Universiteit Leiden
The Netherlands

Computer Science

Error message needs of novice programmers in gradual programming languages

Roos van Amerongen

Supervisors:
Felienne Hermans & Giulio Barbero

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

24/12/2022

Acknowledgements

During the design and execution of my research, I was closely followed and guided by my professor, Felienne Hermans. Words cannot express my gratitude for all the effort and motivating conversations we were able to have about my thesis topic. I am extremely grateful for this way of guidance with a personal approach, which has helped my research to run as smoothly as possible. Besides, I would like to extend my sincerest thanks to Giulio Barbero for also providing accessible contact and feedback at any time. The constructive words and guidance of the writing process really helped me to create the final results of this thesis.

Abstract

Novice programmers can learn text-based programming step-by-step through the gradual programming language and program Hedy. New programming concepts are taught through several different levels and challenges in this program, to slowly help teaching Python to children. During these first learning stages of writing code, understandable feedback on possible code-writing complications plays a crucial role. Therefore, it is very important that clear and comprehensible error messages are shown in Hedy. However, little research has been done into what novice programmers consider to be the clearest error message structure in gradual programming languages. That is why this study dives into the error messages needs of novice programmers in a gradual programming language.

First, a literature search was conducted for this research consisting of three parts. The first part shows a list of characteristics good error messages could have. The second part analyzes various previously executed studies on the intelligibility of error messages among groups of novice programmers. The final literature part shows information about different forms of feedback and their effects on learning results. All gathered information was used to prepare for a practical study. The design of this practical part of the study was classroom-based observation research. Various newly created error messages were presented to a group of eleven and twelve-year-old novice programmers. Using thematic analysis, themes and patterns were filtered out of the participants' opinions on the error messages they saw. In conclusion, this study revealed that explicit correction and repetition of theoretical information could best be processed in error messages to improve the clarity and comprehensibility of the error messages in gradual programming languages the most. Besides this, the location of the errors could be highlighted more explicitly. These increase the positive effects of learning from errors in novice programming languages.

Contents

1 Introduction

It is often a major challenge for novice programmers to enter the programming world without prior knowledge. Bennedsen and Caspersen 2007 stated that a large number of new concepts, rules, and limitations in programming can quickly become overwhelming. Students can therefore learn programming gradually, for example by using a gradual programming language like Hedy (Hermans n.d.). While learning, students become very dependent on the error messages processed in the first languages they encounter. According to McGough 2013, this is also where most first feedback on code mistakes is received.

Ewerlöf n.d. describes error messages as compact texts, intended to indicate mistakes in programmed code. The messages can include information about *why*, *where*, and *when* an error occurred. But also *how* the error was detected and *what* fixing can be done. To store this much information compactly in one comment, error messages are often summarized with specific commands and lots of programming jargon.

However, previous research conducted by Shneiderman n.d. shows that this build-up of error messages is not always understood by novice programmers. When novices encounter new phrases or codes in their error messages, they can get confused and discouraged from continuing to repair a code. Upon that, Bakker 2021 also explains that commands can be confusing. He explains that novice programmers normally expect more of a natural language conversation with the messages, than a command-like conversation from error messages. The novice programmers tend to assume some type of cooperation with their computer.

Knowing this information about the limited understandability of error messages under novice programmers, the following question arises. What is the clearest and most comprehensible way to inform novice programmers of the errors they encounter in gradual programming languages?

1.1 Goal of the research

This research aims to find a list of the best characteristics error messages in a gradual programming language should have. The gradual programming language for which these error message needs are specifically collected is Hedy. Novice programmers will be able to give their own opinion on certain error messages in the Hedy environment. The different comments and opinions of participants of the study could help to eventually make error messages in Hedy more clear, comprehensible, and therefore more meaningful for novice programmers. The main research question of this study is.

RQ1: What are the error message needs of novice programmers in a gradual programming language?

To answer this question, information will first be gathered and processed through theoretical research. In this section of the thesis, the general guidelines for comprehensible error messages will be examined. Factors that benefit or worsen the readability of error messages will also be investigated. And by researching corrective feedback, it is investigated how different feedback types can be processed in error messages. Through practical research, the following sub-research

question will also be investigated more specifically:

RQ2: Which corrective feedback strategy can best be incorporated into Hedy's errors to increase the clarity and comprehensibility of the error messages?

2 Background

Hedy is a textual gradual programming language intended to learn Python to young novice programmers step-by-step. Hermans n.d. is the founder of the programming language. She is a Professor of Computer Science Education at VU University Amsterdam. The programming language was created as an open-source project, which means that, in addition to Felienne Hermans, computer scientists from all over the world can contribute to the expansion of the platform.

As mentioned, Hedy targets young novice programmers. Specifically, the audience of this programming language consists of approximately eleven and twelve-year-old novices. All code, theory, and error messages in the programming language Hedy are intended to be understood by this audience. Therefore, error messages are different from those in non-gradual, general-purpose bigger programming languages like `Python` and `C++`. Error messages in Python and C++ are more command-like, while Hedy shows more of a natural language-like conversation.

Hedy's programming language teaches programming through several levels. At each level, a bit more programming concepts are thought. For example, in level 1 the use of the `print` command is explained, while in level 7 loops are shown for the first time. Running over the same code several times is introduced with the word `repeat`. In addition to levels, the Hedy platform also contains games and quizzes that can put programming knowledge to the test. The platform can be found via the website: www.hedycode.com/hedy.

3 Review of literature

This literature review consists of three parts to determine the structure, guidelines, and needs for error messages for novice programmers. The first part of the literature review looks at general guidelines for error messages that could benefit learning results from the feedback. The second part analyzes studies about the readability of commonly used and newly created error messages under novice programmers. Lastly, the third part of the literature review focuses specifically on what feedback forms can be incorporated into new error messages for novice programmers.

