On Designing Programming Error Messages for Novices: Readability and its Constituent Factors

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

James Prather
Abilene Christian University
Abilene, Texas, USA
james.prather@acu.edu

Brett A. Becker
University College Dublin
Dublin, Ireland
brett.becker@ucd.ie

Catherine Mooney
University College Dublin
Dublin, Ireland
catherine.mooney@ucd.ie

John Homer
Abilene Christian University
Abilene, Texas, USA
john.homer@acu.edu

Zachary Albrecht
Abilene Christian University
Abilene, Texas, USA
zca16a@acu.edu

Garrett Powell
Abilene Christian University
Abilene, Texas, USA
gbp18a@acu.edu

ABSTRACT

Programming error messages play an important role in learning to program. The cycle of program input and error message response completes a loop between the programmer and the compiler/interpreter and is a fundamental interaction between human and computer. However, error messages are notoriously problematic, especially for novices. Despite numerous guidelines citing the importance of message readability, there is little empirical research dedicated to understanding and assessing it. We report three related experiments investigating factors that influence programming error message readability. In the first two experiments we identify possible factors, and in the third we ask novice programmers to rate messages using scales derived from these factors. We find evidence that several key factors significantly affect message readability: message length, jargon use, sentence structure, and vocabulary. This provides novel empirical support for previously untested long-standing guidelines on message design, and informs future efforts to create readability metrics for programming error messages.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**; **HCI theory, concepts and models**; *Empirical studies in HCI*; • **Social and professional topics → Computing education**; *CS1*; • **Software and its engineering → Error handling and recovery**; *Compilers.*

KEYWORDS

compiler design and implementation; compiler error messages; CS1; HCI; human computer interaction; introductory programming; novice programmers; programming error messages; students; readability

ACM Reference Format:
Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary Albrecht, and Garrett Powell. 2021. On Designing Programming Error Messages for Novices: Readability and its Constituent Factors. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3411764.3445696

1 INTRODUCTION

Feedback is a fundamental aspect of human-computer interaction, defined by Shneiderman as communication with a user resulting directly from the user's action [54]. For programmers, this communication often takes the form of error messages generated by a compiler resulting from the action of building (or compiling) code in preparation for execution. And yet, feedback provided in the form of compiler error messages has been notoriously poor for decades. In 1983, Brown stated in *Communications of the ACM*: "One of the most important yet most neglected aspects of the human/machine interface is the quality of the error messages produced by the machine when the human makes a mistake" [15, p246]. This, we argue, is still true today. A recent case study revealed that developers at Google found some error messages to be "confusingly worded," they spent a median of 12 minutes resolving each one, and such errors are common – affecting nearly 30% of all builds [52]. Even more recently, compelling evidence of the need for improving programming error messages was provided by Barik et al., who used eye-tracking to discover that student software developers allocate 25% of their fixations to error messages [6]. While experienced developers may find poor error messages an irritation, for novices they can be paralyzing [5].

In educational contexts, the detrimental effects of cryptic error messages on novice programmers have been reported throughout the literature for decades [12, 20, 26, 31, 32, 34–36, 41, 60]. Student frustration is exacerbated because such feedback is provided regardless of whether an instructor is present and comes from a machine that many novices see as infallible [36]. Educators are affected indirectly as they must devote time to helping students correct programming errors when the messages cannot be understood [7, 17, 21, 25, 56]. As early as 1965, systems were beginning to be developed that provided students with detailed error messages as this was seen as essential [51]. Nevertheless, educators and researchers are still documenting the many problems that poor programming error messages present to students [9].

Improving programming error messages is difficult. Becker et al. advocate that improving error messages after they are issued by the compiler but before they are seen by a user – commonly called Enhanced Compiler Error Messages (ECEMs) [8] – is only a makeshift fix and that improving messages from first-principles is likely a better solution [9]. Aside from the technical challenges around accurate error diagnosis and localization [49], there are many ways in which the description of an error can be presented to a programmer. There are at least two basic levels at which error messages can be assessed – usability and readability. In this paper we focus on readability which, we argue, is more fundamental. A message that is unreadable is likely unusable. Additionally, a very easy to read message isn't necessarily usable (e.g. a readable message suggesting a wrong solution). Specifically, readability is necessary but not sufficient for usability. We view usability as how actionable a message is, and this carries with it certain context (such as error locality and message precision) that is independent of message readability.

To date, the readability of programming error messages has received very little attention. Papers from the 1970s onward that provide any guidelines whatsoever for designing or evaluating programming error messages have almost unanimously listed 'readability' or some synonym as an essential guideline [4, 9, 29, 40, 42, 53, 58]. However these guidelines are vague, often conflicting, and rarely backed up with empirical evidence [9]. In all of this prior work, the word 'readability' is either undefined, or uses the same definition as the readability of natural languages. The readability of natural language prose is quite well researched, but most obviously not the same. Brown seems to echo this when he wrote in 1982 [14, p94]: "Most of us who write systems do not produce good error messages; we produce shoddy ones. Just because error messages are in some approximation to natural language, we claim that they are easy to understand. We deceive no-one but ourselves." Almost forty years later, this problem remains: there is still no definition of what makes a programming error message "readable," and therefore no usable metric for assessing message readability, unlike the variety that exist for natural languages. A recent systematic review of the literature on programming error messages by Becker et al. concludes with a specific call for research on readability, as a prerequisite for improving message design [9].

In this paper, we explore the characteristics of programming error messages that impact their readability for students learning to program. We report the results from three experiments, the first two of which inform the design of the third. Each experiment involved a distinct cohort of novice programming students from institutions in North America, China, and Australasia. Our rationale for diverse recruitment was two-fold: to avoid survey fatigue, and to strengthen external validity. The first experiment was a quantitative study, in which participants rated the readability of the most common error messages from three popular languages on a numeric scale. By ranking these messages from most to least readable, we identified the most relevant potential factors among those that had previously been suggested in the literature. The second experiment was a qualitative study where factors were elicited directly by showing participants a set of messages and asking them what made the programming error messages easy or difficult to read. A number of common themes were identified by coding responses from students across two countries. Those themes that overlapped with factors from the first study were of particular interest. Finally, in the third experiment, participants were asked to rate a subset of the messages from the first study according to a fixed set of criteria derived from the first two studies. Figure 1 provides an overview of these three experiments. Across these related experiments, we address the following research questions:

- **RQ1:** What factors affect the readability of programming error messages for novices?
- **RQ2:** To what extent does each factor have an impact on message readability?

We provide three important contributions in this paper. First, we report on the results of multiple studies aimed at discovering which characteristics impact the readability of programming error messages for novices. This is the first work to empirically identify such factors. Second, we discuss how these characteristics can inform the design of more readable error messages. Third, we find several insights that can help guide future efforts to design a readability metric for programming error messages.

We begin by considering the literature related to programming error messages, their design, empirical studies for enhancement, and how a lack of definition of readability has frustrated those efforts (Section 2). Next, we report on Study 1, which was an exploratory study to begin uncovering why some messages are perceived as more readable than others (Section 3). We then report on Study 2, a qualitative effort to elicit from novices the factors they believe improve or impede readability (Section 4). Study 1 and Study 2 led us to design Study 3 (see Figure 1), which follows and confirms the factors we empirically discovered as being direct characteristics of readability for programming error messages (Section 5). We discuss these results and their implications for design (Section 6). We address some limitations in our work (Section 7), then summarize our findings and conclude with a clear call for future work (Section 8). To aid the reader, Table 1 provides a high-level summary of the three studies and their key results.

2 BACKGROUND

From both HCI and pedagogical points of view, compiler error messages play a unique role in programming and learning to program. They are integral to the input/output loop between programmers and programming environments. However, for more than fifty years educators and researchers have been documenting the difficulties that programming error messages present, particularly to students

Table 1: Summary of studies and key results.

