, 11] or by rewriting the error messages in more natural language [4, 5, 7].

Although there is a general consensus throughout this literature that readability is essential for enhancing error messages effectively, the literature lacks actionable guidelines for achieving it. In a recent

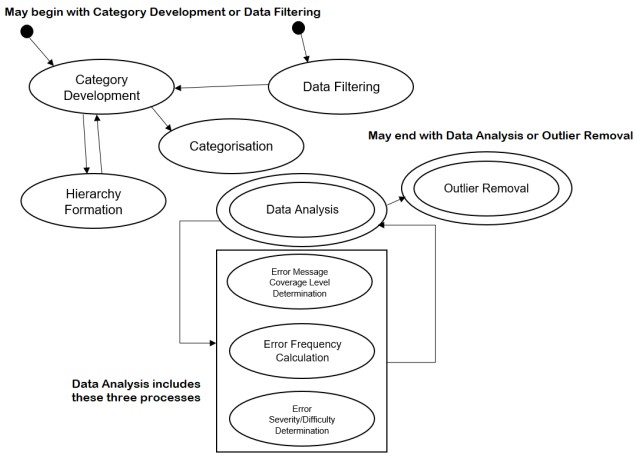


Figure 1: Visualisation of the Statistical Analysis Process for Identifying Most Commonly Encountered Errors

study, Denny et al. [2021] conducted three experiments in order to determine the factors that affect the readability of an error message, and the extent to which each factor has an impact on error message readability. In the first experiment, 1st year students doing a CS1 course were tasked with providing numeric ratings to evaluate the readability of the top 20 error messages derived from 3 widely used programming languages: Java, Python and C. This study revealed that participants perceived error messages that were shorter and had less jargon, such as "EOF" or "positional argument" to be more readable. The second experiment involved showing 8 error messages, four of which were standard and four which were reworded to participants. Participants were asked to describe what made each message easy or difficult to read, considering factors such as word count, word length, specific characters and line numbers. Thematic analysis of the participant responses identified length, jargon, sentence structure, and vocabulary as common factors affecting readability. The third experiment involved 18 error messages from the 1st experiment and participants were asked to rate them on a fixed set of criteria derived from the first two experiments. A strong correlation with earlier ratings in the first experiment was reported. The research concluded by providing four guidelines for error message readability, namely:

- Remove jargon
- Write messages in complete sentences
- Use simple vocabulary
- Use an economy of words

that can be applied to improve the readability of programming error messages. As far as the researchers of the proposed research know, one other study has applied the established guidelines using a machine learning model[14]. The study found positive results in terms of the guidelines improving the readability of programming error messages. This project aims to reproduce this study. Using a different set of methods and new set of participants, the proposed research will investigate the validity of the guidelines in improving readability of programming error messages for CS1 students.

With the large percentage of South African citizens being non–native English speakers, there is a gap in the literature to investigate the impact of having error messages entirely in English and whether changing this would improve them. So far there has not been any work done in literature to investigate the impact of specifically changing the language in which error messages are presented. However, there is research on the broader issue of localising programming languages which is reporting positive results. This includes the work of Guo [2018] who investigated the barriers faced by non-native English speakers when learning how to program; and the work of Dasgupta and Hill [2017] and Raj et al. [2018] who conduct empirical investigations on the influence of learners' native language on their ability to learn a programming language. The study by Guo [2018] reported that localising programming languages (presenting them in a language spoken by the programmer) can have a positive impact for non–native English speakers. It found that Italian-speaking novice programmers performed better and showed a more positive attitude when using a localised version of Scratch, a visual programming language, compared to using the English version. Raj et al. [2018] reported positive reactions from Tamil-speaking students who were taught programming in their native language and English. Overall, the studies suggest that multilingual programming languages may improve novice programmers' learning experience, but more research is needed to evaluate its effectiveness and how it can be incorporated to improve error messages.

3 PROCEDURES AND METHODS

This section describes the proposed system design, along with how it will be utilised to gather the data that will be required to answer the research questions.