3.1 Guidelines for error messages

3.1.1 Characteristics that benefit error messages

Lots of research has been done on how good error messages can be written. According to Minhas 2008, one of the most important parts of an error message is problem identification. Problem identification consists of clearly identifying the root cause of a programming problem and then providing a detailed problem statement. An error should therefore at least tell what type of error occurred. This is more important than just offering directly a solution to the error.

Several characteristics of error messages are also important to be incorporated into error messages. For example, the tone of an error message is very controlling. Birkett n.d. created a framework for characteristics of error messages named "the 4 H's". The four H's in this framework stand for: human, helpful, humorous, and humble. These four characteristics are explained in Figure 1.

Human	Error messages should contain logical language for people to be understood. Sentences should not come across as robot-like.
Helpful	A message must always provide enough room for solutions to recover from a noted error.
Humor	In some cases, humor could help emphasize a mistake to quell frustration. For example, an animation next to an error can be added. But it has to stay professional.
Humble	An error message must be humble. " Don't blame the user" is one of the most important thoughts while creating error messages to quell frustration again.

Figure 1: Framework for characteristics of error messages created by Birkett n.d., called "the 4 H's".

According to Molich n.d. and Nielsen 2001, error messages should be constructive, comprehensible, precise, visible, and polite. Nielsen 2001 also stated that error messages should be explicit and preserved. All characteristics are explained in Figure 2.

3.1.2 Factors that worsen error messages

Besides beneficial aspects of error messages, there are also lots of derogating aspects to be named. According to researchers, like Becker and Bouchard 2019, feedback in error messages can be considered useless if they are too long and complex. The message of the feedback is strongly

Constructive	Instead of only emphasizing what is "wrong" or "illegal", guiding sentences can be used that hint more towards possible solutions for mistakes. Sentences like "[this] must be [that]" are therefore of better structure.
Comprehensible	Jargon or abbreviations that are not recognized should be avoided in error messages.
Precise	Vague generalizations should be avoided. The location, time, and type of error can best all be mentioned instead of only naming general labels for errors.
Visible	An error must be indicated very conspicuously. It must be clear which elements of a code exactly need to be changed.
Polite	An error message should not blame the user for a mistake. It should have a neutral or even uplifting tone.
Explicit	A very explicit mention that there is a mistake is important. The opposite (not directly mentioning that an error has been detected) ensures that programmers can get lost.
Preserve	A way to learn form mistakes is to save old attempts and programming mistakes so programmers can see what they've already done during previous attempts.

Figure 2: Characteristics error messages should have, according to Molich n.d. and Nielsen 2001. The explanations of characteristics are provided by Nielsen 2001.

weakened because the long-winded becoming of the feedback is distracting. Many readers will drop out and not receive the full evaluation. There is only a certain amount of information that can be mentioned at once to make the feedback message come across clearly.

A second example of bad aspects in error messages is given by Schute n.d. She explains that if goals are set too high and are therefore unattainable, the message can also backfire. When difficult solutions in error messages are given on simple subject matter, a learner gets discouraged. The student is more likely to fail repetitively. Feedback should therefore not be too demanding. On the other hand, R. C. Birney 1969 also explains that if goals are set too simple, the power of feedback also reduces. Feedback will be read more carelessly if it describes too simple concepts.

3.2 Studies on readability of error messages

In May 2021, P. Denny 2021 described three studies in one paper on the readability of error messages. The studies were conducted by a team of seven university students from all around the world. The first study was an exploratory, quantitative study where participants rated the readability of common error messages on a scale of 1 to 10. The second study was qualitative where participants gave their opinions about 8 self-written error messages. These comments were concluded using thematic analysis. The last study combined information from the two previously executed studies and concluded views on the understandability of error messages, instead of readability.

Beneficial	Times appointed	Detrimental	Times appointed
More English in the error messages	29	No line number	39
Explanation of error	27	Verbose	22
Brevity	24	Poor wording	22

Figure 3: Overview of thematic analyses in P. Denny 2021's qualitative study.

3.2.1 Study 1: exploratory, quantitative study

For the exploratory, quantitative study on the readability of error messages, P. Denny 2021 collected the 60 most common error messages in Java, Python, and C. In the study, 33 novice programmers had to rate these common error messages on how easy to read they are using a scale from 1 to 10. The 60 error messages could then be sorted in a list from best to worst readable, to find the salient points that most affected the readability of the messages. In conclusion, it turned out that lots of jargon and "long" sentences mainly decreased the readability of error messages. The following two views from the study confirmed this conclusion:

- The top 10 best-read error messages contained no jargon. Seven out of ten messages consisted only of the phrase "[character] expected" without having any abbreviations or terminology alongside. On the bottom side of the list, there was much more jargon used in the sentences. The technical terms "EOF", "EOL", and "(triple-quoted) string literal" appeared five times in the bottom 10 error messages of the list.
- The top 15 best-read error messages of the list mainly consisted of only 21 characters (median), while the bottom 15 error messages contained approximately 44 characters (median). Which is more than twice as long.

The researchers concluded that jargon could be a big part of the reason why novice programmers didn't rate the messages too well for readability. The long sentences were also worse read which could have influenced the readability.

3.2.2 Study 2: qualitative study

In a second study conducted by P. Denny 2021 and his research group, 41 English-speaking novice programmers were investigated. All participants of the study saw a total of eight self-written error messages and reviewed those. The error messages differed in length, jargon usage, sentence structure, and vocabulary. Using guiding questions, participants were able to indicate whether they were missing information or found some information unnecessary and excessive in the error messages they saw. Using thematic analysis, the themes and patterns could be continued. Some common aspects that were most beneficial and detrimental for readability can be seen in Figure 3.

In conclusion, this study indicated that natural English language (rather than jargon) has a positive influence on the readability of error messages. Furthermore, short error messages in which an explanation is given of *what* and *where* an error occurs also benefit the readability.

3.2.3 Study 3: second quantitave study and combined conclusions

P. Denny 2021 lastly conducted another research, where information from the previous studies are processed in one more combined study. Also new practical research is done. This time, the research focuses on the comprehensibility (/understandability) of error messages under novice programmers, instead of the readability of error messages. Munsour 2017 describes the difference of these terms as follows. "Readability is the extent to which each sentence reads naturally, while comprehensibility is the extent to which the text as a whole is easy to understand."