Study	Participants	Method summary	Key results
1: Exploring Message Readability	33 (US)	Participants rated the *readability* (on a 1–10 scale) of the 20 most frequently occurring error messages in three popular languages (Java, Python and C)	Two initial factors emerged: shorter messages tended to be rated as more readable; messages containing jargon and technical terms tended to be rated as less readable
2: Eliciting Readability Factors	114 (US, China)	Participants were shown 8 error messages (used in a prior study on message readability) and asked to describe what makes each message easy or difficult to read	Thematic analysis revealed four factors relating to readability: length, jargon, sentence structure and vocabulary – the latter of importance to non-native English speakers
3: Confirming Readability Factors	95 (Australasia)	Participants were shown 18 messages (selected from Study 1) and rated each (on a 1–5 scale) against the four factors identified in Studies 1 and 2, and a holistic rating of *understandability*	Ratings for all four factors strongly correlated with understandability ratings, which in turn were highly correlated with ratings of readability from Study 1, providing an empirical basis for design guidelines

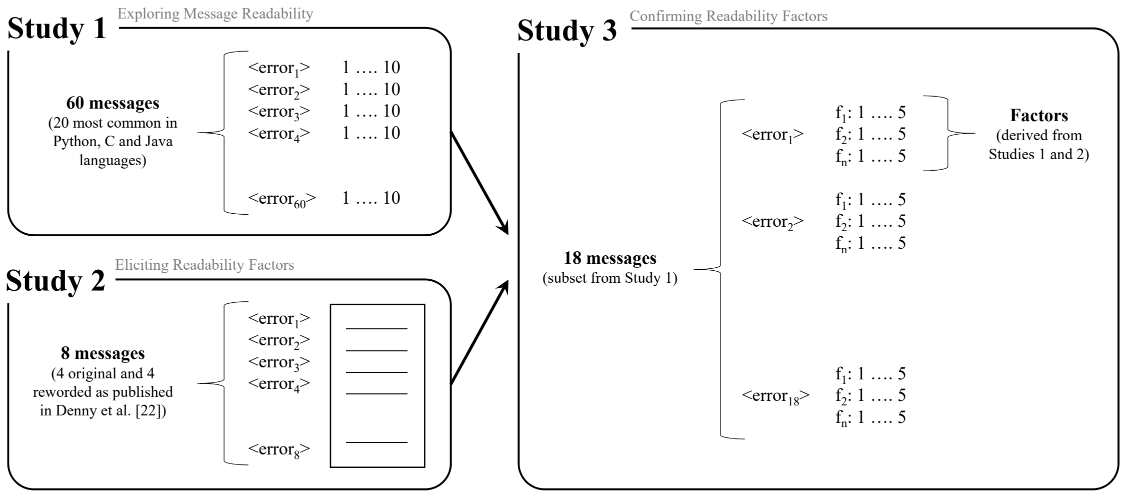

Figure 1: In Study 1 (quantitative), participants rated common error messages using a holistic numeric scale. In Study 2 (qualitative), participants were asked what affected message readability. In Study 3, messages were rated using scales derived from common themes emerging from Studies 1 and 2.

[4, 29]. In a recent landmark review of the literature on programming error messages, Becker et al. surveyed 107 papers dating back to 1967, finding that guidelines for authoring programming error messages were rarely supported by robust evidence and were often entirely anecdotal [9]. They found more than half of the reviewed papers that presented guidelines mentioned *readability* as a core component of good message design, yet no metric exists for assessing such readability. Although similar metrics do exist for natural language prose and even programming source code, these are not suitable for application to error messages. Becker et al. conclude their review by calling for research that focuses on understanding and assessing readability, claiming it is fundamental for future efforts to improve the design and effectiveness of programming error messages.

2.1 Motivating example

To illustrate how error messages can be problematic for novices, consider a hypothetical student, May, who is interested in becoming a civil engineer and is required to take an introductory C programming course at her university. With no prior experience in programming, she is learning by carefully copying examples from the textbook and observing the behaviour of the running programs. She has just finished typing in the source code for a program that converts temperatures from fahrenheit to celsius, and she is now ready to run the program to perform some conversions. Unfortunately, she has made a small typographical error by forgetting to include a comma between two identifiers ('fahrenheit' and 'celsius'). When she tries to build the program, her compiler (*gcc*) issues an error message. The message reads: "*error: expected '=', ',', ';', 'asm' or '__attribute__' before 'celsius'*". This is the beginning in a series of interactions between May and her compiler, the first three of which are shown in Table 2. In Step 1, the line of code that contains the syntax error is shown, along with the generated error message. On the surface, the message is not easy to read, however it suggests that some type of symbol is "expected" before the identifier 'celsius', and several possible symbols are shown. May selects the first one listed, the '=' symbol, and places it between the two identifiers (as shown in Step 2) before building her code again. This time, a new error message suggests that the identifier 'celsius' is "undeclared". This message is short, but not descriptive of a solution using language with which May is familiar. Searching for the error online, May finds a simple fix – the identifier should be defined prior to the statement – and she has seen examples of this in her coursebook. Doing so, she finds the compiler issues yet another message, leading May even further from the simple solution to her original typographical error – the missing comma.

Such interactions are all too common. Denny et al. document a case of a student spending more than two hours in a similar kind of interaction loop with a compiler, without success, before abandoning their session [19]. In our own experience, students have responded to surveys about learning to program with disparaging statements about error messages, including: "Absolutely losing my mind trying to figure out what the error messages are asking me to do." There is even documented evidence of programming error messages being a contributing factor to students leaving computing majors [39]. There is a clear need to improve the usefulness of programming error messages for novices. We argue that achieving this goal requires an evidence-based understanding of message readability.

2.2 The Problem with Programming Error Messages

Wrenn and Krishnamurthi argue that as error messages are a human-computer interaction element, they should be subject to user studies and similar forms of evaluation [62]. Why more effort has not gone into improving error messages in popular programming environments is an open question. Alexandrescu posed this question to compiler designers, finding that many are so familiar with their environment they don't appreciate the difficulty of those with less expertise, along with the simple fact that producing better error messages is simply not as high a priority as building new features [3].

An eye-tracking study conducted by Barik et al. [6] provided compelling empirical evidence of the need for improving error messages, including:

(1) Programmers do read error messages (corroborated by [46]);
(2) the difficulty of reading these messages is comparable to the difficulty of reading source code;
(3) difficulty reading error messages significantly predicts participants' task performance, and;
(4) participants allocate a substantial portion of their total task to reading error messages (13-25%).

The final item above resonates with Buse and Weimer who cited several sources noting that the act of reading code is the most time-consuming of all maintenance activities [16]. This paints a picture contrary to common wisdom which says that programmers spend most of their time writing code. Instead there is evidence that programmers spend a lot of time reading code and *reading error messages*.

While poorly designed error messages can affect all programmers in writing code, for novices they can interfere with learning which is more formative and complex. Educators and researchers have been documenting for more than 50 years the difficulties that programming error messages present to students [9]. The effects of poor error messages may also be particularly severe for students who lack confidence in the subject. In their seminal work, "Unlocking the Clubhouse", Margolis and Fisher examined the gender gap in computing education, and found that women were more likely than men to transfer out of undergraduate computer science degrees early in their studies [24]. The authors propose that this may be due to a lack of confidence, stating "women and other students who do not fit the prevailing norm are disproportionately affected by problems like poor teaching, hostile peers, or unapproachable faculty" [24, p140]. It is possible that student confidence may be negatively impacted by the use of unfriendly tools, which produce cryptic and unhelpful messages that are hard to read. Despite the problems with poor error messages persisting for over half a century, there remain ongoing calls for their improvement to help novices learn more effectively [48].

2.3 Programming Error Message Readability

Programming error messages are a subset of the much broader area of system error messages which, like programming error messages, leave much to be desired. Maglio & Kandogan found that System Administrators spend up to 25% of their time down blind alleys suggested by poorly constructed and unclear messages [38]. In 1982, Shneiderman developed recommendations for system error message design including several specific recommendations for error messages as shown in Table 3.

We take Shneiderman's guideline of 'comprehensible' to be a synonym for 'readability', which is the focus of the present study. While error message readability has received almost no attention from human factors and usability researchers, a few studies through the years touch on it. At the second CHI conference in 1983, Isa et al. presented their work on error messages from an HCI perspective, noting that they can contain unnecessary jargon, be cryptic, unfriendly, or misleading [30]. They concluded that "Surely, the use of guidelines is a valuable first step in the production of usable error messages" [p71].

The only guidelines for the *readability of programming error messages* that have been put forward in the literature are mostly vague and not easily measurable. There is such a lack of literature that most have to be inferred from more general usability guidelines not necessarily specific to programming, for instance Jakob Nielsen at CHI '94 wrote that "[Error Messages] should be expressed in plain language..." [44, p156]. In contrast, the readability of prose is a well-studied area. Metrics such as the Dale-Chall formula, Farr-Jenkins-Paterson formula, Flesch formula, Fry's Readability Graph, Kincaid formula, Gunning Fog Index, and Linsear Write Index have been used for decades [37]. These metrics all focus on natural language prose for novels and technical manuals, and some even produce a grade-level as output (such as Fry's Formula) which

Step	Code	Error message
1	int fahrenheit celsius;	error: expected '=', ',', ';', 'asm' or '__attribute__' before 'celsius'
2	int fahrenheit = celsius;	error: 'celsius' undeclared
3	int celsius; int fahrenheit = celsius;	error: 'celsius' is used uninitialized in this function

Table 2: A short interaction between May, a novice programming student, and her compiler.

