

Enhancing Python Compiler Error Messages via Stack Overflow

Emillie Thiselton and Christoph Treude

School of Computer Science

University of Adelaide

emillie.thiselton@gmail.com, christoph.treude@adelaide.edu.au

Abstract—Background: Compilers tend to produce cryptic and uninformative error messages, leaving programmers confused and requiring them to spend precious time to resolve the underlying error. To find help, programmers often take to online question-and-answer forums such as Stack Overflow to start discussion threads about the errors they encountered.

Aims: We conjecture that information from Stack Overflow threads which discuss compiler errors can be automatically collected and repackaged to provide programmers with enhanced compiler error messages, thus saving programmers’ time and energy.

Method: We present Pycee, a plugin integrated with the popular Sublime Text IDE to provide enhanced compiler error messages for the Python programming language. Pycee automatically queries Stack Overflow to provide customised and summarised information within the IDE. We evaluated two Pycee variants through a think-aloud user study during which 16 programmers completed Python programming tasks while using Pycee.

Results: The majority of participants agreed that Pycee was helpful while completing the study tasks. When compared to a baseline relying on the official Python documentation to enhance compiler error messages, participants generally preferred Pycee in terms of helpfulness, citing concrete suggestions for fixes and example code as major benefits.

Conclusions: Our results confirm that data from online sources such as Stack Overflow can be successfully used to automatically enhance compiler error messages. Our work opens up venues for future work to further enhance compiler error messages as well as to automatically reuse content from Stack Overflow for other aspects of programming.

Index Terms—Compiler errors, Stack Overflow, think-aloud

I. INTRODUCTION

In Richard L. Wexelblat’s 1976 paper “Maxims for malfeasant designers, or how to design languages to make programming as difficult as possible” [1], published at the second International Conference on Software Engineering, the author facetiously proposes “cryptic diagnostics” as one way of maximising difficulty for the user, arguing that a useless compiler error message should only state the effect of an error instead of its cause. Unfortunately, the reality is often not far from this tongue-in-cheek proposal, and not much has changed in terms of the helpfulness of compiler error messages in the meantime: Programmers often encounter cryptic compiler error messages that are difficult to understand

and thus difficult to resolve [2], compiler error messages are cryptic, uninformative, terse, and misleading and pose a significant barrier to progress [3], and they fail to convey information accurately [4].

When faced with cryptic error messages, Wexelblat suggested that “at best, the programmer can get more information from a manual. More often, there is no help available”. This aspect has changed significantly since the mid-70s with the availability of software documentation from a multitude of sources on the Internet. For example, on the question-and-answer forum Stack Overflow,¹ almost 15,000 questions have been tagged with `compiler-error`,² with many more questions requesting help in understanding and addressing a compiler error without the explicit tag—a simple search for compiler error on Stack Overflow returns more than 350,000 results.³ Past work has found that questions which include a specific error message are the fourth most common question type on the site [5] and that debug-corrective questions are common [6], [7].

Following work on automatically combining data from multiple sources related to software development with the ultimate goal of making programmers more productive (e.g., [8]–[11]), in this paper, we introduce PYCEE (Python Compiler Error Enhancer), a plugin for the popular Sublime Text IDE⁴ which automatically augments Python compiler error messages with data from the question-and-answer website Stack Overflow to provide programmers with additional information in concise and summarised form and to offer concrete suggestions for error resolution. We chose to target Python since it is the fastest-growing major programming language today, edging out Java according to the recently published results of the 2019 Stack Overflow Developer Survey.⁵ In addition, other programming languages, such as Java [12], have already been the target of research to enhance compiler error messages, while we are not aware of any similar effort for Python.

Our plugin PYCEE parses a Python compiler error message automatically as soon as a programmer encounters an error. PYCEE then constructs a Stack Overflow query and uses query expansion and reformulation depending on the error type, e.g.,

¹<https://stackoverflow.com/>

²<https://stackoverflow.com/questions/tagged/compiler-errors>

³<https://stackoverflow.com/search?q=compiler+error>

⁴<https://www.sublimetext.com/>

⁵<https://insights.stackoverflow.com/survey/2019>

```

13 def game(turnPlayerA, turnPlayerB):
14     if turnPlayerA == "rock" and turnPlayerB == "scissors":
15         print("PlayerA wins")
16         elif turnPlayerA == "scissors" and turnPlayerB == "paper":
17             print("PlayerA wins")
18
19     else if turnPlayerA == "paper" and turnPlayerB == "rock":
20         print("PlayerA wins")
21
22

```

File "/Users/emilliethiselton/Documents/Uni/2018/Adv Topics Enhancement/Testing/Questions/3-8.py", line 16
else if turnPlayerA == "scissors" and turnPlayerB == "paper":
^
SyntaxError: invalid syntax

In python "else if" is spelled "elif".
Also, you need a colon after the elif and the else.
I had the same problem, when I first started (in the last couple of weeks).
def function(a):
 if a == '1':
 print('1a')
 elif a == '2':
 print('2a')

Fig. 1. Screenshot of PYCEE. The first few lines on white background show the original compiler error message produced by Python, the additional lines show the enhanced error message produced by PYCEE. The message provided by PYCEE is a summary of Stack Overflow answer 2395167. Note that in the screenshot, the offending line has already been corrected.

by adding related verbs and syntax from other programming languages to the query. After selecting an answer from Stack Overflow, PYCEE produces a customised summary of the text and code fragments in the answer and displays the resulting enhanced compiler error message in the Sublime Text IDE.

We evaluated PYCEE through a think-aloud user study during which 16 participants—fourteen professionals and two students—completed programming tasks while using two variants of PYCEE: the original PYCEE which retrieves its data from Stack Overflow and PYCEEDOC which retrieves its data from the official Python documentation. In total, our participants encountered 115 compiler errors during the study. The majority of our participants agreed that PYCEE was helpful while completing the study tasks, citing concrete suggestions for fixes and example code as major benefits. PYCEE generally outperformed PYCEEDOC in terms of perceived helpfulness. Our findings confirm that external knowledge sources, such as Stack Overflow, are not only helpful as reference documents, but they can also be harvested automatically to enhance compiler error messages inside an IDE. PYCEE is available as an open-source project on GitHub.⁶

II. MOTIVATING EXAMPLES

In this section, we present two examples of PYCEE in action, taken from our user study (cf. Section IV).