In this study, the same error messages from the quantitative study are presented and rated based on how well understood they are. A score of 1 means that an error messages is barely understood, while a score of 10 represents a perfect comprehensible error message. Subsequently, a scatterplot was created (Figure 4) to compare the outcomes of the studies. In this figure the ratio normalized understandability and normalized readability of different errors is expressed. Each dot in the scatter plot is an error message, with the readability score from study 1 and the comprehensibility score from study 3 being considered. It can be seen that the terms understandability and readability have a strong correlation with each other.

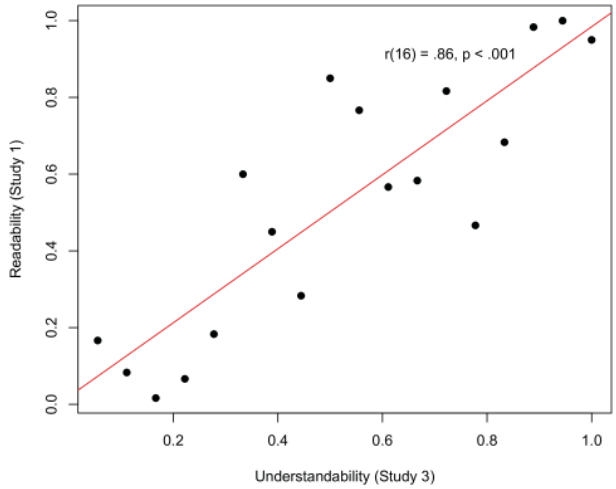

Figure 4: P. Denny 2021 created this scatter plot of normalized readability vs normalized understandability for common error messages in Java, Python and C. 0 means that the error message is most readable/understandable and 1 is least readable/understandable.

Finally, based on all three studies, this study also concludes about the influence of jargon, complete sentences, vocabulary, and economy of words on the comprehensibility of error messages. The following design criterion were created for readable error messages for novice programmers:

1. Jargon could best avoid error messages.
2. English sentences with a clear natural sentence structure are well-read.
3. Vocabulary in error messages should be simple.
4. Error messages should be concise.

3.3 Corrective feedback

Hattie and Timperley 2007 explain that "feedback" is the evaluation and analysis of one's behavior or performance. It can be valuable in spotting and improving errors and flaws. A subcategory of feedback is corrective feedback (CF), which specifically helps with redirecting students to a correct solution after making a mistake. CF could be useful to guide programming in the right direction while writing code. It can be provided in error messages since this is where the fundamental interaction between computers and people takes place.

3.3.1 Why corrective feedback

There are some more reasons why CF is useful for advising novice programmers. At first, Professor Jenkins 2019 stands behind the fact that CF "positively reinforces what a student is doing well while simultaneously indicating where there are clear areas of improvement". He explains that CF also ensures that responses to errors are not seen as criticism. This will make students react less defensively to the comments and learn more from them. Furthermore, Gayathri n.d. explains that CF is solution-oriented and focuses on "what a child can do". There is no emphasis on "what a child could have done", which has a motivating effect. Lastly, K. Guinness and Keyworth n.d. also state that students are also expected to respond actively to the feedback, which is an effective learning technique.

3.3.2 Corrective feedback strategies

Within the category CF, several different strategies can be used to provide helpful information. All of them can contribute to the correction of errors. There are 6 CF strategies. Tedick 1998 and Astia 2018 created a list to explain all strategies briefly. It is as follows.

1. **Explicit correction:** it is clearly emphasized that an error has been made, for example by repeating the error. An explicit correction of the error is mentioned next.
2. **Recast:** a wrong comment is immediately implicitly reformulated to the correct answer.
3. **Clarification request:** an error is emphasized with a phrase like *"I'm sorry?"* or *"I don't understand."*. No cues are given for improvement.
4. **Metalinguistic clues:** a mistake is subtly emphasized using a question or comment that hints at learned theory. An example: *"Is that an existing command?"*.

5. **Elicitation:** to emphasize an error an unfinished sentence or question is given that can be completed or answered by the student. An example: *"The python command with which you can display text is...?"* Student fills in: *"print"*.

6. **Repetition:** a complete comment is repeated, but this time with an explicit emphasis on the error. Written down, an error can be highlighted or underlined for example. The error is not corrected yet.

Ellis and Shintani 2014 and Mi n.d. state that the biggest differencebetweenf CF strategies are in the approach of the strategies. Some CF strategies can be classified as "input-providing", while others are "output-prompting". So some strategies will pre-chew the improvement of the mistake, while other strategies will encourage one to improve an error on its own. Besides, strategies can also be classified as "explicit feedback' and "implicit feedback'. Explicit feedback points to an error, while implicit feedback does not explicitly provide clues to the nature of an error. The classification of CF strategies can be seen in Figure 5.

CF strategy	**Input providing (IP)/ Output prompting (OP)**	**Implicit (I)/ Explicit (E)**
1. Explicit correction	IP	E
2. Recast	IP	I
3. Clarification request	OP	I
4. Metalinguistic clues	OP	E
5. Elicitation	OP	I/E
6. Repetition	OP	E

Figure 5: CF strategies are labeled as input providing or output providing, and implicit or explicit feedback. This tabular is created by Mi n.d. and supplemented with information gathered by Ellis and Shintani 2014.

3.3.3 Best corrective feedback strategies

Research published by Ellis and Loewen 2006 on the different effects of implicit and explicit feedback, concludes that explicit feedback is overall more effective than implicit feedback. Other research executed by Zhang and Chen 2021 shows that the effectiveness of each CF strategy varies by learning level. Upper intermediate level, for example, like repetition, while low intermediate learners mainly grow because of metalinguistic clues. This researcher also concluded that receivers and givers of feedback may have different preferences for CF strategies. When the preferences of the receiver and giver do not match, the effectiveness of the feedback reduces. Due to the differing opinions of learners and teachers, there is no standard ranking of what CF strategies benefit learning most.