3.1 Enhancing Standard Error Messages

The first step towards enhancing the standard error messages will be to determine the subset of error messages that will be used. This will be subset of error messages that are mostly encountered by inexperienced programmers. For the Java system, this data set will be extracted from the Blackbox project and for the Python error messages, the data set will be extracted from [16] and Python documentation. After determining the subset of error messages to enhance, the four readability factors will be applied, jargon will be removed, messages will be rewritten in full sentences, simple words will be used and the economy of words in defining templates will be used to generate the new error messages. The templates will be in the form of predefined formats to generate error messages based on their types (i.e., all syntax errors presented in a similar structure and all run-time errors will be presented in a different structure). These templates will be designed using the expertise of CS1, TAs or tutors. They have experience with first year Computer Science students and are aware of the programming difficulties that the students experience when it comes to programming and reading error messages. The templates will be designed such that they include placeholders for the relevant information that cannot be predefined (which are specific to the error message that occurs), such as the line number where the error occurred, the variable or expected data type. For example, the template for a python run-time error might look like this:

- *"Error": There is a **{error type}** on line **{line number}**. **{text}**.*

A ZeroDivisionError, which has the standard form:

- *Traceback (most recent call last): File "C:.py", line 1, in <module> print (2/0) builtins.ZeroDivisionError: division by zero*

in Python would be:

- *Error: There is a **ZeroDivisionErroron line 2**. **You cannot divide by zero**.*

Generating error messages using these templates will allow for the enhanced error messages to be presented in a consistent way. After having designed the format for the different error types, algorithms based on a parsing mechanism that can extract relevant information from standard error messages, such as the error type, line number or any other relevant context will be designed. The information will be given to the NLG tool. The NLG tool will use the information to generate an enhanced error message, based on the guidelines and templates with which it has been provided.

3.2 Prototype System for Enhancing Error Messages

To investigate the research questions, an experiment will be carried out in which test subjects will debug code in an IDE. The IDE will contain a plugin that the researchers will build. The plugin will take the default messages provided by the IDE (PyCharm for Python and JGrasp for Java) when the code is run as input. The input will be passed to the developed NLG algorithms, which generate enhanced error messages, which will be given as output and presented to the subject instead of the default error messages. The plugin will have an option to generate the enhanced error messages in English, Afrikaans or isiXhosa (the most commonly spoken languages in the Western Cape).

The architecture of the system has three main components: The IDE (PyCharm for Python and JGrasp for Java), which is the main component and is responsible for providing an interface for writing, debugging, and executing code. The enhanced error message handler plugin, which is to be installed and activated in the IDE. It will override the default error messages of the IDEs and retrieve corresponding enhanced/ translated for given errors from the database. Error message database, which stores mappings between error messages and corresponding enhanced error message templates in two different languages (English and Afrikaans for Java system) and (English and isiXhosa for Python system).

Figure 2 depicts the IDE as the top-level component, which includes the logic, presentation, and data layers. The enhanced error message handler, which is the core component of the system, resides in the logic layer and handles the processing of error messages. The error message database and the mappings component, which are part of the data layer, provide the necessary templates for the enhanced error message generation to the enhanced error message handler. The user interface (console), which is part of the presentation layer, provides a way for students to interact with the system. When an error occurs, the Handler processes the error message using the error message database and mappings to produce an enhanced/ translated error message, which is then displayed via

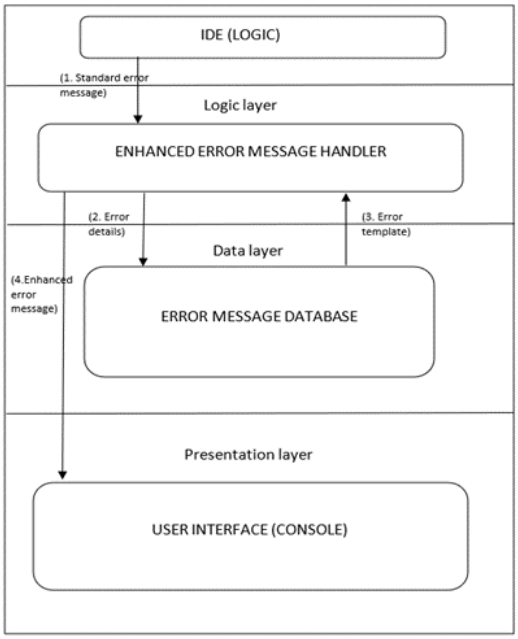


Figure 2: Diagram depicting the different components of the system in a layered format

the console interface. Organising the system into separate layers improves its modularity, maintainability and scalability. Each layer has a well-defined responsibility, which makes it easy to modify and update the system without affecting the other layer (i.e., to add new error message mappings, the error message database can be modified without affecting the other layers.