Error messages should be...	Error message should not be...
Brief	Wordy
Positively toned	Negatively toned
Constructive	Critical of errors
Specific	General
Comprehensible	Cryptic
Emphasize user control-over system	Suggest system control-over user

Table 3: System error message design guidelines from [53].

might not be a helpful distinction to make in programming. These metrics were not designed for programming error messages, which are often terse and filled with symbols, and have recently been shown to produce entirely nonsensical results when applied to them [22].

Buse and Weimer developed a model for the readability of *source code*. They define readability as "a human judgment of how easy a text is to understand" [16, p121], and argue that the readability of a program is related to its maintainability, and therefore is a critical factor in overall software quality [16]. They also make several findings that could inform a readability metric for programming error messages which are, by nature, short. They found that:

(1) Length of identifier names had almost no influence on source code readability. This is important as programming error messages must necessarily repeat identifier names (and thus have no control over them). Additionally, it indicates that an error message metric might be stable from programmer to programmer, and language to language.
(2) A high number of identifiers had a very strong influence on readability of source code.
(3) A high number of characters on a line had a very strong influence on readability of source code.

Buse and Weimer conclude that code readability is an essential characteristic of code quality. We propose that the readability of programming error messages is an essential characteristic of their potential usefulness. Although prior work has argued that more readable error messages will result in increased student satisfaction [63], lower error rates and fewer repeated errors [7], and less frustration for students [50], they have stopped short of defining readability, assessing readability in isolation, or providing a way to measure readability.

2.4 Improving Programming Error Message Usability

Various efforts have been made, with increasing frequency in recent years, to enhance, side-step, or otherwise improve the usability of programming error messages. For instance:

(1) Many efforts have been made to improve programming error messages by intercepting them between the compiler and the user and translating them into more "usable" forms – often called Enhanced Compiler Error Messages (ECEMs) [12]. However, the evidence for the effectiveness of ECEMs is not overwhelming. Efforts focusing on enhancing/improving compiler error messages have been made by researchers such as Barik [4], Becker [13], Denny [19], Kohn [33], Pettit [45], Prather [46], and Karkare [2]. However, it is not apparent that any have addressed the core issue of readability.
(2) Changes in the presentation of error messages, particularly through non-textual IDE features, have also attempted to address the shortcomings of the messages themselves. One very recent attempt by Dong and Khandwala shortened incredibly verbose error messages behind ellipses, utilized color, and added newlines and meaningful headings [23] to programming error messages in Java. They found a large increase in usability (measured via comprehension and resolution rate) with just those small cosmetic changes.
(3) Other work has sought to reimagine the standard programming error message. Hartmann et al. updated error messages in their system *HelpMeOut*, which presented novices with examples of how other programmers fixed the same kinds of errors [28]. While certainly novel, it sidesteps the issue of readability entirely.
(4) Programming language designers are now starting to take "good" error messages seriously (for instance, Rust [59] and Elm [18]). However, most efforts on this front focus on structuring messages, consistency, and context-specific improvements, overlooking the more basic issue of *readability*. Often it seems to be a premise that more information is better. The Quorum language takes an evidence-based approach to language design [56], including ongoing work on error messages. Although the focus on message design in these modern languages is promising, the vast majority of novice programming students still learn using one of the three languages we explore in this work [11, 55].
(5) Other efforts seek to use external information, such as Stack Overflow, to provide more error-correcting ability to users [57, 61]. Given the advances in machine learning techniques, large repositories outside of the code can be used to diagnose problems with source code [2]. Yet again, these efforts seem to take the view that more information is better, and are operating at a level much higher than improving basic readability.

As is evident, researchers have gone to great lengths to alleviate the ineffectiveness of programming error messages they see in the classroom. One of the root causes of this ineffectiveness must lie in the poorly-understood readability of these messages – a critical aspect which has remained largely unstudied.

3 STUDY 1: EXPLORING MESSAGE READABILITY

We set out to answer our first research question without any preconceptions of what affects message readability for novices. We therefore designed Study 1 as an exploratory pilot study, to begin developing an understanding of novice perceptions of readability. The goal of this study was to produce a ranked list of messages using a holistic measure of readability. We then inspected differences between those rated as most and least readable to identify possible factors that may influence readability.

3.1 Methodology

We began by constructing a set of 60 error messages to present to novices to rate. We selected the 20 most frequently occurring error messages in three popular programming languages: Java (SE 7), C (gcc 5.4.0), and Python (3.6). For the Java dataset [7], consisting of 28,000 code submissions, the top 20 messages cover 89% of all errors present in the set. The top 20 messages in the Python dataset [61] covered 99% of the 35,000 submissions. The C dataset [43] contained 5,002 total errors with the top 20 errors covering 95% of the submissions. These are the most commonly taught introductory languages. The Java and C messages are drawn from existing corpora of syntax errors generated directly by novice (first-year) programmers. Although the Python corpus is from a more general source (Stack Overflow) the messages themselves are frequently encountered by novices. Specifically, "invalid syntax" is the most common novice message, and the EOF and indentation-based messages are among the five most common [47] accounting for four of the six Python messages shown in Study 3.

We constructed a questionnaire that presented each of the twenty most frequent error messages from our three languages (Java, Python, and C), and asked participants to rate the *readability* of the message on a scale from 1 (least readable) to 10 (most readable). No specific criteria were provided to define what "readable" meant, as in this exploratory study we did not wish to constrain participants by the researchers' definitions without any evidence.

3.2 Results

Participants were first-year students enrolled in a typical CS1 course being taught in C++ at a North American university (n=33) with 8 identifying as women and 25 as men. Data was collected in the fourth week of term, and thus participants had only a few weeks of experience encountering programming error messages. Participants responded to the questionnaire during a 50 minute in-class session. Two of the participants rated all 60 messages at 10 and were therefore excluded from analysis.

Table 4 shows the 10 most readable and the 10 least readable messages, ranked according to mean readability score. Despite the fact that our sample represented three different programming languages, the individual error messages with the highest average readability score generally appeared to be alike and shared certain qualities. Likewise, the messages with the lowest average readability score also shared certain qualities. Examination of the similarities of these messages within groups (the top and bottom 15 represent quartiles) suggests two qualities which may play a role in the perceived readability for novice programmers: message length and the density of jargon/acronyms.

Across all participants, shorter messages tended to be rated as more readable. The median number of characters per message among the 15 most readable messages (21) is less than half that of the 15 least readable messages (44). While a few more words of explanation can often go a long way to helping a novice understand an error, these messages were often verbose without aiding understanding. This supports earlier work in which the addition of words does not consistently improve novice comprehension [9].

There is also some evidence that messages with more jargon, technical terms, and acronyms are perceived to be less readable. Acronyms such as 'EOF' and 'EOL', and technical terms such as 'positional argument', 'keyword argument', and 'triple-quoted string literal' are prevalent among the least readable messages. In addition, the lowest ranked message contained quite a bit of jargon: "unicode error 'unicodeescape' codec can't decode bytes."

The main limitation of this first study was that we provided no guidance to participants on what "readable" meant, although as an exploratory study this was also a strength. The purely quantitative nature of the data was also a limitation. To better understand how readability is perceived, in our second study we sought to elicit responses from participants. For analysis of other data from this study, see our prior work [10].

4 STUDY 2: ELICITING READABILITY FACTORS

In our first study, we identified message length and use of jargon as possible factors affecting readability, however these were inferred from manual inspection of the ranked list of messages. In our second study, we sought explicit statements from students. A smaller number of messages were needed given the nature of the data collection, in which participants provided written feedback rather than responding with a numeric rating. For this purpose, we adopted the set of eight error messages described in the study by Denny et al. [22]. This is one of the few studies we are aware of that has presented empirical evidence for the greater *usability* of one set of messages over another, as measured by the time required for participants to resolve errors. If message usability is indeed impacted by readability, as we expect, including both sets of messages from this prior study may elicit a wide range of responses from our participants.

4.1 Methodology

Table 5 shows the eight messages used in Study 2, taken from [22]. These messages correspond to four basic syntax errors in a C-language program: (1) misspelled file name, (2) missing comma in sequenced variable declaration, (3) missing function opening brace, and (4) missing ampersand in input scan. Each error is represented by two error message variants: [A] the unedited compiler messages shown in the "Regular" column, and [B] a reworded message intended to provide a clearer explanation of the error in the "Enhanced" column.