a) *Invalid syntax*: Figure 1 shows a screenshot of one of our study participants using PYCEE. As shown in the Figure, the participant was not familiar with Python syntax and wrote `else if` instead of `elif` in a conditional statement aimed

at implementing Rock-paper-scissors.⁷ The Python compiler responded with `SyntaxError: invalid syntax` (first four lines on white background in Figure 1), but did not provide any concrete help on how to solve the error. The remaining lines on white background in Figure 1 show PYCEE’s output in this scenario, clearly stating how to fix the error (In python “else if” is spelled “elif”) and providing a code fragment as example. The output produced by PYCEE is a summary of Stack Overflow answer 2395167,⁸ containing a subset of the answer’s sentences as well as a part of the answer’s code fragment.

b) *‘list’ object is not callable*: The following two code fragments show another example of a compiler error encountered by one of our participants who was working on a function to deduplicate lists. The compiler detected a `TypeError`, explaining that ‘list’ object is not callable.

```

list = [3, 3, 5, 7, 7, 9, 11, 11]
new_list = list(dict.fromkeys(list))

print(new_list)

```

```

Running python -u "/Users/.../2-14.py"
From compiler:
Traceback (most recent call last):
  File "/Users/.../2-14.py", line 18, in <module>
    new_list = list(dict.fromkeys(list))
TypeError: 'list' object is not callable

```

⁷Note that the `else if` instance in line 16 had already been corrected by the time the screenshot was taken—line 19 shows an uncorrected one.

⁸<https://stackoverflow.com/a/2395167>

⁶<https://github.com/EmillieT/Pycee>

PYCEE used text and code from Stack Overflow answer 12836173⁹ to enhance this message, recommending to not use tuple, list or other special names as a variable name, along with an example where `list` had been replaced with `1`:

```
It should work fine.
Don't use tuple, list or other special names as
  a variable name.
It's probably what's causing your problem.
>>> l = [4,5,6]
>>> tuple(l)
(4, 5, 6)
```

In this case, PYCEE did not summarise the answer since it contains fewer than five sentences. In the next section, we describe how PYCEE works.

III. PYCEE

PYCEE works in two phases: In the first phase, the Python compiler error message is parsed and used to construct a query for Stack Overflow. In the second phase, an answer from Stack Overflow is selected, customised, and summarised. We describe the details of this process in this section.

A. Compiler Error Message Parsing and Query Construction

As soon as the user encounters a Python compiler error while working in the Sublime Text IDE, PYCEE parses the corresponding compiler error message and determines the name of the affected Python file as well as the libraries which the user had imported, the error type (e.g., `SyntaxError`), and whether the compiler returned a specific error message (e.g., `EOL while scanning string literal`). PYCEE then constructs a Stack Overflow query, with the exact content of the query depending on the type of compiler error. The following settings were used in the final evaluation described in Section IV, and they are the result of extensive experimentation and a preliminary evaluation with student participants. Note that the current implementation of PYCEE queries Stack Overflow every time an error is raised. Caching responses to common errors is part of our future work.

AttributeError, NameError. In the case of an `AttributeError` or a `NameError`, PYCEE uses ideas from query reformulation [13] and query expansion [14] to construct a suitable Stack Overflow query. PYCEE first extracts the word(s) identified by the compiler as problematic, i.e., the ones surrounded by single quotation marks (e.g., `module` and `Number` in the error message `'module' object has no attribute 'Number'`). Following Stefik and Siebert's conjecture that syntactical variations of programming language constructs might affect accuracy among programmers [15], PYCEE then looks up each word in an online resource cataloguing syntax across programming languages.¹⁰ PYCEE attempts to locate each word in one of the tables in this resource, and if successful, replaces

the word with the most frequently occurring word in the same table while removing any non-letters. Adopting the insights of related work with regard to the importance of tasks (e.g., [16])—programmers are usually not interested in a concept by itself, but work with the concept as part of an action or task—PYCEE then attempts to find actions associated with each word. We use task phrases extracted by TaskNav [17], [18] from the titles of the one million most recent Stack Overflow threads tagged with `python` as input data, and determine the verb most commonly associated with each word. In addition, the following Python data types are replaced by their English equivalent (e.g., `str` → `String`) during query construction: `int`, `bool`, `str`, and `dict`.¹¹ Words are replaced with their most common domain-specific synonym, using the SEWordSim database [19]. The final search query is then expanded to contain the error type, the words, and the associated verbs.

SyntaxError. In the case of a `SyntaxError`, PYCEE first determines whether the error might stem from a common Python programming mistake [20], i.e., mismatched quotes, mismatched brackets, or incorrect syntax for `for`-loops, `while`-loops, and conditionals. In these cases, PYCEE adds the terms quotation marks, bracket meanings, `for loop`, `while loop`, and `else if syntax` to the query, respectively. Using a catalogue of Python keywords and builtins, PYCEE attempts to fix typos if the word similarity is at least 0.6 using Python's `get_close_matches` function.¹² If the error was not caused by a common mistake, the search query is `SyntaxError: invalid syntax`.

TypeError. In the case of a `TypeError`, PYCEE uses the phrase `must have first callable argument` as Stack Overflow query if the compiler error message contains the words `the first argument must be callable`, and it removes the error type from the query if the compiler error message contains the phrase `not all arguments converted during string formatting`. These two exceptions were added as a result of our preliminary user study. In all other cases, the original error message is used for the query, including error type and description.

IndentationError, TabError. In the case of an `IndentationError` or a `TabError`, PYCEE performs a Stack Overflow query with the error description only, i.e., not including the error type.

KeyError. In the case of a `KeyError`, PYCEE queries Stack Overflow using only the error type, but not its description.

All other cases. For all other cases, PYCEE queries Stack Overflow using the error type and its description.

We do not include code fragments as part of the Stack Overflow queries since past work has shown that the Stack Overflow search does not handle code fragments well [21]. We encourage readers to inspect PYCEE's source code⁶ for additional details.

⁹<https://stackoverflow.com/a/12836173>

¹⁰<http://rigaux.org/language-study/syntax-across-languages.html>

¹¹Note that the names of other datatypes which can be retrieved using the `dir(__builtins__)` command, such as `tuple`, are already English words.

¹²https://docs.python.org/2/library/difflib.html#difflib.get_close_matches

B. Answer Selection, Customisation, and Summarisation

The Stack Overflow query is configured to only return threads tagged with `python` which contain at least one answer, sorted by relevance according to the Stack Overflow search algorithm.¹³ PYCEE considers the first page of search results, i.e., up to ten Stack Overflow threads, and selects the first accepted answer in these ten threads for further processing. If no answer has been accepted, the answer with the highest score is selected, provided its score is greater than zero. If no such answer is available, PYCEE does not produce an enhanced compiler error message.