4 Methodology

4.1 Research design

Novice programmers are able to read error messages in the gradual programming language Hedy to learn about problems in programmed code. But not much research has yet been done about the comprehensibility and clarity of the sentence structure of error messages in this gradual programming language. Therefore it is now discussed if this sentence structure is actually clear to novice programmers, or if some changes can still be made to increase the positive effect of the error messages. The following research questions are posed and investigated through practical research:

RQ1: What are error message needs of novice programmers in a gradual programming language?
RQ2: Which corrective feedback strategy can best be incorporated into Hedy's errors to increase the clarity and comprehensibility of the error messages?

Aspects of CF may be incorporated to improve Hedy's error messages, since CF has a positive effect on learning results. It is checked if this benefits the comprehensibility and clarity of error messages. In preparation for the study, six different types of error messages were created, based on six different types of corrective feedback. These and Hedy's current error messages were then presented to a group of novice programmers to evaluate during their programming lessons at school. Each participant of the study only saw one type of error message and several example error messages from this type which belonged to a erroneous Hedy code. They could evaluate the errors they saw, by answering questions about the clarity and comprehensibility of the wording during an individual oral interview. During this interview, there was also room for general opinions and comments about the errors or error messages. With this, qualitative and categorical data was collected through individual interviews with the novice programmers.

After the interviews, novice programmers were observed while they looked at that same erroneous Hedy code without any restrictions or questions to answer. They could rewrite it to working code with the help of the error messages they just evaluated. This step is only added as a check to see if the research was understood correctly. If no errors could be corrected at the end of the research, the data could be considered unusable.

The overall steps taken for this research are summarized in Figure 6.

4.2 Research methods

The first part of the study (where novice programmers evaluated errors through interviews) can be described as an structured interview. Research is carried out in an objective and controlled fashion. The second part of the practical study (where novice programmers improved their erroneous Hedy code) is an observational study. Their comments and way of working were analyzed afterward, with the use of audio fragments.

Conducting interviews and creating room to look at erroneous code individually, seemed to be

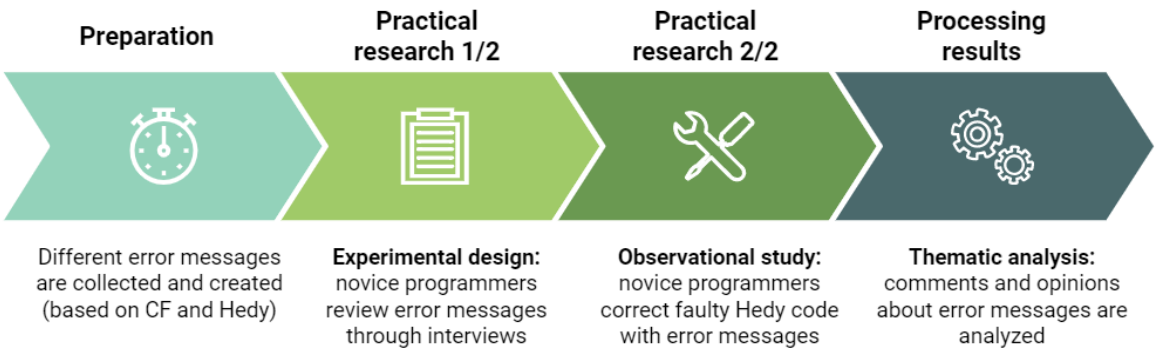

Figure 6: All methodology steps of this study.

the most suitable approach to find answers to the research questions. Comments of participants were processed through thematic analysis. These themes accurately reflect the meanings evident in the data set to maintain the validity and reliability of this type of research.

4.3 Data collection

In total, fourteen students individually took part in the interviews where they evaluated errors and error messages. The faulty Hedy code containing the errors can be seen in section 4.5.1. The code was displayed in Hedy level 12. All participants understood Hedy theory from level 1 to 12 prior to starting the research. The participating students were all of ages eleven and twelve. Their school level was either Havo or Havo/VWO. All participants were in seventh grade and followed the same programming course taught by the same teacher. Six out of fourteen students identified themselves as boys and the other eight students identified as girls. Differences in age, gender and school level became negligible for further analysis because of the relatively small scale of this study.

The conducted individual interviews lasted in between 8 minutes and 43 seconds, and 13 minutes and 26 seconds. The difference in duration was caused by the fact that some students were simply more talkative than others. During the interview multiple choice questions and open questions were proposed via a Google Form (moreover in section 4.5.3). Other comments during the interview were recorded via audio recordings. The structure of all interview questions was exactly the same. Only the error messages shown differed. In total there were seven different types of error messages and Google Forms to evaluate (six based on different CF strategies and one showed Hedys errors). So each Google Forms could be completed by two of the fourteen participants.

The group high school students who participated in this study took lessons at Lyceum Kralingen, located in Rotterdam, the Netherlands. They were contacted through Felienne Hermans who also worked at this school. All interviews took place at Lyceum Kralingen as well, during two coding lessons.

4.4 Analysis

Thematic Analysis is used to analyze audio fragments made during the interviews in the practical study. This is an unsupervised approach that allows one to perform statistical tests without any rules or procedures in advance. The analyzing method is inductive, so the theory about clear and understandable error messages in Hedy can be built from scratch.

The information about the multiple choice questions could easily be converted through bar charts. This showed how many times answers were given in relation to each other.

4.5 Materials

In this methodology, there are several references to an erroneous Hedy code. This can be seen in section 4.5.2. There are also some references to the different types of error messages and differences in Google Forms. Examples of those can be seen in section 4.5.2. The full Google Form layout is explained in section 4.5.3.

4.5.1 Erroneous Hedy code

The erroneous Hedy code that all students saw during the practical research is shown in Figure 7. Six out of seven mistakes in the code cause immediate errors in Hedy (outlined with yellow). The seventh mistake is a typo, which causes problems too (outlined with green). An example of a correction of this code can be seen below the erroneous code in Figure 8. All errors of the code are explained in Figure 9.