3.3 Evaluation

To investigate the research questions, a user study will be conducted with a sample of first year computer science students at UCT. There will be two separate studies, one using the Python system and one using the Java system. However the same methodology will be used. The study will be designed as follows: Approximately 20 to 40 participants will be recruited. Within the two studies, participants will be divided into two equally split groups: a control group and an intervention group. The control group will have access to the IDE without the plugin, so will be presented with the default error messages when completing the debugging tasks. The intervention group will have access to the IDE with the plugin, so will be presented with the enhanced error messages when completing the debugging ta. Participants should be able to speak English and either Afrikaans or isiXhosa (and preferably there should be a fair distribution between Afrikaans and isiXhosa speakers). Before the study begins, participants will be expected to complete a mini survey and a few programming tasks to assess their programming skills. The survey will use an online survey with Likert scale questions with questions such as: “On a scale of 1 to 5, how confident are you in your ability to write code in Python/Java?" Programming tasks on different difficulty levels for first year programming will be designed for participants to complete. Assessment of programming skills will establish the baseline for comparison and control for differences in programming skills.

Participants will then be given erroneous code to debug. The coding tasks will be written by the researchers and will contain deliberate errors based on introductory programming principles such as data types, variables, operators, conditional statements and loops. The CS1 lecturers, TAs or tutors will be asked to assist in the development of these tasks. This will allow the different levels of difficulty of the programming tasks to be controlled. The total number of the tasks to be given to users depends on the subset of error messages that will be supported by the system. Participants will be given one task at a time in random order along with a brief explanation of what the code is supposed to do and the sample input/ output so that they can determine when they have successfully debugged the code. To ensure that participants do not consult other sources, such as Stack Overflow during the experiment, clear instructions will be given to the participants at the beginning of the study that they are not allowed to use any external resources to complete the debugging task and the participants will be monitored throughout the exercise. This plugin will keep track of the time taken by each participant to complete the task. Every time the code is run, the plugin will record the current time and the time after the first successful compilation will be stored. This will be done for both enhanced error messages that are presented in English and those presented in the Afrikaans/isiXhosa. A statistical analysis will be performed on the time data. It will determine if there are any significant differences between the mean time to solve erroneous code between control and intervention groups. This analysis will provide answers for RQ2 and RQ3. After the participants have completed these tasks, they will be asked to fill out a survey at the end of the experiment to determine RQ1 based on student perceptions about whether the enhanced error messages were more readable and understandable relative to the standard ones. This information will be analysed through by the means of thematic analysis to find patterns in user responses.

4 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

In order to formulate enhanced error messages, first year Computer Science lecturers, TAs or tutors will be consulted. These discussions will not involve anything potentially dangerous or unethical. Their participation in the study will be beneficial to their students as the study aims to makes learning coding easier.

In order to evaluate the efficacy and usefulness of the enhanced error messages, a group of first year students will be recruited and will be given a debugging task to complete. All first year students should be at the age of majority. The debugging task will not involve anything potentially dangerous or unethical. The students’ participation in the study will positively contribute to the experience of future students and their participation may be compensated by means of a simple voucher. The students’ participation will be voluntary and informed consent will be sought. A clearance form will be submitted to ensure that the study complies with UCT’s research ethics for human subjects. The researchers recognise that error messages are an integral part of the programming learning process,

and the proposed changes to their presentation may potentially affect students' comprehension. However, they believe that any such risk is minimal, considering that the study will not last longer than two days. To further mitigate this risk, the researchers will explicitly inform the participants that the enhanced error messages used in the study are not intended to replace the standard error messages, nor are they claimed to be more effective. Participants will be informed that this is an experiment aimed at collecting data on presenting error messages and its impact on students.

The code used in the debugging task will be generated by the researchers themselves and will comply with any language licensing requirements. The code will only be used for research and not for commercial gain. The plugin will measure the time taken by the participants to complete the task. After the first successful run (when the code is error free), the current time will be stored in a database.

5 ANTICIPATED OUTCOMES

5.1 System

The system is expected to be in the form of an IDE plugin that will take raw error messages as input, and produce enhanced error messages as output.

The raw error messages will be classified by means of an error hierarchy data structure, and then mapped to a corresponding enhanced error message in a database.

A potential design challenge would be to design an algorithm that will be able to correctly classify every possible error message provided by a compiler/interpreter. This will be dealt with by only including errors in the debugging task that the algorithm can correctly classify. Future work on this can investigate improvements to the algorithm.

5.2 Impact

Firstly, it is hypothesised that if a set of guidelines (remove jargon, use full sentences, and use an economy of words) is applied to an error message, its readability will improve.