The selected answer is customised and summarised as follows:

Customisation. PYCEE locates error messages in the answer’s code fragments, identifies the line which caused the error, and replaces any error message in the answer’s code with the compiler error message encountered by the user to better fit the code to the user’s situation. If the error is a `SyntaxError`, PYCEE uses the arrow (`^`) position in the compiler error message to verify that the compiler has identified the correct error line. If that is not the case and if the Stack Overflow answer mentions an offending line (the erroneous line), PYCEE replaces this line with the selected line from the answer.

Summarisation. PYCEE summarises the selected answer using Luhn’s summarisation algorithm [22] to at most four sentences, following the advice of Nienaltowski et al. [23] who reported that longer error messages do not benefit programmers. We experimented with different summarisation algorithms, aiming to maximise the quality criteria of understandability, completeness, and efficiency [24], and found Luhn’s algorithms to work best on our data. Special characters are converted into a more user-friendly format (e.g., `>` \rightarrow `>`), and unnecessary formatting and padding are removed from the summary. Note that while the goal of the summarisation is to extract parts of posts which are most relevant to an error, there is of course no guarantee that the summary will only contain relevant content.

PYCEE then returns the result and displays it below the original compiler error message in the Sublime Text IDE, cf. Figure 1.

IV. EVALUATION METHODOLOGY

In this section, we outline our evaluation methodology for PYCEE in terms of research questions and data collection and analysis.

A. Research Questions

While some researchers have identified common problems with compiler error messages (e.g., [2]–[4], [12]), not much of this work has focused on the Python programming language, with Guo’s work [20] as a notable exception. However, the focus of Guo’s work is not on programmer perceptions of

compiler error messages. Therefore, and to establish a baseline for PYCEE, with our first research questions, we ask:

RQ1 How do programmers perceive Python compiler error messages?

After establishing this baseline, our next research question analogously investigates the compiler error messages produced by PYCEE:

RQ2 How do programmers perceive working with PYCEE?

One of the key features of PYCEE is its reliance on Stack Overflow as a data source for enhancing compiler error messages. To investigate the impact of this feature, we compare PYCEE to a baseline with similar functionality but with the official Python documentation as data source. We conduct this comparison in terms of perceived helpfulness, perceived potential time savings, and programmer preferences, as captured by our last research question:

RQ3 How do programmers perceive the two PYCEE variants?

RQ3.1 How do programmers perceive the helpfulness of the PYCEE variants?

RQ3.2 How do programmers perceive the potential time savings of the PYCEE variants?

RQ3.3 Which PYCEE variant do programmers prefer and why?

B. Data Collection

To answer our research questions, we conducted a think-aloud user study during which 16 participants—fourteen professional programmers and two students—used two PYCEE variants while completing programming tasks in Python. In the following, we characterise the study protocol, tasks, and participants, and we introduce the baseline tool PYCEEDOC.

1) *Study Protocol:* Table I shows the questions that we asked each participant before, during, and after the study. To ensure that all participants had the same experience and used the same versions of Sublime Text and Python as well as the same set of pre-installed libraries, all participants used the Sublime Text IDE plugins PYCEE and PYCEEDOC by remotely connecting to the first author’s machine using the TeamViewer remote access software.¹⁴

All studies started with questions aimed at collecting demographic information regarding programming experience, programming job, preferred code editor, and Python experience, followed by questions aimed at answering our first research question about programmers’ perceptions of Python compiler error messages and their usual debugging processes.

Participants were then given at least one Python task (see below for task selection) to solve with each variant of PYCEE enabled, with the order in which the variants were chosen alternating between participants. Participants were generally limited to 20 minutes per task, however, if they were willing to spend more than one hour for the overall study session, they were allowed to spend more time on each individual task. If a participant had not written any code five minutes into a

¹³<https://meta.stackoverflow.com/questions/355532/how-does-sort-by-relevance-work>

¹⁴<https://www.teamviewer.com/en/>

TABLE I
QUESTIONS ASKED DURING THE USER STUDY

	question	answer
Before	How many years programming experience do you have?	int
	Do you use programming for your job? If so, what do you do?	description
	How much experience do you have with Python?	int, description
	Do you believe Python compiler error messages provide enough information?	yes/no, description
	What resources do you usually reference when debugging code?	description
	Do you believe the information in these resources usually assists you in solving your problem?	yes/no, description
During	What do you usually use to edit code?	list
	What are you doing / thinking now? (prompt if required)	extended answer
	Overall, do you believe the plugin was helpful?	agreement scale
	Why?	extended answer
	Overall, do you believe the plugin saved time?	agreement scale
	Why?	extended answer
	Overall, were you satisfied with the information provided by the plugin?	agreement scale
Why?	extended answer	
After	What did you like and/or dislike about the plugin?	extended answer
	Which plugin did you prefer?	PYCEE/PYCEEDOC
	Why did you prefer this plugin?	extended answer
	What could make this plugin better?	extended answer
	What would the ideal debugging plugin/tool do?	extended answer
	Any other comments?	extended answer

task or verbally expressed that they had no idea what the task was asking them to do or how to solve it, they were offered another task. If a participant did not encounter any compiler error while completing a task, they were given another task until they encountered a compiler error to ensure that both PYCEE variants were used by all participants. Depending on how long participants took per task, they were given the choice of another task or a chance to improve their existing solution until the end of the session.

To best replicate a real programming setting, participants were explicitly given permission to execute code and to refer to the Internet while completing the study tasks. We clarified that the purpose of the study was not to assess their programming ability and we encouraged them to think aloud during the study. To not bias participants with leading questions or hints, we only prompted them by asking the pre-defined questions “What are you doing / thinking now?” during their work if needed.

After participants had used a PYCEE variant, we inquired about its perceived helpfulness, time savings, satisfaction, and preferences along with the corresponding reasons. To ensure that responses to these questions were unbiased, participants were told no information about the PYCEE variants other than they provide “enhanced error messages”. At the end of each study session, we asked which PYCEE variant they preferred and how the tool could be further improved.

All study sessions were video and audio recorded, and extensive notes were taken during each session. PYCEE recorded all compiler errors encountered during a session and stored information regarding the error type and description. Note that we cannot make raw data available to protect our participants’ privacy.

2) *Study Tasks*: To use objective criteria for the creation of study tasks to the extent possible and not bias the tasks toward

TABLE II
ASSIGNMENT OF TASKS TO PARTICIPANTS, STARS INDICATE DIFFICULTY. P_6 AND P_7 SKIPPED ONE TASK EACH. P_{13} SKIPPED TWO TASKS AND PREFERRED TO EXECUTE CODE FOR THEIR OWN TASK FOR PYCEEDOC.