In our experiment, novice programmers in CS1 and CS2 courses were given the erroneous program along with a set of four error

			Readability	
Rank	Language	Error message	Mean	SD
1	Java	‘else’ without ‘if’	9.3	1.7
2	Java	‘(’ expected	9.2	1.4
3	Java	‘]’ expected	9.2	1.5
4	Java	‘;’ expected	9.2	1.7
5	C	‘else’ without a previous ‘if’	9.1	2.0
6	Java	‘)’ expected	9.0	1.8
7	Java	while expected	8.6	2.2
8	C	expected <x> before <y>	8.5	1.7
9	Java	<identifier> expected	8.4	1.9
10	C	no such file or directory	8.2	2.5
51	Java	reached end of file while parsing	5.9	2.7
52	Python	positional argument follows keyword argument	5.8	2.6
53	Python	unexpected EOF while parsing	5.8	3.2
54	C	invalid type argument of <...>	5.8	2.5
55	C	missing terminating <x> character	5.7	2.6
56	Python	EOL while scanning string literal	5.6	3.1
57	Python	EOF while scanning triple-quoted string literal	5.4	3.0
58	C	invalid operands to binary <x> (have <y> and <z>)	5.1	2.7
59	C	expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘_attribute_’ before <x>	4.3	2.9
60	Python	(unicode error) ‘unicodeescape’ codec can’t decode bytes	4.3	2.9

Table 4: From the original set of 60 messages, the 10 rated most readable, and the 10 rated least readable.

	Regular	Enhanced
1	1:10: fatal error: studio.h: No such file or directory.	A file name appears to be misspelled. The file being included cannot be found and so is probably not spelled correctly.
2	9:21: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribute__’ before ‘feet’.	A comma appears to be missing. When declaring multiple variables on the same line, names should be separated by commas.
3	13:5: error: expected declaration specifiers before ‘scanf’.	An opening brace appears to be missing. Functions should have a matching opening and closing brace.
4	13:13: error: format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘int’.	An ampersand (&) appears to be missing. An address must be provided to the scanf() function, by using an & before the variable name

Table 5: Regular and enhanced compiler error messages (ECEMs) from [22].

messages chosen at random from four counter-balanced sets: {1A, 2A, 3B, 4B}, {1B, 2B, 3A, 4A}, {1A, 2B, 3A, 4B}, or {1B, 2A, 3B, 4A}. In this way, each participant viewed the four errors in a consistent order, but provided feedback on the readability of two regular and two enhanced messages.

To assess readability, participants were asked the yes/no question, “*Would the inclusion of certain information make the error message above easier to read?*” with the following prompts for further explanation: “*If yes, what information?*” and “*If no, is there any information that’s unnecessary, excessive, or could be removed?*”. Participants were also asked to directly respond to the question: “*For the error message shown, what do you think makes it easy or hard to read? (Consider: number of words, word length, specific characters, line numbers, sentence structure, etc.)*” with separate responses for “Easy” and “Hard”.

4.2 Results

Participants were recruited from two introductory programming courses taught at universities in the United States (n=41) and in China (n=73). Since programming error messages are written in English, this design allows us to explore differences between students whose first language is English and students whose first language is Chinese. Of the 73 Chinese respondents, 26 identified as women, and 47 as men. The survey was available to respondents for one week, and students had already taken a semester of Python, but were learning C at the time of this study. Of the 41 US respondents, 6 identified as women, and 35 as men. Survey responses were collected during a 50 minute in-class session. Students in this cohort were familiar with C++ at the time they participated in the study.

Two authors collated the open-ended responses into thematic groups based on use of similar words and synonyms. The groupings for participants whose first language was English are shown in Table 6. Likewise, groupings for participants whose first language was Chinese are shown in Table 7. Participants were able to mention multiple factors that help or hinder readability, so counts in the tables do not add up to participant totals.

English-speaking and Chinese-speaking participants tended to mention the same kinds of ideas, though the groups prioritized them differently. English-speaking participants thought that more English (i.e. more explanation instead of the often terse existing error messages) would be the most helpful, while Chinese-speaking participants prized fewer and more precise words over lengthier explanations. Likewise, only four participants whose first language was English mentioned heavy jargon as a problem, while participants whose first language was Chinese mentioned it 43 times. However, it seems both groups valued brevity where possible, the inclusion of source code line numbers, and clear sentences. Both groups also thought the use of jargon and unfamiliar vocabulary could make it more difficult to understand the messages.

One participant wrote that what hindered their understanding of the original message was that it was "perhaps too concise to figure out where/what caused the error from the code." Another participant wrote almost the opposite about one of the enhanced messages "It is clear and concise and tells the problem immediately." Short and concise can be helpful at times, but brevity alone is not the goal. One participant wrote, "It is short and thus easy to read but there is little information, thus not as helpful." Another participant noted that "It is easy to read. It is more in English than in code" perhaps noting the penchant of error messages to make heavy use of jargon, symbols, and code. Clearly, some kind of balance must be struck between concision and helpfulness.

Most of the common themes between groups align well with existing, but untested, guidelines from programming error message design [9], such as being as succinct as possible without sacrificing meaning. In this experiment, because participants were shown the program code as well as the corresponding error messages, some of the themes were directly tied to context and could not be independently attributed to the error messages themselves. The inclusion of line numbers mapping to the corresponding source code is one such example. Also, because some of the messages were enhanced (the variants in set [B]) to deliberately suggest potential solutions, themes that related to this are of less practical value for evaluating existing programming messages in popular languages.

Three authors met and discussed these concerns alongside the existing guidelines proposed in the literature for the design of programming error messages. We settled on four themes emerging from Studies 1 and 2 that appeared promising for evaluating existing messages:

(1) succinct / verbose
(2) less jargon / more jargon
(3) clear sentence structure / unclear sentence structure
(4) simple vocabulary / complex-advanced vocabulary

The first three were inspired by the English-speaking participants with support from the Chinese-speaking participants. However, we learned from the Chinese-speaking participants that even non-jargon words could affect readability if English isn't one's first language. We therefore added a fourth theme to help target accessibility for non-native English speakers. Additionally, the first two themes overlapped with those from Study 1, contributing evidence to their validity.

5 STUDY 3: CONFIRMING READABILITY FACTORS

Finally, having identified four possible factors through our first two studies, we turn to our second research question and examine the extent to which each one impacts message readability. To this end, we designed a third experiment to collect student perceptions of message readability with respect to message length, use of jargon, sentence structure and vocabulary use. We used a subset of the same 60 messages from Study 1, so that we could relate the results back to the holistic ratings of readability collected in the earlier experiment. Data for Study 3 was collected from students at an institution distinct from those used in Studies 1 and 2.

5.1 Methodology

We prepared a questionnaire consisting of a set of error messages, each of which could be rated against the four scales: length, jargon, sentence structure and vocabulary. We selected a set of 18 error messages from the original set of 60 messages that were used in Study 1. Including all 60 messages on the questionnaire was not practical, as early piloting revealed that it would take too much time for students to complete. The trade-off between questionnaire length and response rate is complex, however previous studies with university students have shown that shorter questionnaires can lead to higher response rates in a non-linear fashion [1]. Selecting the messages from the original Study 1 pool ensured that we were using messages that commonly occur in practice.

Messages were shown to participants one at a time, and for each message, an item for each factor could be rated on a 5-point Likert scale. We also included, for each message, a holistic rating of the general *understandability* of the message. We use this to compare with prior ratings of *readability* for the same messages from Study 1, and to correlate with each of the four factors being tested. Table 8 gives the full list of items, and associated descriptors, that were presented to participants for each error message.

5.1.1 Error message selection. Selection of the 18 messages appearing in the questionnaire was accomplished with a view of achieving an even balance across language, length, and prior ratings of readability. The original set of 60 messages were ranked, in order from most to least readable, based on the holistic scores from Study 1. With three programming languages represented, all messages for a language could be grouped into quartiles of size five using these rankings. The length of the messages in the original set ranged from two to nine words, respectively. The set of 18 messages were selected such that all lengths between two and nine words were represented either two or three times, and for each language, six messages were chosen with one or two messages taken from each readability quartile. Table 9 shows these 18 messages.

5.1.2 Participant recruitment. Students were recruited from a large urban university in Australasia. Invitations were sent by email to

Beneficial to Readability	Count	Detrimental to Readability	Count
More English makes it easier to read	29	No line number	39
Explanation of error	27	Verbose	22
Brevity	24	Line number output confusing or difficult to read	20
Line number helps in finding error	20	Poor wording	20
Error separated into 2 lines	10	Excessive punctuation (colons, commas, quotations, ...)	16
Clear wording	7	Vague description of error	12
Gives many possible solutions	7	No template / example of proper declaration	8
		Multiple solutions cause confusion	5
		Heavy jargon	4
		Too short or not descriptive enough	3
		Error separated into 2 lines	1

Table 6: Readability themes of students whose first language was English and the number of responses grouped by that theme.