Secondly, if an error message is displayed in another language spoken by the programmer, the error message will be easier to understand.

This is expected to make learning programming easier for novices.

5.3 Key Success Factors

It is expected that the project will improve the coding and debugging abilities of novice programmers by means of enhanced error messages. If the guidelines are effective, users will find the error message more readable. This will be demonstrated by whether the user found the error messages easy to read. It will also be demonstrated by an improvement in time-to-fix. If the students who were given the enhanced error messages debugged the code faster than the students who were given the raw error messages, the project would be judged as a success. If the language of the error message is changed to the another language spoken by the programmer, users will find the error message more understandable. This will be demonstrated by an improvement and time-to-fix: If the students who were given the translated error messages debugged the code faster than the students who were given the raw error messages, the project would be judged as a success.

6 PROJECT PLAN

The project will be divided up as follows:

Danny Guttmann will create a plugin for the JGrasp IDE that will convert the default Java error messages into enhanced error messages.

Mandisa Tunzi will create a plugin for the PyCharm IDE that will convert the default Python error messages into enhanced error messages.

All parts of the system as described in the architecture will be implemented using separate tools for the Python and Java systems. The approach will be the same (template-based approach), but the IDE and natural language processing tools or libraries that will be used for extracting information from the standard error message and into the templates will be implemented separately.

The project will require a laptop with a good Internet connection to complete the coding component. A suitable number of first year computer science students will be needed for testing. PyCharm and JGrasp will be used for obtaining raw error messages and will be integrated with the developed plugin.

The potential risks and how they will be managed are outlined in Figure 5. The significant milestones of the projected are depicted in Figure 3. Figure 4 contains a Gantt Chart featuring the project timeline and full labour plan.

REFERENCES

[1] Shubham K Agrawal. 2016. Syntax errors identification from compiler error messages using ML techniques. https://doi.org/10.31274/etd-180810-5281

[2] 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

[3] 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

[4] 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

[5] 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

[6] Sayamindu Dasgupta and Benjamin Mako Hill. 2017. Learning to code in localized programming languages. In *Proceedings of the fourth (2017) ACM conference on learning@ scale*. 33–39.

[7] 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

[8] 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.

[9] Philip J Guo. 2018. Non-native english speakers learning computer programming: Barriers, desires, and design opportunities. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–14.

[10] 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

[11] 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

[12] 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.

[13] 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

[14] James Prather, Paul Denny, Brett A Becker, Robert Nix, Brent N Reeves, Arisoa S Randrianasolo, and Garrett Powell. 2023. First Steps Towards Predicting the Readability of Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1.* 549–555.

[15] Adalbert Gerald Soosai Raj, Kasama Ketsuriyonk, Jignesh M Patel, and Richard Halverson. 2018. Does native language play a role in learning a programming language?. In *Proceedings of the 49th ACM technical symposium on computer science education.* 417–422.

[16] Alexander William Wong, Amir Salimi, Shaiful Chowdhury, and Abram Hindle. 2019. Syntax and Stack Overflow: A methodology for extracting a corpus of syntax errors and fixes. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 318–322.