	PYCEEDOC	PYCEE
P_1	Password Generator****	Decode A Web Page****
P_2	Password Generator****	Decode A Web Page Two****
		Decode A Web Page****
P_3	Password Generator****	Decode A Web Page Two****
		Decode A Web Page****
P_4	Decode A Web Page****	Password Generator****
P_5	Tic Tac Toe Game***	Decode A Web Page****
		Rock Paper Scissors***
P_6	Check Primality Functions***	Reverse Word Order***
P_7	Check Primality Functions***	Tic Tac Toe Game***
P_8	Guessing Game One***	Cows And Bulls***
P_9	Guessing Game One***	Password Generator****
P_{10}	Rock Paper Scissors***	Guessing Game One***
P_{11}	Tic Tac Toe Game***	Guessing Game One***
P_{12}	Guessing Game Two***	Cows And Bulls***
P_{13}	<i>Own</i>	Guessing Game One***
P_{14}	List Overlap Comprehensions**	Fibonacci**
		List Remove Duplicates**
P_{15}	Tic Tac Toe Game***	Cows And Bulls***
P_{16}	List Comprehensions**	String Lists**

PYCEE, we chose study tasks from the Practice Python website, a resource aimed at providing small, short, and relevant introductory Python programming exercises for beginners.¹⁵ The site contains 25 Python tasks after merging sub-tasks which are part of a larger task and excluding tasks not suitable for the study, categorised by their difficulty into four categories (from one chilli up to four chillies, with increasing difficulty). We eventually excluded two tasks which required stable and fast Internet access (decoding a web page, parts one and two) after we noticed that the Internet connection speed of participants varied. We also excluded one task that required

¹⁵<https://www.practicepython.org/>

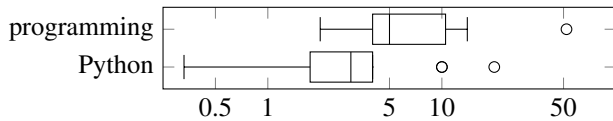


Fig. 2. Participant experience in years (log scale)

TABLE III
RESOURCES USED WHEN DEBUGGING CODE

resource	participants
Stack Overflow	13
Google	10
Official Documentation	6
Compiler	1

graphing of data, which did not work in our setup with Sublime Text. To ensure that participants did not find the tasks too trivial and to give us a realistic chance at encountering compiler errors, tasks were allocated to participants according to their difficulty rating: For participants who indicated at most half a year of Python experience, we allocated two-chilli tasks, and for participants who indicated at least half a year of Python experience, we allocated three- and four-chilli tasks. Within these constraints, tasks were assigned randomly to participants. Table II shows the assignment.

3) *Study Participants*: We advertised our study on social media and through professional contacts and we recruited participants through the freelancing website Upwork,¹⁶ resulting in a total of 16 participants. The majority of participants ($14/16 = 88\%$) were professional programmers who use programming as part of their job, the remaining two were students. Job titles ranged from software developer and project manager to CTO and big data engineer. Figure 2 shows the experience of participants in years. Participants had a median of five years of programming experience, with a minimum of two years and a maximum of 52 years, and they had a median of three years of Python experience, with a minimum of four months and a maximum of 20 years.

As shown in Table III, the majority of participants indicated to resort to Stack Overflow ($13/16 = 81\%$) and Google ($10/16 = 63\%$) when debugging code. Other resources mentioned included colleagues, textbooks, tutorial sites, and discussion forums. Table IV shows the code editors usually used by participants, with Sublime Text ($7/16 = 44\%$) and PyCharm ($6/16 = 38\%$) being the most common. Other editors mentioned included Eclipse, IntelliJ, Atom, and Vi.

¹⁶<https://www.upwork.com/>

TABLE IV
CODE EDITORS USUALLY USED BY PARTICIPANTS

resource	participants
Sublime Text	7
PyCharm	6
Visual Studio	4
Vim	2

4) *PYCEEDOC*: To be able to assess the helpfulness of using content from Stack Overflow to enhance Python compiler error messages, we implemented a baseline called PYCEEDOC which accesses the official Python documentation for each compiler error instead of Stack Overflow. When encountering a compiler error, PYCEEDOC reproduces the corresponding content from the Exceptions page of the Python API¹⁷ in the Sublime Text IDE. Note that PYCEEDOC does not make use of the description of compiler error messages, but only uses the error type since there is only exactly one explanation available in the official documentation for each error type. To be compatible with the Sublime Text IDE, we removed links from the documentation as well as any version changes. As an example, the documentation for `IndentationError` states: “Base class for syntax errors related to incorrect indentation. This is a subclass of `SyntaxError`”.¹⁸

C. Data Analysis

In this section, we describe how we analysed the collected data to answer each of our research questions.

1) *RQ1: Programmer perceptions of Python compiler error messages*: To answer our first research question, we analysed participant responses about general perceptions of Python error messages and the resources typically referenced when debugging (cf. Table I). We conducted open coding following the definition of Strauss and Corbin [25], i.e., generating categories and considering their variations response by response [26]. The coding was done by the second author using NVivo [27] and verified by the first author. Where applicable, we quote participants when presenting findings to increase traceability to raw data. We show a subset of codes in the following text in *italics*, and we indicate how many participants mentioned the particular code in superscript. Note that these numbers only indicate how much evidence the data analysis yielded for each code, they do not necessarily indicate the importance of a code since we did not explicitly ask all participants about each code specifically.

2) *RQ2: Programmer perceptions while using PYCEE*: To answer our second research question, we analysed the notes we took and the screen and audio recordings of participants using PYCEE during the think-aloud part of the study. Since these notes were taken separately for each of the compiler errors encountered by participants during the study, we analysed the notes compiler error by compiler error during open coding. The coding was conducted again by the second author and verified by the first author.

3) *RQ3: Programmer perceptions of PYCEE variants*: To answer our last research question, we analysed the participants’ responses regarding helpfulness, time savings, satisfaction, and preference of each PYCEE variant. In addition to reporting the responses on the Likert scales, we qualitatively analysed the reasons that participants gave for their opinions, using the previously described qualitative analysis process.

¹⁷<https://docs.python.org/3/library/exceptions.html>

¹⁸<https://docs.python.org/3/library/exceptions.html#IndentationError>

V. FINDINGS

In this section, we describe our findings for each research question.

A. RQ1: Programmer perceptions of Python compiler error messages

Most participants indicated that Python compiler error messages had room for improvement, citing issues when encountering complex problems and the need to look at other resources. One of the codes which emerged from our qualitative analysis of participants' responses regarding their perception of Python compiler error messages was that they are *bad for beginners*⁽⁴⁾. For example, when asked whether Python compiler error messages provide sufficient information, P_4 explained that "Only to people who are familiar with the error" and P_8 answered "Generally no but yes after years of experience".