Beneficial to Readability	Count	Detrimental to Readability	Count
Specific characters	76	Confusion with or lack of line number	51
Brevity/right amount of words	63	Specific characters/confusing jargon	43
Line number helps with finding error	48	Sentence is too long/too many words	36
Sentence structure (little to no specification)	46	Sentence structure (little to no specification)	31
Word length	35	Word length	19
		Message is too short	4

Table 7: Readability themes of students whose first language was Chinese and the number of responses grouped by that theme.

Item	Range (1) – (5)	Descriptor
Length	Succinct – Verbose	Is the message expressed using more words than needed?
Jargon	Less – More	Does the message contain jargon and technical terms?
Sentence structure	Clear – Unclear	How clear is the sentence structure of the message?
Vocabulary	Simple – Complex/Advanced	How complex is the vocabulary used?
Understandability	Easy – Hard	How easy/hard do you think the message is to understand?

Table 8: The items against which each error message was rated.

all students across the faculties of Engineering and Science who were enrolled in programming intensive courses in their first year of study. Approximately 2,000 students received an invitation, containing a link to the online questionnaire, and an indication that it would take approximately 15 minutes to complete and that all responses were anonymous. Participation was optional, in that no course credit was associated for completing the questionnaire.

5.2 Results

The questionnaire was available for approximately seven days, and in that time received a total of 95 responses (representing approximately a 5% response rate). One student was removed because they did not answer 20 of the 90 core questions (five Likert questions × 18 messages). Two students did not answer two of the 90 questions and four did not answer one. We assumed that these were erroneously skipped and we replaced the blank with the median response for that question.

Of the 94 respondents, five students did not identify as a woman or man, 24 identified as women, and 65 as men. 67% reported that English was their first language. 54% of students listed Python as a language they are most familiar with, followed by Java (20%), MATLAB (19%) and C (6%). We found no statistically significant differences between the following groups in terms of understandability:

(1) Students identifying as women and men (the small numbers not identifying as either a man or woman prevented us from making meaningful statistical comparisons outside these groups)
(2) Native and non-native English speakers
(3) Number of natural languages spoken
(4) Number of programming languages students are familiar with.

Table 9 lists the 18 error messages used in Study 3, along with the corresponding language and understandability ranking (1 is most understandable).

5.2.1 Length, Jargon, Sentence Structure, Vocabulary & Understanding. Table 10 reports coefficients for the Pearson's product-moment

Language	Error message	Understandability rank
C	no such file or directory	1
C	‘else’ without a previous ‘if’	2
Java	‘else’ without ‘if’	3
Java	‘;’ expected	4
Python	unexpected unindent	5
C	storage size isn’t known	6
Python	invalid syntax	7
Java	not a statement	8
Java	reached end of file while parsing	9
Python	unindent does not match any outer indentation level	10
C	conflicting types for ‘add’	11
C	expected declaration or statement at end of input	12
Java	bad operand types for binary operator ‘*operator*’	13
Python	generator expression must be parenthesized if not sole argument	14
Java	illegal start of type	15
C	expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘_attribute_’ before <x>	16
Python	(unicode error) ‘unicodeescape’ codec can’t decode bytes	17
Python	EOF while scanning triple-quoted string literal	18

Table 9: The 18 error messages in Study 3, corresponding languages, and understandability rankings (1 = most understandable).

correlation between each of the four factors (length, jargon, sentence structure and vocabulary) with the holistic rating of the understandability of the error message. All factors are strongly correlated with understandability, and all correlations are positive which matches our intuitive sense for each factor to varying extents. This is evidence that students find messages easier to understand when those messages are more succinct, include less jargon, use a clear sentence structure, and have simpler vocabulary. Message length correlates less strongly with understandability compared with the other factors. This makes some intuitive sense as students may find messages that are too short harder to understand. Theoretically the optimal length can’t be zero. This is not the case for other factors, such as jargon, where a complete absence of jargon in an error message may indeed make it easier to understand. In other words, message length will always be a matter of balance.

Figures 2A-2D show length, jargon, sentence structure, and vocabulary vs. understandability. Each plot has 1,692 data points representing all 18 error messages for each of the 94 students. For instance, Figure 2A shows all length scores for all messages, for all students. The white ‘bands’ are a visualization artifact due to ‘jitter’ (x and y) being used to make density perceptible – otherwise all points would be 1, 2, 3, 4 or 5, exactly on top of other points at the same coordinates. On the x-axis, each point is grouped into one of three categories: easy, moderate and hard (to understand). These groupings were determined by summing all understandability scores given to each of the 18 error messages. The error messages were then ranked (see Table 9) and divided into three groups corresponding to easy (messages ranked 1–6), moderate (ranked 7–12) and hard (ranked 13–18) to understand. Therefore, in Figure 2A, a given point represents a student’s length score for a given message, and that messages’ position in relation to the others in terms of understandability. The horizontal bar represents the median and the boxes represent the interquartile range of each distribution.

Although our raw data was normal, when grouped by easy, moderate and hard, the data was not. Therefore we utilised non-parametric tests for significance. We performed a Kruskal-Wallis rank-sum test on each triad in each of Figures 2A-2D. In all cases the differences between each understandability grouping is statistically significant. Details such as test statistics are presented in Table 11.

The differences in rating distributions between pairs of understandability groupings are also statistically significant, and hold for each of the four factors (Figures 2A-2D), as determined by pairwise Wilcoxon signed rank tests. Details of these comparisons are presented in Table 12. These agree with the correlations discussed in Table 10. The biggest differences occur with Jargon and Vocabulary, which are also the factors with the largest correlations. Combined with the correlations this is evidence that shorter messages, less jargon, clearer sentence structure and simpler vocabulary all result in more understandable error messages. A Bonferroni correction for 12 Wilcoxon signed-rank tests would yield a significance level of .004 (with $\alpha = 0.05$) and our results are still significant with this correction. A similar correction for the Kruskal-Wallis tests would not affect the significance of those tests.

5.2.2 *Readability and Understandability.*

In Study 1, participants were asked to make a holistic assessment of *readability*, without specific guidance on a scale from 1 to 10, as we did not wish to influence participants’ views, given that no one definition of readability is agreed on, particularly when it comes to programming error messages. In Study 3, which involved a subset of the messages from Study 1, participants were asked to rate the messages for *understandability*. We did this because we were concerned that *readability* without any guidance may be interpreted in a number of different ways, and that *understandability* likely has a more universally understood meaning. Figure 3 shows a scatterplot of the normalized readability rank in Study 1 against the normalized understandability rank in Study 3, for all 18 messages. In this plot, 0 corresponds to most readable/understandable and 1 corresponds to least readable/understandable. Pearson’s correlation coefficient

Factor	Correlation	p	Relationship with understandability
Length	$r(1690)$=.56	<.001	Succinct ≙ more , Verbose ≙ less
Jargon	$r(1690)$=.65	<.001	Less jargon ≙ more , More jargon ≙ less
Sentence structure	$r(1690)$=.64	<.001	Clear ≙ more, Unclear ≙ less
Vocabulary	$r(1690)$=.68	<.001	Simple ≙ more, Advanced/Complex ≙ less

Table 10: Correlations of the four factors with *understandability*; r is Pearson's correlation coefficient (≙ → "corresponds to").