Appendix A PROJECT MILESTONES

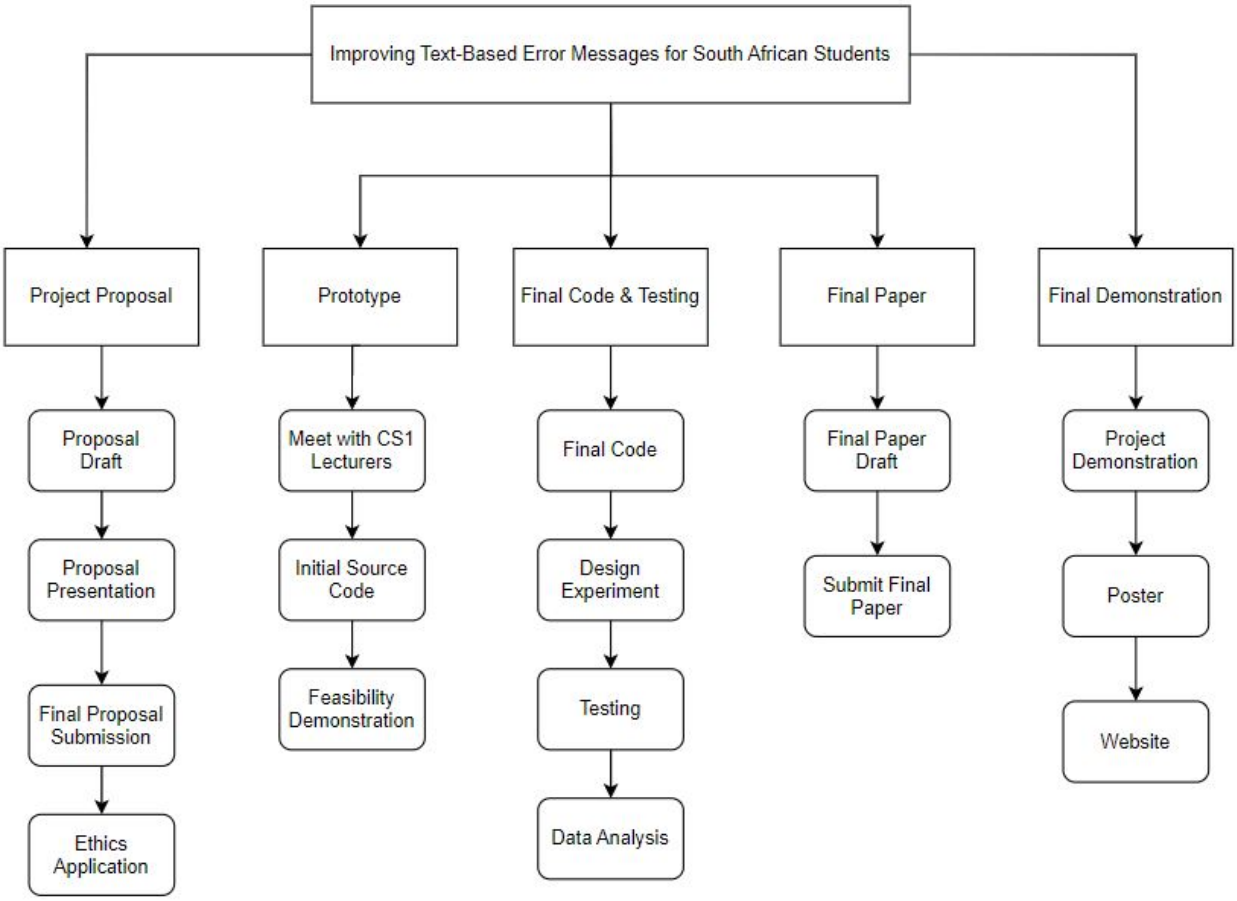
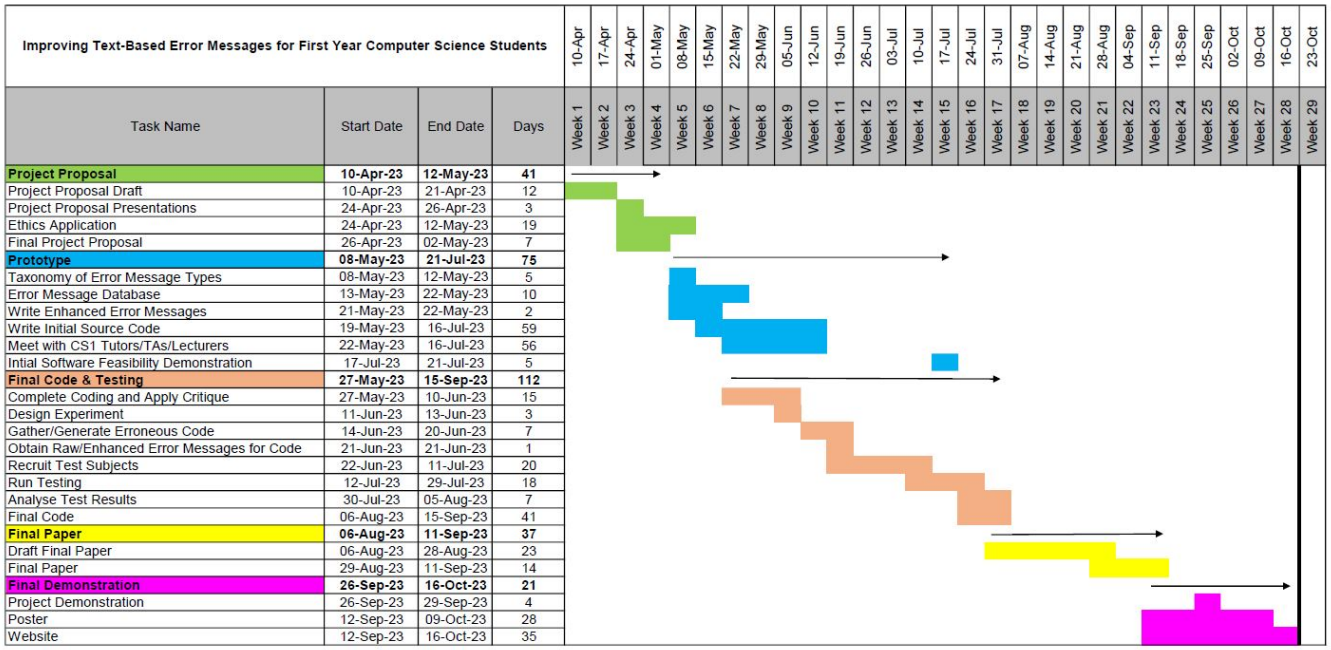