The majority of participants ($12/16 = 75\%$) indicated that information in other resources, such as Stack Overflow, usually assists them in solving their problems. However, looking up information in external resources is not trivial, as explained by P_{15} : "Most of the time but can be consuming for unique cases". These results support our initial motivation in building PYCEE to bridge the gap between Python compiler errors and documentation found in external resources.

SUMMARY RQ1

The majority of participants perceived Python compiler error messages to have room for improvement, in particular for beginners. External resources could usually assist participants when encountering a compiler error, but this might be time-consuming.

B. RQ2: Programmer perceptions while using PYCEE

Table V shows the compiler errors encountered by our study participants while using PYCEE and its baseline variant PYCEEDOC. In total, participants encountered 115 compiler errors, 62 while working with PYCEE and 53 while working with PYCEEDOC. The most common error type was `SyntaxError` followed by `NameError`. All participants encountered at least one compiler error per PYCEE variant with a median number of two and a half errors while using PYCEEDOC and four errors while using PYCEE. Error messages which only differed in their variable names have been grouped and errors caused by Sublime Text or PYCEE, such as connection errors, have been filtered out.

When using the baseline tool PYCEEDOC, several participants criticised that the enhanced compiler error message—directly copied from the Python API documentation—contained *too much information*⁽⁶⁾. P_9 for example encountered a `TypeError` and did not bother to read all information provided by the tool, stating "It's a bit too long". Another common complaint was that the output of PYCEEDOC was *too generic*⁽⁶⁾, e.g., P_1 : "It did not tell me what I should be doing". Participants also noted that the documentation was *too*

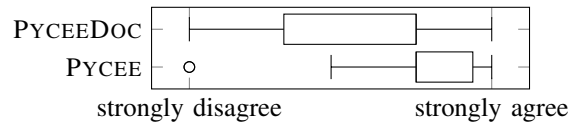


Fig. 3. Perceived helpfulness of PYCEE variants

formal⁽³⁾, e.g., P_3 noted "This is not clear at all, I want plain English" after encountering an `IndentationError`. There was a general feeling among participants that a tool for enhancing compiler error messages should focus on common errors, as stated by P_3 : "You should use common (basic) errors as test cases when testing this plugin".

When using PYCEE, participants commended the tool for *suggesting a fix or giving an example*⁽⁴⁾ in various scenarios. For example, P_{14} , after encountering a `SyntaxError`, stated "It's helpful and suggests how to fix the problem" and P_{12} , also after encountering a `SyntaxError`, added "I like the language and the code examples". These comments show that at least in some scenarios, PYCEE was able to address the main weaknesses participants perceived when working with the baseline tool: a lack of specificity, formal language, and too much information. Participants further noted the additional context provided by PYCEE, as expressed by P_4 : "I liked the context and that it explains the specific location". To some extent, the success of PYCEE *depended on the type of error*⁽³⁾, as noted by P_5 : "It has useful information but not for this case [a `TypeError`]". In some cases, the information provided by PYCEE was *incorrect*⁽³⁾. For example, P_1 encountered a `NameError` and stated: "It provided me with the wrong information, it was a global variable but I was told to look in local variables".

SUMMARY RQ2

During the think-aloud part of the study, participants encountered a total of 115 enhanced compiler error messages. Common perceptions about the compiler error messages generated by the baseline tool PYCEEDOC were that these messages contained too much information and were too generic. Compiler error messages generated by PYCEE were commended for including suggestions for fixes and examples, but were not perceived to be correct in all cases.

C. RQ3: Programmer perceptions of PYCEE variants

After completing the think-aloud portion of the study, we asked participants about their perceptions of the PYCEE variants in terms of helpfulness, time savings, and satisfaction, along with their preference for one variant over the other.

Figure 3 shows the results for helpfulness on a 5-point Likert scale. All but three participants (i.e., $13/16 = 81\%$) agreed or strongly agreed with PYCEE's helpfulness. The number for the baseline tool PYCEEDOC is slightly lower at $11/16 = 69\%$. For PYCEE, participants explained their rating with *successful instances*⁽⁴⁾ where the tool had worked well, e.g., P_{16} stated: "I liked how the `NameError` message showed

TABLE V
COMPILER ERRORS ENCOUNTERED BY STUDY PARTICIPANTS

type	description	PYCEEDOC		PYCEE	
		occurrences	by type	occurrences	by type
AttributeError	'X' object has no attribute 'Y'	4	4	6	6
ImportError	cannot import name 'X'	1	4	–	2
	No module named 'X'	3		2	
IndentationError	expected an indented block	1	3	8	10
	unindent does not match any outer indentation level	2		2	
IndexError	list index out of range	–	3	2	2
	'X' index out of range	3		–	
KeyError	'class'	–	–	1	1
NameError	global name 'X' is not defined	1	11	1	13
	name 'X' is not defined	10		12	
SyntaxError	EOL while scanning string literal	1	19	1	21
	invalid syntax	18		20	
TypeError	can only concatenate list (not "str") to list	1	7	–	5
	cannot concatenate 'str' and 'int' objects	1		1	
	'int' object is not iterable	1		2	
	'list' object is not callable	–		1	
	'NoneType' object is not callable	2		–	
	unsupported operand type(s) for +: 'X' and 'Y'	1		–	
	'X' takes exactly 'Y' arguments ('Z' given)	1		1	
ValueError	invalid literal for int() with base 10: 'X'	–	–	1	1
ZeroDivisionError	integer division or modulo by zero	2	2	1	1
sum		53	53	62	62

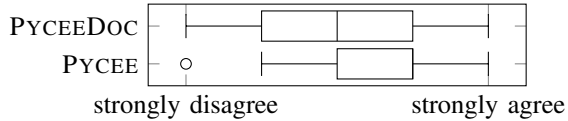


Fig. 4. Perceived time savings of PYCEE variants

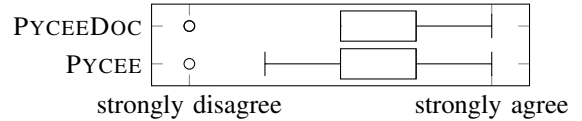


Fig. 5. User satisfaction for PYCEE variants

how to check for the error”. For PYCEEDOC, participants pointed at *novices as a target audience*⁽⁴⁾, e.g., P_2 explained: “It wasn’t helpful for me but it would be for novices”.