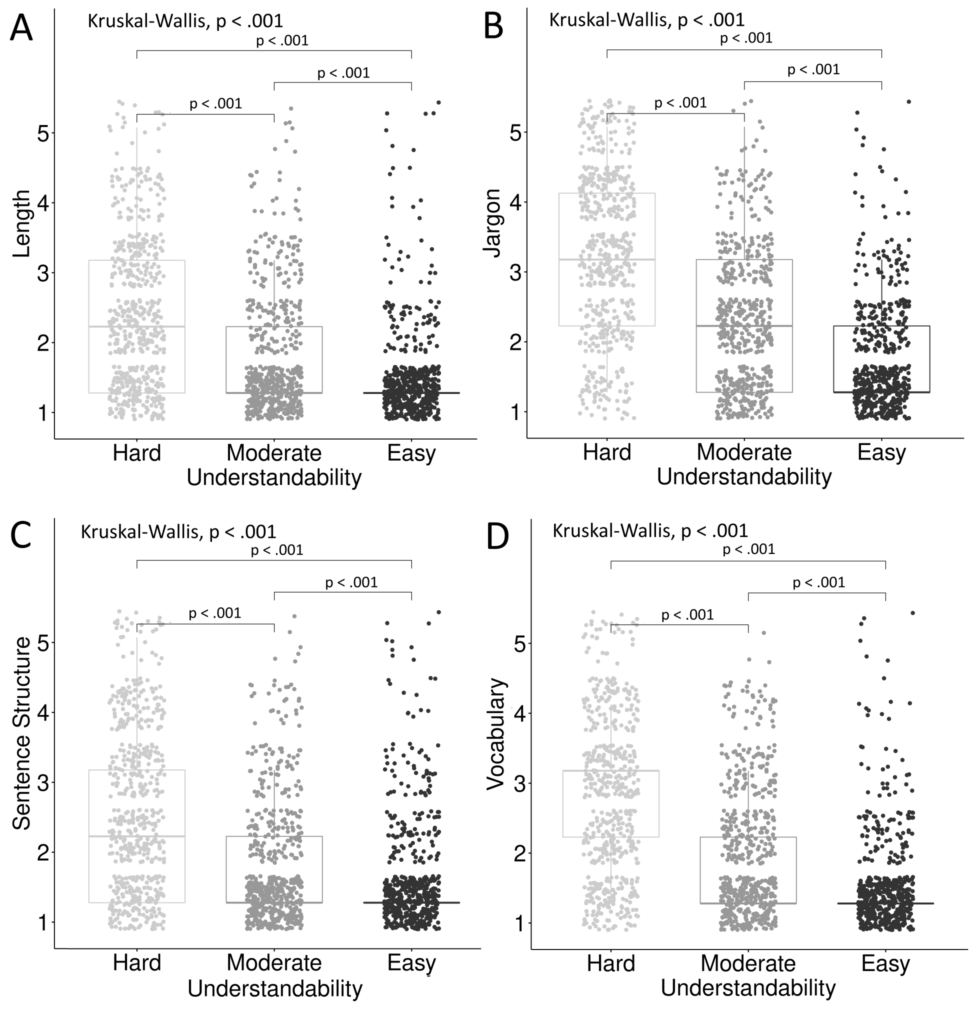

Figure 2: Message length, jargon, sentence structure and vocabulary scores vs. understandability (grouped into easy, moderate and hard to understand). Wilcoxon signed-rank tests show statistical significance between pairs of distributions, and Kruskal-Wallis rank-sum tests show statistical significance within each triad.

shows that the correlation between readability and understandability is very strong, $r(16)$=.86, p<.001. This is important, as it indicates that each of the correlations of understandability with length, jargon, sentence structure, and vocabulary are also applicable to a general but undefined *notion* of readability.

Although these results were expected and may even seem to be purely common sense, it is important to note that to date there has been no empirical data on what makes a programming error message readable. We have provided the first evidence of the factors that make up readability in this context. Moreover, as discussed above, this is necessary work because researchers have been calling for "readable" error messages for decades and yet no one has defined it and error messages continue to frustrate novices on multiple levels.

Fig. 2	Kruskal-Wallis
A	$\chi^2(2)$ = 299.68, p < .001
B	$\chi^2(2)$ = 479.18, p < .001
C	$\chi^2(2)$ = 275.20, p < .001
D	$\chi^2(2)$ = 390.68, p < .001

Table 11: Kruskal-Wallis rank-sum test results for Figure 2 (α = 0.05) - see Section 5.2.1 for discussion on correction for multiple tests.

Fig. 2	Pair	N	M	SD	N	M	SD
A	Hard-Moderate	564	2.21	1.13	564	1.57	0.90
	Moderate-Easy	564	1.57	0.90	564	1.28	0.70
	Hard-Easy	564	2.21	1.13	564	1.28	0.70
B	Hard-Moderate	564	3.10	1.23	564	2.08	1.03
	Moderate-Easy	564	2.08	1.03	564	1.51	0.82
	Hard-Easy	564	3.10	1.23	564	1.51	0.82
C	Hard-Moderate	564	2.41	1.23	564	1.61	0.94
	Moderate-Easy	564	1.61	0.94	564	1.41	0.84
	Hard-Easy	564	2.41	1.23	564	1.41	0.84
D	Hard-Moderate	564	2.57	1.20	564	1.79	0.98
	Moderate-Easy	564	1.79	0.98	564	1.31	0.72
	Hard-Easy	564	2.57	1.20	564	1.31	0.72

Table 12: Wilcoxon signed-rank test details for Figure 2 (α = 0.05, 0.004 with Bonferroni correction). In all cases, $p < .001$.

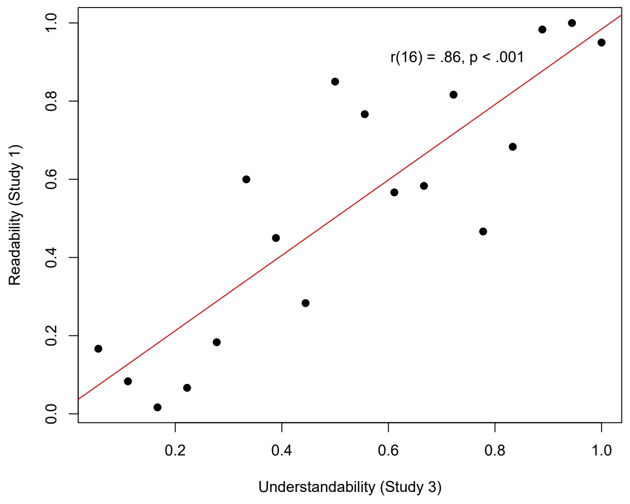

Figure 3: Scatterplot of normalized readability vs normalized understandability for all 18 error messages. On the axes, 0 is most readable/understandable and 1 is least. r is Pearson's correlation coefficient.

6 DISCUSSION: DESIGN IMPLICATIONS

Finally, in light of our findings, we now present specific insights about the design of programming error messages that will make them more readable and, therefore, more usable. We present these in the form of four concrete design guidelines. These guidelines can be used directly to inform the creation or enhancement of programming error messages, and may also form the basis for future efforts to construct a readability metric appropriate for such messages.

6.1 Remove Jargon

It is clear that students found messages with less jargon to be more readable than those with more. The highest ranked message in Study 3 was the following C error message: "no such file or directory". From the perspective of a typical novice programmer, this contains no jargon. In contrast, the lowest ranked message, a Python error message, consisted almost entirely of jargon: "EOF while scanning triple-quoted string literal". Applying the proposed design criterion to this lowest ranked message could yield something like "A triple-quoted string is missing the triple closing quotes." This removes jargon from the message without sacrificing readability.

6.2 Write Messages in Complete Sentences

As mentioned in our introduction to this paper, Brown wrote in 1982 that programming error messages are in some semblance of English and so we deceive ourselves about their readability [14]. However, our results show that this is not always the case. Many messages use very little English and are not written in complete sentences. For instance, the lowest-ranked message in C was "expected '=', ',', ';', 'asm' or '_attribute_' before <x>" which can be broken down to "expected <symbol> before <statement>" which is still not a complete sentence. Applying the design criterion to this statement could perhaps result in the following: "A symbol, such as a comma (',') or semicolon (';'), is missing."

6.3 Use Simple Vocabulary

While programming error messages are often full of jargon, even the remaining words are often written using advanced terminology and vocabulary. Our results show that this harms the message's readability for English-speaking novices, but we found that this is especially exacerbated among non-native English speakers who may not have such a deep well of vocabulary upon which to draw. One of the lower-ranked Python error messages, "generator expression must be parenthesized if not sole argument" is a prime example. Words like "argument" and "expression" are common programming jargon, but the words "generator," "parenthesized," and even "sole" represent a more advanced vocabulary that may prove a barrier for non-native English speakers. Looking up those words in a dictionary may prove entirely unhelpful. Moreover, in this Python example, if the generator expression was the sole function argument, the error would not be triggered in the first place. Therefore, shortening the message (see Section 6.4) and removing the complex vocabulary might produce something like: "Put '(' and ')' around expression that generates list". This supports the findings of Guo et al. [27] who found that non-English speakers wanted simplified English without English-specific slang.

6.4 Use An Economy of Words

Many programming error messages are terse, probably in an effort to be precise and concise. Other messages are often verbose, probably to communicate as much information and detail as possible. But either of these extremes typically means that error messages are not readable because they do not have what is often called an "economy of words," which means that only as many words are used as are necessary to communicate the point to the reader. One might be tempted to think that brevity, in and of itself, is a reasonable goal, but our results indicate this is not the case. One of the lowest-ranked Java messages: "illegal start of type" is very short and yet rated nearly incomprehensible by our participants. The message positioned immediately above it in the rankings was a Python message: "generator expression must be parenthesized if not sole argument" and the longest of all messages in the set. Of course, two of the most readable messages in the set were very terse: 'else' without 'if' *and* ';' expected. Our novel results above indicate that while novices generally prefer shorter messages, this category had the weakest correlation to understandability and therefore readability. Therefore, the final design criterion is that messages should use an economy of words, using as few as necessary without sacrificing clarity or eschewing important details.