Figure 3: Diagram Depicting Project Milestones

Appendix B GANTT CHART



Task Name	Start Date	End Date	Days
Project Proposal	**10-Apr-23**	**12-May-23**	**41**
Project Proposal Draft	10-Apr-23	21-Apr-23	12
Project Proposal Presentations	24-Apr-23	26-Apr-23	3
Ethics Application	24-Apr-23	12-May-23	19
Final Project Proposal	26-Apr-23	02-May-23	7
Prototype	**08-May-23**	**21-Jul-23**	**75**
Taxonomy of Error Message Types	08-May-23	12-May-23	5
Error Message Database	13-May-23	22-May-23	10
Write Enhanced Error Messages	21-May-23	22-May-23	2
Write Initial Source Code	19-May-23	16-Jul-23	59
Meet with CS1 Tutors/TAs/Lecturers	22-May-23	16-Jul-23	56
Intial Software Feasibility Demonstration	17-Jul-23	21-Jul-23	5
Final Code & Testing	**27-May-23**	**15-Sep-23**	**112**
Complete Coding and Apply Critique	27-May-23	10-Jun-23	15
Design Experiment	11-Jun-23	13-Jun-23	3
Gather/Generate Erroneous Code	14-Jun-23	20-Jun-23	7
Obtain Raw/Enhanced Error Messages for Code	21-Jun-23	21-Jun-23	1
Recruit Test Subjects	22-Jun-23	11-Jul-23	20
Run Testing	12-Jul-23	29-Jul-23	18
Analyse Test Results	30-Jul-23	05-Aug-23	7
Final Code	06-Aug-23	15-Sep-23	41
Final Paper	**06-Aug-23**	**11-Sep-23**	**37**
Draft Final Paper	06-Aug-23	28-Aug-23	23
Final Paper	29-Aug-23	11-Sep-23	14
Final Demonstration	**26-Sep-23**	**16-Oct-23**	**21**
Project Demonstration	26-Sep-23	29-Sep-23	4
Poster	12-Sep-23	09-Oct-23	28
Website	12-Sep-23	16-Oct-23	35

Figure 4: Gantt Chart Depicting Project Timeline

Appendix C RISK MATRIX

Rank	Risk	Likelihood	Impact	Severity	Mitigation	Monitor	Management
1.	Loadshedding compromising productivity.	High	Falling behind schedule.	Minor	Check the Load Shedding schedule and plan time accordingly.	Keep track of changing load shedding stages.	Try to make up for lost time when the power is back on.
2	Losing project code.	Medium	Having to start from the beginning, falling behind schedule, reduced quality due to hastiness.	Catastrophic	Backup regularly to Cloud-based code repository.	Ensuring that all important modifications to code are saved and every version available.	Replan schedule to obtain more efficient use of time.
3.	Not being able to get a meeting with CS1 lecturers.	Medium	Will not have insight into common student errors and programming troubles.	Major	Contact several lecturers (not just one) well before the deadline.	Be mindful of lecturers busy schedules, monitor email responses from them and send follow up emails.	Rely solely on guidelines from previous research.
.4	Gold Plating.	Low	Important requirements not fulfilled.	Major	Be clear on the critical requirements.	Constantly refer to timeline and plan.	Focus on the project's core components.
5.	Falling behind schedule.	Low	Missed deadlines or reduced quality.	Major	Stick to tasks and their deadlines on Gantt Chart.	Frequent check ups with partners to determine project progress.	Reevaluate scope; Attempt to finish future tasks, even though previous tasks are not yet complete.

Figure 5: Table Depicting Identified Risks of the Project Ranked in Order of Likelihood