4.5.2 Different types of error messages

The students who participated in this study all saw the same code mistakes, but several different error messages. In total, the fourteen participants were divided into seven subgroups consisting of two individuals. Each group was presented with a different type of error message. Six of the seven observation groups saw and rated errors based on different forms of corrective feedback. The seventh group acted as a control group and analyzed the error messages currently displayed in Hedy for these errors. An example of the different types of error messages can be seen in Figure 10. This figure shows the different errors that were shown with the code mistake in the second line of the erroneous Hedy code. The table also shows a 'code' for each error message type. This is the abbreviation used to easily reference the viewed error message type. The codes will be used throughout the rest of the report.

4.5.3 Interviews

There were seven surveys in total, each used to accompany two interviews. The interviews contained of the same questions but showed different error messages. In the survey, students first had to indicate per line whether they spotted a code mistake in the erroneous Hedy code at all. The true errors were then emphasized with their corresponding error message. Students could express their opinion about the wording or defects of the error message, with the help of two multiple choice questions. These were: "Did you understand the error message?" and "How clear is the error message?". Per code mistake, there was also room to answer the open question:

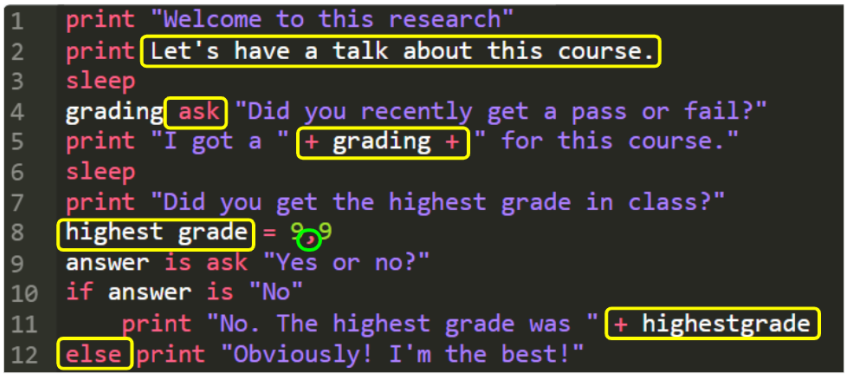

Figure 7: Hedy code with seven outlined code mistakes. This code is presented in Hedy level 11.

```
print "Welcome to this research"
print "Let's have a talk about this course."
sleep
grading is "pass"
print "I got a " + grading + " for this course."
sleep
print "Did you get the highest grade in class?"
highestgrade = "9.9"
answer is ask "Yes or no?"
if answer is "No"
    print "No. The highest grade was " + highestgrade
else
    print "Obviously! I'm the best!"
```

Figure 8: Corrected Hedy code.

Line 2	There are missing quotationmarks around the outlined text.
Line 4	The word ‘is’ is missing here before the outlined command ‘ask’.
Line 5	’grade’ is an input from ask, which is a different type than text/string. Therefore it may not be added to a text/string. Update: Since ...., if the input of ask is still a text, it can be added with other strings.
Line 8	Code mistake 1: A variable name cannot contain a space.
Line 8	Code mistake 2: The circled comma must be a period. A comma between two numbers creates a list, but in this text the numbers must create a single decimal number (9.9). This does not give an official Hedy error, but it is a crucial typo.
Line 11	‘the highest number’ is a list or number/float (depending on whether the code mistake from line 8 has already been changed) and may therefore not be added with a text/string.
Line 12	The text after the outlined ‘else’ must be indented with 4 spaces on the next line.

Figure 9: The code mistakes from Figure 7 explained.

Code	Error message type	Error message shown for the code mistake in line 2 (in Figure 7)
CF1	Explicit correction	You forgot to add something to the text. Add ' before and after the text.
CF2	Recast	Correct code: print "Are you good at programming?"
CF3	Clarification request	I don't understand this.
CF4	Metalinguistic clues	How does the print command work?
CF5	Elicitation	Before and after the printed text is still missing a ...?
CF6	Repetition	print Are you good at programming?
Hedy	Hedy	The code you entered is not a valid Hedy code. There is a mistake on line 2, at position 32. You typed a question mark, but that is not allowed in that place.

Figure 10: Different error messages per participating subgroup.

"What would you consider a better error message?". This could be answered written down or verbally. The complete surveys can be found via the links in Apendix 8.1.

5 Results

For this study, individual interviews were conducted with fourteen novice programmers to measure the comprehensibility and clarity of certain error messages. During these interviews, two participants assessed several error messages of one specific type of error message. Two multiple-choice questions and one open question were answered per student and there was room for other comments. For reasons of privacy and clarity, these results refer to the students with numbers (1.1, 1.2...) instead of real names. The numbers also include the type of error message that was read and whether this was the first or second interviewee within this category. In Figure 11 the complete overview with the student numbers is shown.

Students	Read error messages based on...
STUDENT 1.1; STUDENT 1.2	CF1 - Explicit correction
STUDENT 2.1; STUDENT 2.2	CF2 - Recast
STUDENT 3.1; STUDENT 3.2	CF3 - Clarification request
STUDENT 4.1; STUDENT 4.2	CF4 - Metalinguistic clues
STUDENT 5.1; STUDENT 5.2	CF5 - Elicitation
STUDENT 6.1; STUDENT 6.2	CF6 - Rehearsal
STUDENT 7.2; STUDENT 7.2	Hedy

Figure 11: Student numbers of participants of the study.

5.1 Multiple choice questions

Reminder:
This study showed an erroneous Hedy code, containing six official errors, and an "unofficial" (see section 4.5.1). Because of the unofficial error, there are only six error messages for STUDENT 7.1 and STUDENT 7.2 who assessed Hedy's errors. The other participants saw seven error messages.

The multiple choice question "do you understand the error message?" was answered a total of twelve times after seeing Hedy's error messages (by two students, after six error messages). So there are twelve data points about this labeled comprehensibility. The same question was answered fourteen times per error message type based on CF (by two students per error message type, after seven error messages).