We propose one final example which applies this design guideline to another of the poorly-rated Python messages "(unicode error) 'unicodeescape' codec can't decode bytes". Although this message also contains jargon, with respect to using an economy of words, we would argue that in this case reducing the number of words would further harm readability. In other words, this message does not use enough words and could use *more*. The message describes an error in which a string contains a unicode escape sequence ("\U") followed by illegal characters (this is commonly due to specifying a file path incorrectly, such as "C:\Users\", which can be resolved by duplicating each '\' character to create appropriate escape sequences). Applying the current design guideline, keeping the message succinct but with sufficient detail, could result in the following message: "A string contains an invalid character after '\U'. Check the unicode sequence or duplicate each '\'."

These four design guidelines do not necessarily stand alone, but together can be applied to make readable messages. Many of the examples of poor messages above could be improved through application of several of these guidelines – if not all four – and further research is needed to test this. Taken together, these guidelines could be used to inform the creation of new programming error messages and in the revision of existing ones.

7 LIMITATIONS

There are several limitations to the work we have presented here, though we have tried to mitigate each one. The first is that neither a formal definition nor informal guidance was provided to participants in Study 1 regarding how to rate *readability*. As a result, individual participants may have interpreted the term quite differently, potentially invalidating the ratings. However, evidence from Study 3 shows that the notion of *readability* expressed by novice programmers is valid, and strongly correlated to both *understandability* and the four constituent factors of readability.

A further limitation in this work is the subjective decision regarding the programming languages from which to select error messages. We intentionally chose three contemporary languages that are frequently used in introductory programming courses [11, 55], since our work is targeted at novices. For Study 1, we selected the error messages from these languages that are most often encountered in practice by novices. Our rationale is that guidelines rooted in empirical analysis of the most common error messages are likely to have the largest impact. The messages used in Study 2 were taken from prior work by Denny et al. [22], in which a set of enhanced messages resulted in faster error resolution times for participants compared to a set of existing compiler messages. Our rationale for using both sets of messages was that we might elicit a wider range of perceptions regarding readability by showing participants messages known to be both effective and ineffective in practice. For Study 3, the messages were a representative subset of those from Study 1 with good, medium, and poor *readability* rankings. We only used a subset of the messages, instead of all 60, because early piloting of the full questionnaire revealed it was far too long (60 error messages x 5 scales) and led to high abandonment rates.

We observed a low response rate of around 5% for Study 3. In accordance with the institution's ethical protocols, no external incentive was provided to students for engaging with the anonymous questionnaire, and this likely had an impact on the response rate. Another factor may have been the length of the questionnaire itself, despite our deliberate efforts to trade-off some length for a higher rate of response [1]. Although it is likely therefore that our sample exhibits some selection bias, the proportional representation of participants with respect to gender and language experience was consistent with the courses across which the invitations were distributed. To mitigate the low rate of response, we were able to distribute the invitation widely, and thus obtained nearly 100 complete responses – enough to give our statistical analyses sufficient power.

Another limitation of this work is that, although we have presented four distinct factors of readability for programming error messages, we cannot guarantee that these are *all* the factors. Other researchers may determine there are additional factors and we leave this to future work.

Finally, our findings are based on self-reported and subjective data. We have attempted to mitigate this concern by running three studies, collecting both quantitative and qualitative data, and confirming the validity of Study 1 via Studies 2 and 3. It is also important to note that the first step towards producing a metric for the readability of programming error messages involves wading into the subjective idea of what *readability* means to novices, which necessitates self-reported data. Future work producing a formula-based metric can build upon the research we have presented here.

8 CONCLUSION

In this paper, we presented the results of three related studies that targeted the concept of *readability* and its constituent factors for programming error messages. We originally set out to answer two research questions – the first to identify potential factors that might affect readability, and the second to measure the extent to which they do. In answer to RQ1, we found four factors that are independent of presentation or development environment: length, jargon, sentence structure, and vocabulary. Based on these factors, we presented concrete design guidelines for writing more readable error messages in Section 6. In answer to RQ2, we found that each factor is strongly correlated to message readability and understandability. The factor with the weakest correlation was message length, because shorter messages are not necessarily easier to read – what matters is that words are used economically to communicate the error to the reader. Future work includes the creation of a readability metric for programming error messages that can use our data to confirm its accuracy. Finally, we hope that the empirically-derived guidelines presented here will aid in the creation of new programming error messages, as well as guide efforts to revise or enhance existing messages. Our goal is to see error messages become more readable, and therefore more usable, in the coming years – easing a decades-long struggle between countless novices and their programming environments.

ACKNOWLEDGMENTS

This project was approved by the University of Auckland Human Participants Ethics Committee, reference UAHPEC2883.

REFERENCES

[1] L. La Mar Adams and Darwin Gale. 1982. Solving the Quandary Between Questionnaire Length and Response Rate in Educational Research. *Research in Higher Education* 17, 3 (1982), 231–240. https://doi.org/10.1007/BF00976700

[2] Umair Z. Ahmed, Pawan Kumar, Amey Karkare, Purushottam Kar, Sumit Gulwani, and A.; et al Ahmed, U.; Kumar, P.; Karkare. 2018. Compilation Error Repair: For the Student Programs, From the Student Programs. In *ICSE-SEET 2018 : 2018 ACM/IEEE 40th International Conference on Software Engineering : Software Engineering Education and Training : proceedings : 30 May - 1 June 2018, Gothenburg, Sweden.* ACM Press, New York, New York, USA, 78–87. https://doi.org/10.1145/3183377.3183383

[3] Andrei Alexandrescu. 1999. Better Template Error Messages. *C/C++ Users J.* 17, 3 (March 1999), 37–47.

[4] Titus Barik. 2018. *Error Messages as Rational Reconstructions.* Ph.D. Dissertation. North Carolina State University, Raleigh. https://repository.lib.ncsu.edu/handle/1840.20/35439

[5] Titus Barik, Denae Ford, Emerson Murphy-Hill, and Chris Parnin. 2018. How Should Compilers Explain Problems to Developers?. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018).* Association for Computing Machinery, New York, NY, USA, 633–643. https://doi.org/10.1145/3236024.3236040

[6] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do Developers Read Compiler Error Messages?. In *Proceedings of the 39th International Conference on Software Engineering* (Buenos Aires, Argentina) *(ICSE '17).* IEEE Press, Piscataway, NJ, USA, 575–585. https://doi.org/10.1109/ICSE.2017.59

[7] Brett A. Becker. 2015. *An Exploration Of The Effects Of Enhanced Compiler Error Messages For Computer Programming Novices.* Masters Thesis. Dublin Institute of Technology. https://doi.org/10.13140/RG.2.2.26637.13288

[8] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16).* ACM, New York, NY, USA, 126–131. https://doi.org/10.1145/2839509.2844584

[9] 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. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE-WGR '19).* Association for Computing Machinery, New York, NY, USA, 177–210. https://doi.org/10.1145/3344429.3372508

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

[11] Brett A. Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19).* Association for Computing Machinery, New York, NY, USA, 1011–1017. https://doi.org/10.1145/3287324.3287485

[12] 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

[13] Brett A. Becker, Kyle Goslin, and Graham Glanville. 2018. The Effects of Enhanced Compiler Error Messages on a Syntax Error Debugging Test. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18).* ACM, New York, NY, USA, 640–645. https://doi.org/10.1145/3159450.3159461

[14] P. J. Brown. 1982. My System Gives Excellent Error Messages - Or Does It? *Software: Practice and Experience* 12, 1 (Jan 1982), 91–94. https://doi.org/10.1002/spe.4380120110

[15] P. J. Brown. 1983. Error Messages: The Neglected Area of the Man/Machine Interface. *Commun. ACM* 26, 4 (Apr 1983), 246–249. https://doi.org/10.1145/2163.358083

[16] Raymond P.L. Buse and Westley R. Weimer. 2008. A Metric for Software Readability. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA '08).* ACM, New York, NY, USA, 121–130. https://doi.org/10.1145/1390630.1390647

[17] Natalie J. Coull. 2008. *SNOOPIE: Development of a Learning Support Tool for Novice Programmers within a Conceptual Framework.* Ph.D. Dissertation. University of St Andrews, St Andrews, Scotland. http://hdl.handle.net/10023/522

[18] Evan Czaplicki. 2015. Compiler Errors for Humans. https://elm-lang.org/news/compiler-errors-for-humans

[19] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing Syntax Error Messages Appears Ineffectual. In *Proceedings of the 19th Conference on Innovation and Technology in Computer Science Education* (Uppsala, Sweden) *(ITiCSE '14).* ACM, New York, NY, USA, 273–278. https://doi.org/10.1145/2591708.2591748

[20] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All Syntax Errors Are Not Equal. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12).* ACM, New York, NY, USA, 75–80. https://doi.org/10.1145/2325296.2325318