As shown in Figure 4, there were no discernible differences in the distributions of participant perceptions regarding the potential for time savings between the two PYCEE variants. $10/16 = 63\%$ participants for PYCEE and $6/16 = 38\%$ participants for PYCEEDOC agreed or strongly agreed with the statement that the tool saved time. Note that these results might be biased against the PYCEE variants due to slow Internet connection of some participants as a result of the remote study setup. The reasons which participants mentioned for their perceptions did not differ much between the PYCEE variants: Some participants found that they *did not have to look for information elsewhere*⁽⁴⁾, such as P_{16} who stated “Yes, I could skip using the browser”. In other cases where PYCEE was not helpful, participants still needed to search elsewhere, e.g., P_{15} : “I still need to search how to fix the problem”.

In terms of user satisfaction, both PYCEE variants scored equally well as shown in Figure 5. In both cases, $9/16 = 56\%$ agreed or strongly agreed with the statement that they were satisfied with the tool. For PYCEE, participants justified their

answers by referring to the *style of the enhanced error messages*⁽⁴⁾ and the *presence of code examples*⁽⁴⁾. For example, P_4 explained: “I liked the code examples and full sentences in normal English, written for humans”. On the other hand, one of the disadvantages of PYCEE is that it relies on information from Stack Overflow which may or may not be correct. Several participants mentioned this trade-off, e.g., P_9 : “The additional information is helpful and saves time but the information was not always correct and then you lose time”. For PYCEEDOC, participants commended the *additional context*⁽⁵⁾ added in the enhanced compiler error message, e.g., P_2 stated “Some of it expands the information and provides context ... and focuses on the problem”. However, participants complained about a *lack of direction*⁽⁴⁾ in fixing errors, e.g., P_{13} : “It says what has happened but not how to fix it”.

Finally, we asked participants which PYCEE variant they preferred. As shown in Table VI, there is a preference toward PYCEE, with an equally large number of participants not having a preference. Among those who preferred PYCEE, i.e., the variant relying on Stack Overflow, participants mentioned reasons such as “It felt more personal” (P_4) and the *presence of examples*⁽⁴⁾, e.g., P_9 : “Both seemed the same, the examples

TABLE VI
PREFERRED PYCEE VARIANT

variant	participants
PYCEE	7
PYCEEDOC	2
no preference	7

from the plugin were the best part”. Note that all examples came from Stack Overflow and were therefore only available in PYCEE.

SUMMARY RQ3

The majority of participants agreed that PYCEE is helpful and that it saves time. When compared to the baseline PYCEEDOC, participants generally preferred PYCEE in terms of helpfulness, referring to concrete suggestions on how to fix compiler errors and code examples as strong points.

VI. DISCUSSION AND OPEN RESEARCH CHALLENGES

Our work has provided evidence that it is indeed possible to use data from online sources such as Stack Overflow to automatically enhance compiler error messages. The majority of participants agreed that PYCEE is helpful and that it saves time, primarily thanks to the inclusion of code examples and concrete suggestions on how to fix errors in the automatically generated compiler error messages. A trade-off we encountered as part of this work is the potentially low quality of content on Stack Overflow (see for example Ragkhitwetsagul et al.’s recent work on toxic code snippets on Stack Overflow [28]). While Stack Overflow likely contains information on most errors, not all of it is correct or relevant. In one case, P_3 had misspelled the Python print command as `print`. PYCEE provided advice on how to check for the existence of a local variable, using the variable name `myVar` in a code example, which led to additional confusion. In another case, P_{13} attempted to concatenate a string and an integer using the `+` symbol. PYCEE produced a code snippet with a call to Python’s `lambda` function which had no relevance to the solution (a call Python’s `str` function). We will continue to explore these challenges in future work, e.g., by trying to identify minimal working examples on Stack Overflow.

With one of the last questions in our study, we asked participants to describe their hypothetical ideal debugging tool. While P_{11} ’s response sums up the general sentiment well: “It would solve the error for me so I don’t have to do any work”, other participants had more concrete suggestions, such as, a good debugging tool should be helpful in fixing common mistakes (P_{13} : “I would like to see common examples of how the error is thrown and suggested usage of the solution or the function that caused the error”). Current compiler error messages—much like API documentation [29]—focus on covering all possible cases instead of covering the common ones well. Past work has shown that Stack Overflow works the other way around: The crowd is capable of generating a rich

source of content with code examples and discussion that is actively viewed and used by many more developers, but does not usually achieve perfect coverage [30].

Context awareness [31] was mentioned as an important feature of the ideal debugging tool by many participants. Context can refer to relevant links to external resources (P_{11} : “Links to relevant Stack Overflow posts and additional information”) as well as to the user’s code base (P_{12} : “Integration of the user’s code into the plugin would be great”). We tried to improve PYCEE’s context awareness by replacing error messages in the Stack Overflow answer’s code with the compiler error message encountered by the user to better fit code examples to the user’s situation (cf. Section III), but more work is needed to achieve better context awareness in debugging tools.

Many of our participants discussed the visual appearance of compiler error messages, suggesting that error messages should favour visual content over textual content (P_3 : “It should have little text and be visual”), exist in a separate layer on top of the source code (P_5 : “A pop up window with information would be good or when you mouseover the code a message box appears”), be accessible similar to other code elements (P_8 : “I want it to be more invisible and to be usable with keyboard shortcuts”), and be interactive (P_3 : “It should be teachable, the user should be able to interact with it”). Not much related work has focused on the human-computer interaction aspect of how to present compiler errors to users (with Barik et al. [32] and Prather et al. [33] as notable exceptions). Our participant responses suggest that more work is required in this area.

VII. THREATS TO VALIDITY

Similar to other empirical studies, there are threats which may affect the validity of our results.

Threats to the *construct validity* correspond to the appropriateness of the evaluation metrics. We evaluated the PYCEE variants in terms of their perceived helpfulness, their perceived potential for time saving, and the user satisfaction. Similar metrics have been used in many other studies before (e.g., [34]) and these metrics reflect our goals behind developing PYCEE. The data on participant experience in Python which we used to allocate programming tasks to participants was derived from participant responses, and we cannot guarantee that these responses accurately reflect each participant’s experience.

Threats to the *internal validity* compromise our confidence in establishing a relationship between the independent and dependent variables. Participants were not informed what sources each of the PYCEE variants used to ensure that responses would not be biased toward or against one variant. While participants were given the opportunity to use both PYCEE variants for at least 20 minutes each, some only encountered as little as one compiler error per variant. These participants would have only experienced a small subset of PYCEE’s functionality, missing out on PYCEE’s handling of specific error types. In addition, participants did not encounter the same compiler errors while working with the PYCEE variants

since they solved different tasks with each variant. This would likely have been reflected in their answers. During the study, we noticed that some participants had a tendency to discuss what programmers of other skill levels might think of PYCEE instead of focusing on their own programming task. This might have affected their answers, but did likely not affect one variant of PYCEE more than the other. One or two participants correctly guessed the sources used by the PYCEE variants. We did not confirm their guesses until after the study, but this might still have influenced their answers. Some participants encountered errors which they already knew how to solve, which may have affected the way they perceived the PYCEE variants. Since the study was conducted remotely, we cannot guarantee that participants did not search for solutions outside of the study setup.