Figure 12 shows per error message type (column) the proportion to which all multiple choice answers have been given (colors in the column) to the title question in the figure. The x-axis of the columns shows the code of each error message type. The y-axis of the graphs is expressed in percentages to make it easier to compare the different amounts of data points (twelve or fourteen) of the answers.

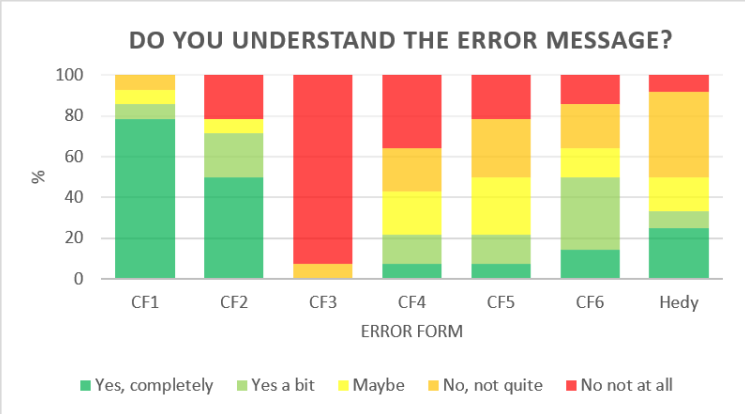

Figure 12: Relative, graphical representation of comprehensibility-labeling (colors on the columns) per error messages type (column). All bars represent fourteen answers to the title question, while "Hedy" represents twelve answers.

Figure 13 shows the column graph that represents how the second multiple-choice question was answered. Fourteen participants in the study indicate how clear they find the error messages shown to them. The graph has the same x- and y-axis as Figure 12.

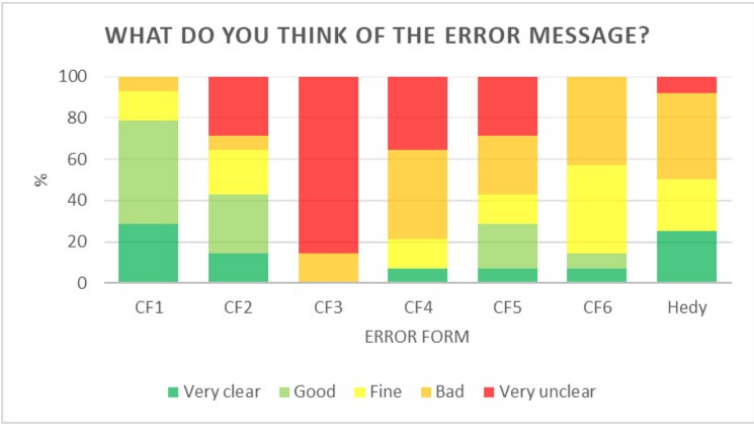

Figure 13: Relative, graphical representation of clarity-labeling (colors on the columns) per error messages type (column). All bars represent fourteen answers to the title question, while "Hedy" represents twelve answers.

5.2 Open question and thematic analysis

In total, there were seven repeating themes in the comments novice programmers made about (adjustments for) error messages shown in Hedy. These comments were a response to the error messages evaluated. All feedback was given after answering the survey question: 'what would you consider a better error message than the one you're viewing right now?'. How often each theme came up and by whom it was mentioned can be seen in the tabular below. The tabular only shows error message adjustments that were named four or more times.

Error message adjustment:	**Times mentioned:**	**Mentioned by (times mentioned):**
Repeat theory	13x	2.1 (I), 3.1 (I), 3.2 (II), 4.1 (IV), 4.2 (II), 5.1 (I), 5.2 (I), 6.1 (I)
Show direct correction	12x	2.1 (I), 2.2 (II), 3.1 (II), 3.2 (I), 4.1 (I), 4.2 (I), 5.1 (I), 6.2 (I), 7.2 (I)
Mention location/line number	11x	1.1 (IV), 1.2 (I), 3.2 (I), 4.1 (I), 5.1 (I), 5.2 (I), 6.1 (II)
More visuals	8x	2.2 (I), 3.2 (II), 4.1 (I), 5.2 (I), 6.1 (II), 6.2 (I)
Use talking language	7x	3.1 (III), 3.2 (I), 4.1 (I), 4.2 (I), 5.1 (I)
Shorten error message	6x	1.2 (I), 2.1 (III), 7.1 (I), 7.2 (I)
Use examples	6x	2.2 (I), 3.2 (I), 5.1 (II), 6.2 (I), 7.1 (I)

These most frequently mentioned error message adjustments are explained below.

Repeat theory

Participants reviewing metalinguistic clues (CF4) only saw the error message: "I don't understand this" after all errors. This led to one of the students showing visible frustration. Both participants mentioned several times that this error message was not helpful because it missed an explanation of what was wrong. Error messages containing recast (CF2) or rehearsal (CF6) also did not show why there was a mistake. Error messages of CF2 showed an example of fully corrected code, without the mistake emphasized. Error messages of CF6 only emphasized where the mistake was, without hinting towards a valid correction of the mistake. Some reactions students gave to their error messages were:

> *The error in line 12 is shown.*
> STUDENT 2.1: *"[The error message] should tell that an else-command always starts with 4 spaces."*

> *The error in line 2 is shown.*
> STUDENT 6.1: *"'Text after print must always be between two high brackets' can be a better error message."*

Show direct correction

In total there has been an improvement in error messages twelve times in the form of direct correction. There was a lot of need for sentence structures such as "turn ... into ..." or "... is missing here".