[21] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. Understanding the Syntax Barrier for Novices. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (Darmstadt, Germany) *(ITiCSE '11).* ACM, New York, NY, USA, 208–212. https://doi.org/10.1145/1999747.1999807

[22] 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

[23] Tao Dong and Kandarp Khandwala. 2019. The Impact of "Cosmetic" Changes on the Usability of Error Messages. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI EA '19).* ACM, New York, NY, USA, Article LBW0273, 6 pages. https://doi.org/10.1145/3290607.3312978

[24] Allan Fisher and Jane Margolis. 2002. Unlocking the Clubhouse: The Carnegie Mellon Experience. *SIGCSE Bull.* 34, 2 (June 2002), 79–83. https://doi.org/10.1145/543812.543836

[25] Thomas Flowers, Curtis A. Carver, and James Jackson. 2004. Empowering Students and Building Confidence in Novice Programmers Through Gauntlet. In *34th ASEE/IEEE Annual Frontiers in Education (FIE '04, Vol. 1).* IEEE, Savannah, GA, USA, T3H/10–T3H/13. https://doi.org/10.1109/fie.2004.1408551

[26] David Gries. 1974. What Should We Teach in an Introductory Programming Course?. In *Proceedings of the Fourth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '74).* ACM, New York, NY, USA, 81–89. https://doi.org/10.1145/800183.810447

[27] 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* (Montreal QC, Canada) *(CHI '18).* Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3173574.3173970

[28] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R. Klemmer. 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10).* ACM, New York, NY, USA, 1019–1028. https://doi.org/10.1145/1753326.1753478

[29] James J Horning. 1976. What the Compiler Should Tell the User. In *Compiler Construction: An Advanced Course*, G Goos and J Hartmanis (Eds.). Springer-Verlag, Berlin-Heidelberg, 525–548.

[30] Barbara S. Isa, James M. Boyle, Alan S. Neal, and Roger M. Simons. 1983. A Methodology for Objectively Evaluating Error Messages. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, Massachusetts, USA) *(CHI '83)*. ACM, New York, NY, USA, 68–71. https://doi.org/10.1145/800045.801583

[31] Matthew C. Jadud. 2006. *An Exploration of Novice Compilation Behaviour in BlueJ.* Ph.D. Dissertation. University of Kent at Canterbury. https://jadud.com/dl/pdf/jadud-dissertation.pdf

[32] 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 & 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

[33] Tobias Kohn. 2019. The Error Behind The Message: Finding the Cause of Error Messages in Python. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. ACM, New York, NY, USA, 524–530. https://doi.org/10.1145/3287324.3287381

[34] Sarah K. Kummerfeld and Judy Kay. 2003. The Neglected Battle Fields of Syntax Errors. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20* (Adelaide, Australia) *(ACE '03)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 105–111. http://dl.acm.org/citation.cfm?id=858403.858416

[35] Thomas Kurtz. 1978. BASIC. *ACM SIGPLAN Notices - Special issue: History of programming languages conference* 13, 8 (1978), 103–118. https://doi.org/10.1145/960118.808376

[36] Michael J. Lee and Amy J. Ko. 2011. Personifying Programming Tool Feedback Improves Novice Programmers' Learning. In *Proceedings of the Seventh International Workshop on Computing Education Research* (Providence, Rhode Island, USA) *(ICER '11)*. ACM, New York, NY, USA, 109–116. https://doi.org/10.1145/2016911.2016934

[37] William Lidwell, Kritina Holden, and Jill Butler. 2010. *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design.* Rockport Publishers, Beverly, Massachusetts.

[38] Paul P. Maglio and Eser Kandogan. 2004. Error Messages: What's the Problem? *Queue* 2, 8 (Nov. 2004), 50–55. https://doi.org/10.1145/1036474.1036499

[39] Qusay H. Mahmoud, Wlodek Dobosiewicz, and David Swayne. 2004. Redesigning Introductory Computer Programming with HTML, JavaScript, and Java. *SIGCSE Bull.* 36, 1 (March 2004), 120–124. https://doi.org/10.1145/1028174.971344

[40] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. 2011. Measuring the Effectiveness of Error Messages Designed for Novice Programmers. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. ACM, New York, NY, USA, 499–504. https://doi.org/10.1145/1953163.1953308

[41] David Mccall. 2016. *Novice Programmer Errors-Analysis and Diagnostics.* Ph.D. Dissertation. The University of Kent. https://kar.kent.ac.uk/id/eprint/61340

[42] Linda Kathryn. McIver and Damian. Conway. 1996. Seven Deadly Sins of Introductory Programming Language Design. In *1996 International Conference on Software Engineering: Education and Practice (SEEP'96).* IEEE Computer Society, Dunedin, New Zealand, 309–316. https://doi.org/10.1109/SEEP.1996.534015

[43] Cormac Murray. 2019. *An Analysis of Programming Process Data in a CS1 Programming Module: Factors Influencing Success.* Masters Thesis. University College Dublin.

[44] Jakob Nielsen. 1994. Enhancing the Explanatory Power of Usability Heuristics. In *Conference companion on Human factors in computing systems - CHI '94.* ACM Press, New York, New York, USA, 152–158. https://doi.org/10.1145/259963.260333

[45] Raymond S. Pettit, John Homer, and Roger Gee. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive.. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 465–470. https://doi.org/10.1145/3017680.3017768

[46] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 74–82. https://doi.org/10.1145/3105726.3106169

[47] David Pritchard. 2015. Frequency Distribution of Error Messages. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools* (Pittsburgh, PA, USA) *(PLATEAU 2015)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/2846680.2846681

[48] Timothy Rafalski, P. Merlin Uesbeck, Cristina Panks-Meloney, Patrick Daleiden, William Allee, Amelia Mcnamara, and Andreas Stefik. 2019. A Randomized Controlled Trial on the Wild Wild West of Scientific Computing with Student Learners. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 239–247. https://doi.org/10.1145/3291279.3339421

[49] H. G. Rice. 1953. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Amer. Math. Soc.* 74, 2 (1953), 358–366. http://www.jstor.org/stable/1990888

[50] Peter C. Rigby and Suzanne Thompson. 2005. Study of Novice Programmers using Eclipse and Gild. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange.* ACM, San Diego, California, 105–109. https://doi.org/10.1145/1117696.1117718

[51] Saul Rosen, Robert A. Spurgeon, and Joel K. Donnelly. 1965. PUFFT - The Purdue University Fast FORTRAN Translator. *Commun. ACM* 8, 11 (nov 1965), 661–666. https://doi.org/10.1145/365660.365671

[52] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. 2014. Programmers' Build Errors: A Case Study (at Google). In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. ACM, New York, NY, USA, 724–734. https://doi.org/10.1145/2568225.2568255

[53] Ben Shneiderman. 1982. Designing Computer System Messages. *Commun. ACM* 25, 9 (1982), 610–611. https://doi.org/10.1145/358628.358639

[54] Ben Shneiderman. 1997. *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (3rd ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[55] Simon, Raina Mason, Tom Crick, James H. Davenport, and Ellen Murphy. 2018. Language Choice in Introductory Programming Courses at Australasian and UK Universities. 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, 852–857. https://doi.org/10.1145/3159450.3159547

[56] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education* 13, 4 (2013), 1–40. https://doi.org/10.1145/2534973

[57] Emillie Thiselton and Christoph Treude. 2019. Enhancing Python Compiler Error Messages via Stack. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* IEEE, Piscataway, New Jersey, 1–12.

[58] V. Javier Traver. 2010. On Compiler Error Messages: What They Say and What They Mean. *Advances in Human-Computer Interaction* 2010 (2010), 1–26. https://doi.org/10.1155/2010/602570

[59] Johathan Turner. 2016. Shape of Errors to Come. https://blog.rust-lang.org/2016/08/10/Shape-of-errors-to-come.html

[60] Richard L. Wexelblat. 1976. Maxims for Malfeasant Designers, or How to Design Languages to Make Programming As Difficult As Possible. In *Proceedings of the 2nd International Conference on Software Engineering* (San Francisco, California, USA) *(ICSE '76)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 331–336. http://dl.acm.org/citation.cfm?id=800253.807695

[61] 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, Piscataway, New Jersey, 318–322.

[62] John Wrenn and Shriram Krishnamurthi. 2017. Error Messages are Classifiers: A Process to Design and Evaluate Error Messages. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software.* ACM New York, NY, USA, Vancouver, BC, Canada, 134–147. https://doi.org/10.1145/3133850.3133862

[63] Stelios Xinogalos, Maya Satratzemi, and Vassilios Dagdilelis. 2006. An Introduction to Object-Oriented Programming with a Didactic Microworld: objectKarel. *Computers and Education* 47, 2 (2006), 148–171. https://doi.org/10.1016/j.compedu.2004.09.005