Threats to *external validity* correspond to the ability to generalise our results. We cannot claim generalisability beyond the Python programming language or the particular implementations of PYCEE and PYCEEDOC used in our study. Recruiting more or different programmers to participate in the study and asking them to work on different tasks may have led to different results. Note that we decided to give participants freedom in terms of how they tackled their programming tasks to create as realistic a scenario as possible—an alternative design in which participants would have been asked to fix a set of given compiler errors would have been more contrived. Some participants were hesitant to write Python code as they felt their programming abilities were being judged. A small number of participants introduced errors to their code on purpose out of curiosity to trigger PYCEE and see its result. These issues might reduce the extent to which our study sessions reflect an actual programming setting.

VIII. RELATED WORK

Work related to PYCEE can be grouped into three categories: compiler error message enhancement, use of Stack Overflow content in IDEs, and summarisation in software engineering.

a) Compiler Error Message Enhancement: Becker [3] suggested that providing enhanced error messages to novices can reduce the future number of error messages received. While the majority of research uses Java compiler errors, Becker et al. [12] discussed the rising popularity of Python as an introductory programming language, suggesting the need for more research. Nienaltowski et al. [23] found that longer error messages do not necessarily benefit students, and that the additional information provided (e.g., error code) may be a cause for additional confusion. Hristova et al. [35] developed an approach to provide enhanced error messages for Java, focused on enhancing the function of a compiler so that the enhanced message heavily references the users' code.

b) Use of Stack Overflow Content in the IDE: Seahawk by Ponzanelli et al. [36] is an Eclipse plugin which automatically formulates queries from the current source code context and presents a ranked and interactive list of Stack Overflow results to the user. A related tool called Prompter was later proposed by the same research group [37]. Cordeiro et

al. [38] developed a tool which integrates the recommendations of question-and-answer web resources related to stack traces into the Eclipse IDE. AutoComment by Wong et al. [39] extracts code-description mappings from Stack Overflow and leverages this information to automatically generate descriptive comments for similar code. NLP2Code by Campbell and Treude [40] and Bing Developer Assistant by Zhang et al. [41] provide code snippets in the IDE via natural language queries, while Treude and Robillard [42] found that less than half of Stack Overflow code snippets are considered to be self-explanatory.

c) Summarisation in Software Engineering: Haiduc et al. [43] found that a combination of text summarisation techniques is most appropriate for source code summarisation. Moreno et al. [24] developed an approach to summarise Java classes, McBurney et al. [44] analysed method calls and leveraged the PageRank algorithm to generate a description of the behaviour of a Java method, and Alqaimi et al. [45] summarised Java lambda expressions. Ying and Robillard [46] developed an approach for the automated summarisation of code fragments, and Buse et al. [47] developed an approach to automatically summarise the conditions of Java exceptions. Rastkar et al. [48] introduced an automated approach which produces a natural language summary describing cross-cutting concerns and how they are implemented. The same research group [49], [50] investigated whether it is possible to summarise bug reports automatically and effectively so that developers can consult summaries instead of entire bug reports.

IX. CONCLUSIONS AND FUTURE WORK

Motivated by the tendency of compilers to produce cryptic and uninformative error messages and the plethora of Stack Overflow threads discussing compiler errors, we have presented PYCEE, a plugin integrated with the Sublime Text IDE to provide enhanced compiler error messages for the Python programming language. PYCEE automatically queries Stack Overflow to provide customised and summarised information about compiler errors within the IDE. Our evaluation through a think-aloud study, during which 16 programmers completed programming tasks while using two PYCEE variants and encountering a total of 115 compiler errors for which PYCEE produced an enhanced compiler error message, showed that the majority of participants agreed that PYCEE was helpful while completing the study tasks. Participants primarily cited the concrete suggestions for fixes and code examples included in the enhanced compiler error messages as major benefits of PYCEE. Our results confirm that data from online sources such as Stack Overflow can be successfully used to automatically enhance compiler error messages.

In addition to improving PYCEE, in particular in terms of its context awareness and its handling of common mistakes, our future work lies in investigating suitable user interfaces for communicating compiler errors to programmers in order to transform error messages from Wexelblat's foreboding "cryptic diagnostics" [1] into tools which can reliably help programmers solve compiler errors.

ACKNOWLEDGEMENTS

The authors would like to thank Greg Wilson for suggesting to build PYCEE and all study participants for their participation. This work was inspired by the International Workshop series on Dynamic Software Documentation, held at McGill's Bellairs Research Institute in February 2017 and February 2018. This work has been supported by the Australian Research Council's Discovery Early Career Researcher Award (DECRA) funding scheme (DE180100153).