At some point, there is an error in the erroneous Hedy code, where a list is created instead of a decimal number. ("9,9" is typed instead of "9.9".) Both students who evaluated recasts (CF2) saw the error message "Correct code: highest mark = 9.9" and said they wanted to change this error. The error message shows an implicit correction of the minor error, but as a result, the

adjustment to create a better code is not noticeably highlighted. Both students wanted a more explicit correction of the error. According to STUDENT 2.1, the following would have been a better error message:

> STUDENT 2.1: *"Just say: 'Change the comma to a period.'"*

Mention location/line number

Seven participants of this research mentioned that they missed the location of an error in their error messages. None of them were in the group reviewing Hedy's error messages. These error messages already showed the line and characters of the mistakes. Direct words of a participant reviewing errors based on explicit correction were:

> STUDENT 1.1 (CF1): *'Maybe they could add the line number.'*
> RESPONSE SUPERVISOR: *'Why?'*
> STUDENT 1.1 (CF1): *'I don't know where to look if there is more text on the screen. ... I see it now because there is only one line. But sometimes I don't see it easily.'*

More visuals

Some participants expressed that they wanted to see more visuals pointing to their errors. The two novices that already saw error messages with highlighted text also responded positively to the colors. One said:

> STUDENT 6.2: *"This error is bad because there is no explanation. ... But it does grab the attention, so that's very good."*

One student who only saw the error "I don't understand this" again, offered the following:

> STUDENT 2 (CF3): *'I don't see the error. [The error message] could point at the error with an arrow maybe.'*

Use talking language

There was much preference for spoken language. Seven students adjusted errors to a sentence consisting of natural language. This is what some of the improvements looked like:

> STUDENT 3.1: *'Maybe the error message could start with "Look closely!"'*

> STUDENT 4.2: *'Hmmm, add "You forgot something!"'*

> STUDENT 5.1: *'It could say: "Try thinking of something else".'*

Shorten error messages

Hedy's error messages, error messages with recasts (CF2), and error messages containing explicit

correction (CF1) mainly showed error messages with two or three sentences. From the people evaluating these messages, four out of six participants commented at least once that they found the error messages long. Also, the error messages from Hedy were not read out in full by both students during the evaluation. The sentence: "The code you entered is not a valid Hedy code" appeared in three Hedy errors but was not read aloud once whenever the rest of the error message was stated out loud.

Participants who read other error messages mainly saw one sentence per error message. None of these eight novice programmers mentioned anything about the length of the sentences.

Use examples

Participants of this study had just learned the theory about the plus-symbol in programming. One participant explicitly stated that this new theory could best be introduced with more examples in the error messages because that's how they always learn new theories.

> STUDENT 6.2: *I don't get it yet. Can I see an example?*

Two other participants, STUDENT 2.2 and STUDENT 7.1 also asked for examples of how to fix certain code a few times.

6 Conclusion

This research seeks an answer to the question: *what are the error message needs of novice programmers in a gradual programming language?* Relying upon the literature research and practical research of the study, the following conclusions can be drawn.

The literature research concluded that error messages should, among other things, be helpful, constructive, comprehensible, and precise, to be well understood by novice programmers (see section 3.1). Aspects that derogate the comprehensibility of error messages are complexity and tediousness. But too difficult solutions for small errors also cause confusion and discouragement when learning to program (see section 3.1). According to previous research on the readability of error messages among novice programmers, the use of a lot of jargon and long sentence structure works counterproductive for the readability of error messages. Concise error messages with simple vocabulary and no jargon are best read by novices whenever the sentences also have a natural English sentence structure (section 3.2). Corrective feedback can be processed in error messages, as this positively reinforces student learning while indicating clear improvements in the learning process. Corrective feedback strategies can be input-providing, output-prompting, implicit, or explicit. There is no right order in which certain strategies can be ranked from best to worst, but explicit feedback seems to overall be more effective than implicit feedback. (section 3.3)

In a practical study where fourteen novice programmers could evaluate error messages, new conclusions about the influence of corrective feedback in error messages can be drawn. The second research question is repeated below and answered using the results of this thesis.

RQ2: "Which corrective feedback strategy can best be incorporated into Hedy's errors to increase the clarity and comprehensibility of the error messages?"

First of all, error messages are poorly understood and unclear to novice programmers when based on the strategy "clarification requests" (Figure 12 and Figure 13). On the other hand, errors with "explicit correction" are best rated for comprehensibility and clarity according to novice programmers (also shown in the same figures). Recasts are also understood very often. Besides error messages based on these feedback types, Hedy's current errors appear to have relatively few improvements. Now that this is known, the main research question can be answered.

RQ1: "What are the error message needs of novice programmers in a gradual programming language?"

Error messages in gradual programming languages can best contain the CF strategy explicit correction, to be comprehensible and clear according to novice programmers. This has been the best-rated CF strategy in this study. Improved error messages could also repeat theory, show direct correction of mistakes, mention the location of a mistake, use talking language, and/or show examples to be as comprehensible and clear as possible. The error messages could be highlighted or emphasized with other visuals to grab more attention of novices. And lastly, error messages should be short (less than two sentences) to be comprehensible.

7 Discussion

7.1 Interpretation theory, results and conclusion

Many of the same views emerge from the theoretical and practical research that answer the main research question of the study. When looking at the clarity and comprehensibility of error messages, both theory and practical research confirm that long sentence structures in error messages work counterproductive. Concise error messages attract the attention of novice programmers better. It also follows from both literature and practical research that explicit CF forms are more effective overall than CF forms containing implicit feedback.

Besides these same conclusions, the theoretical and practical research answer the main question differently. The theoretical research mainly emphasizes the *properties* that error messages can best have, such as: being helpful, constructive, comprehensible, and precise. While in practice research more attention is paid to what *content* in error messages is important. Additional examples, spoken language, highlighted text, repeated theory, and location of errors, for example, strongly contribute to the comprehensibility and clarity of the error messages.

7.2 Differences theoretical framework and practice research

The theory and the practical research differ mainly in terms of one characteristic that error messages should not have. It is explicitly mentioned in the literature that jargon in error messages deteriorates the readability of error messages. But this remark is relatively rare in practical research. This is probably because the Hedy programming language doesn't use a lot of jargon that novice programmers can struggle with in the first place-. Common abbreviations in other high-end, non-gradual programming languages, such as "EOL" or "EOF", do not occur in this programming language for example.

7.3 Validity results

Even though this study draws some clear conclusions, there are also several factors in this study that limit the validity of these results.

First of all, this study used a research method that can be labeled as subjective. The research method of this study is thematic analysis. This is a qualitative research method where conclusions are drawn from children's comments. Recurring, overarching themes are collected with the help of human reasoning, which weakens the validity of the results.

The opinion of children is also plausible in some cases. Children (and people in general) tend to give socially desirable answers, rather than their own opinions in some cases. When asked if an error message is bad in their opinion, the answer can be "no" to show respect for the designer of the error messages. Criticism or improvements remain unmentioned this way.

Lastly, there is also a theoretical weakness in this research. There are relatively few scientific studies done surrounding the same research question. For example, reference was made to three studies on the readability of error messages, but all of these studies were conducted by the

same research group. Fewer different sources weaken the reliability of the research.

7.4 Future work

More research can be done on this research topic. Error messages needs in other programming languages can be traced for example. But for research in Hedy, this study also offers scope for follow-up research. Two suggestions for further research are mentioned down below.

7.4.1 Implementation of new error messages in Hedy

This research suggests that explicit correction can be incorporated more into error messages to improve the comprehensibility and clarity of the error messages. These new error messages in Hedy have not yet been created. In a follow-up study, the conclusions drawn here can be implemented in new error messages in Hedy. For example, the most common error messages in the first 5 levels of Hedy can be rewritten as the first step in upcoming research.

7.4.2 Visuals for error messages

The thematic analysis of this study showed that there is a demand for more visuals with the error messages. There isn't much more debt going into this observation within this thesis. In a follow-up study, it can be determined which visuals best emphasize programming errors. Lines or words can be highlighted, arrows can be used to point at errors, or the image in which error messages are displayed can shake to attract attention, for example. Several visuals can be tried out in the Hedy environment, to see which error messages can best be emphasized for novice programmers. Through new surveys for Hedy users, conclusions can be drawn on which visual effects attract the most attention.

References

1. Astia, M. (2018). "Corrective feedback in English class". MA thesis. Samarinda: Mulawarman University.
2. Bakker, T. (2021). "Hedy programming language introducing the Gradual Feedback Model (GFM)". MA thesis. Leiden: Leiden University.
3. Becker, B. A. and Bouchard, D. (2019). "Compiler Error Messages Considered Unhelpful: The Landscapeof Text-Based Programming Error Message Research". MA thesis. Dublin, Ireland: University College Dublin, Roanoke College.
4. Bennedsen, J. and Caspersen, M.E. (2007). "Failure rates in introductory programming". In.
5. Birkett, A. (n.d.). "Error Messages: Examples, Best Practices Common Mistakes". In: ().
6. Ellis, R. and Loewen, S. (2006). "Implicit and explicit corrective feedback and the acquisition of L2 grammar". MA thesis. Cambridge University Press.
7. Ellis, R. and Shintani, N. (2014). *Exploring Language Pedagogy through Second Language Acquisition Research*. Routledge.
8. Ewerlöf, A. (n.d.). "What makes a good error message?" In: ().
9. Gayathri, V.S (n.d.). "The Importance of Corrective Feedback!" In: ().
10. Hattie, J. and Timperley, H. (2007). "The Power of Feedback". In: *Sage journals*.
11. Hermans, F. (n.d.). "Hedy - A gradual programming language". In: ().
12. Jenkins, S. (2019). "The Importance of Corrective Feedback". PhD thesis. College of Education Westcliff University.
13. K. Guinness, R. Detrich and Keyworth, R. (n.d.). *Overview of Corrective Feedback*. Oakland, CA.
14. McGough, O. (2013). "How to: Error Messages". In: *Get Feedback*.
15. Mi, C. (n.d.). "Top Strategies for Positive Corrective Feedback: Part 1". In: ().
16. Minhas, S. (2008). "How to Write Good Error Messages". In: *UX planet*.
17. Molich, R. (n.d.). "Error messages".
18. Munsour, E. (2017). "Readability and Comprehensibility of Patient Information Leaflets for Antidiabetic Medications in Qatar". In: *Natural Library of Medicine*.
19. Nielsen, J. (2001). "Error message guidelines". In: *Nielsen Norman Group*.
20. P. Denny J. Prather, B. Becker (2021). "On Designing Programming Error Messages for Novices: Readability and its Constituent Factors". In: *CHI Conference on Human Factors in Computing Systems*, pp. 1–15.

21. R. C. Birney, H. Burdick (1969). *Fair of failure*. Van Nostrand-Reinhold Co.

22. Schute, V. J. (n.d.). "Focus on Formative Feedback". In: *Sage journals* 78 ().

23. Shneiderman, B. (n.d.). "Designing computer system messages". In: ().

24. Tedick, D. J. (1998). "Research on Error Correction and Implications for Classroom Teaching". MA thesis. Minnesota: University of Minnesota.

25. Zhang, T. and Chen, X. (2021). "EFL Students' Preferences for Written Corrective Feedback: Do Error Types, Language Proficiency, and Foreign Language Enjoyment Matter?" MA thesis. Chengdu, China: School of Foreign Languages, University of Electronic Science and Technology of China.

8 Appendix

8.1 Appendix 1: surveys

Links to the surveys used in the practical study of this research:

- Survey for participants who reviewed CF1: https://forms.gle/kr9qdCrFQN1uSM2H7
- Survey for participants who reviewed CF2: https://forms.gle/pB9YHhEhJZyKFebe9
- Survey for participants who reviewed CF3: https://forms.gle/u9XyuRyf9c489CSr7
- Survey for participants who reviewed CF4: https://forms.gle/Lg3VgsZ6qq6dbfM58
- Survey for participants who reviewed CF5: https://forms.gle/9qT1NpFVVrmanFu59
- Survey for participants who reviewed CF6: https://forms.gle/ctvRAKVTvsjuQgXC7
- Survey for participants who reviewed Hedy: https://forms.gle/2tpq65hoPHTXiJSe9