REFERENCES

- [1] R. L. Wexelblat, "Maxims for malfeasant designers, or how to design languages to make programming as difficult as possible," in *Proceedings of the International Conference on Software Engineering*, 1976, pp. 331–336.
- [2] V. J. Traver, "On compiler error messages: What they say and what they mean," *Advances in Human-Computer Interaction*, vol. 2010, pp. 3:1–3:26, 2010.
- [3] B. A. Becker, "An effective approach to enhancing compiler error messages," in *Proceedings of the Technical Symposium on Computing Science Education*, 2016, pp. 126–131.
- [4] G. Marceau, K. Fislis, and S. Krishnamurthi, "Mind your language: On novices' interactions with error messages," in *Proceedings of the Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2011, pp. 3–18.
- [5] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (NIER track)," in *Proceedings of the International Conference on Software Engineering*, 2011, pp. 804–807.
- [6] L. B. L. de Souza, E. C. Campos, and M. de Almeida Maia, "Ranking crowd knowledge to assist software development," in *Proceedings of the International Conference on Program Comprehension*, 2014, pp. 72–82.
- [7] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *Proceedings of the International Conference on Software Maintenance*, 2012, pp. 25–34.
- [8] F. M. Delfim, K. V. R. Paixão, D. Cassou, and M. de Almeida Maia, "Redocumenting APIs with crowd knowledge: a coverage analysis based on question types," *Journal of the Brazilian Computer Society*, vol. 22, no. 1, 2016.
- [9] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft, "What information about code snippets is available in different software-related documents? An exploratory study," in *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering*, 2017, pp. 382–386.
- [10] L. Ponzanelli, "Holistic recommender systems for software engineering," in *Companion Proceedings of the International Conference on Software Engineering*, 2014, pp. 686–689.
- [11] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *Proceedings of the International Conference on Software Engineering*, 2016, pp. 392–403.
- [12] B. A. Becker, G. Glanville, R. Iwashima, C. McDonnell, K. Goslin, and C. Mooney, "Effective compiler error message enhancement for novice programming students," *Computer Science Education*, vol. 26, no. 2–3, pp. 148–175, 2016.
- [13] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proceedings of the International Conference on Software Engineering*, 2013, pp. 842–851.
- [14] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via WordNet for effective code search," in *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering*, 2015, pp. 545–549.
- [15] A. Stefik and S. Siebert, "An empirical investigation into programming language syntax," *ACM Transactions on Computing Education*, vol. 13, no. 4, pp. 19:1–19:40, 2013.
- [16] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proceedings of the International Symposium on Foundations of Software Engineering*, 2006, pp. 1–11.
- [17] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 565–581, 2015.
- [18] C. Treude, M. Sicard, M. Klocke, and M. P. Robillard, "Tasknav: Task-based navigation of software documentation," in *Proceedings of the International Conference on Software Engineering - Volume 2*, 2015, pp. 649–652.
- [19] Y. Tian, D. Lo, and J. Lawall, "Sewordsim: Software-specific word similarity database," in *Companion Proceedings of the International Conference on Software Engineering*, 2014, pp. 568–571.
- [20] P. J. Guo, "Online python tutor: Embeddable web-based program visualization for CS education," in *Proceeding of the Technical Symposium on Computer Science Education*, 2013, pp. 579–584.
- [21] M. Monperrus and A. Maia, "Debugging with the crowd: a debug recommendation system based on Stackoverflow," Université Lille 1 - Sciences et Technologies, Tech. Rep. hal-00987395, 2014.
- [22] H. P. Luhn, "The automatic creation of literature abstracts," *IBM Journal of Research and Development*, vol. 2, no. 2, pp. 159–165, 1958.
- [23] M.-H. Nienaltowski, M. Pedroni, and B. Meyer, "Compiler error messages: What can help novices?" in *Proceedings of the Technical Symposium on Computer Science Education*, 2008, pp. 168–172.
- [24] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for Java classes," in *Proceedings of the International Conference on Program Comprehension*, 2013, pp. 23–32.
- [25] A. Strauss and J. Corbin, *Basics of qualitative research: Techniques and procedures for developing grounded theory*, 2nd ed. Sage Publications, Inc., 1998.
- [26] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *Proceedings of the International Conference on Software Engineering*, 2016, pp. 120–131.
- [27] P. Bazeley and K. Jackson, *Qualitative data analysis with NVivo*. Sage Publications Limited, 2013.
- [28] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic code snippets on Stack Overflow," *IEEE Transactions on Software Engineering*, 2019, to appear.
- [29] W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.
- [30] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," Georgia Institute of Technology, Tech. Rep., 2012.
- [31] P. Antunes, V. Heskovic, S. F. Ochoa, and J. A. Pino, "Reviewing the quality of awareness support in collaborative applications," *Journal of Systems and Software*, vol. 89, no. C, pp. 146–169, 2014.
- [32] T. Barik, J. Witschey, B. Johnson, and E. Murphy-Hill, "Compiler error notifications revisited: An interaction-first approach for helping developers more effectively comprehend and resolve error notifications," in *Companion Proceedings of the International Conference on Software Engineering*, 2014, pp. 536–539.
- [33] J. Prather, R. Pettit, K. H. McMurphy, A. Peters, J. Homer, N. Simone, and M. Cohen, "On novices' interaction with compiler error messages: A human factors approach," in *Proceedings of the Conference on International Computing Education Research*, 2017, pp. 74–82.
- [34] A. Seffah, M. Donyace, R. B. Kline, and H. K. Padda, "Usability measurement and metrics: A consolidated model," *Software Quality Journal*, vol. 14, no. 2, pp. 159–178, 2006.
- [35] M. Hristova, A. Misra, M. Rutter, and R. Mercuri, "Identifying and correcting Java programming errors for introductory computer science students," in *Proceedings of the Technical Symposium on Computer Science Education*, 2003, pp. 153–156.
- [36] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack Overflow in the IDE," in *Proceedings of the International Conference on Software Engineering*, 2013, pp. 1295–1298.
- [37] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," in *Proceedings of the Working Conference on Mining Software Repositories*, 2014, pp. 102–111.
- [38] J. Cordeiro, B. Antunes, and P. Gomes, "Context-based recommendation to support problem solving in software development," in *Proceedings of the International Workshop on Recommendation Systems for Software Engineering*, 2012, pp. 85–89.

- [39] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Proceedings of the International Conference on Automated Software Engineering*, 2013, pp. 562–567.
- [40] B. A. Campbell and C. Treude, "NLP2Code: Code snippet content assist via natural language tasks," in *Proceedings of the International Conference on Software Maintenance and Evolution*, 2017, pp. 628–632.
- [41] H. Zhang, A. Jain, G. Khandelwal, C. Kaushik, S. Ge, and W. Hu, "Bing developer assistant: Improving developer productivity by recommending sample code," in *Proceedings of the International Symposium on Foundations of Software Engineering*, 2016, pp. 956–961.
- [42] C. Treude and M. P. Robillard, "Understanding Stack Overflow code fragments," in *Proceedings of the International Conference on Software Maintenance and Evolution*, 2017, pp. 509–513.
- [43] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proceedings of the Working Conference on Reverse Engineering*, 2010, pp. 35–44.
- [44] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for Java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.
- [45] A. Alqaimi, P. Thongtanunam, and C. Treude, "Automatically generating documentation for lambda expressions in Java," in *Proceedings of the International Conference on Mining Software Repositories*, 2019, pp. 310–320.
- [46] A. T. T. Ying and M. P. Robillard, "Code fragment summarization," in *Proceedings of the Joint Meeting on Foundations of Software Engineering*, 2013, pp. 655–658.
- [47] R. P. Buse and W. R. Weimer, "Automatic documentation inference for exceptions," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2008, pp. 273–282.
- [48] S. Rastkar, G. C. Murphy, and A. W. J. Bradley, "Generating natural language summaries for crosscutting source code concerns," in *Proceedings of the International Conference on Software Maintenance*, 2011, pp. 103–112.
- [49] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: A case study of bug reports," in *Proceedings of the International Conference on Software Engineering - Volume 1*, 2010, pp. 505–514.
- [50] —, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.