Final Project

An Empirical Study of the Maintainability of ChatGPT's answers

Thomas Trépanier

Montréal, Canada
thomas.trépanier@polytml.ca

***Abstract*—**
***Index Terms*—llm, maintainability, ChatGPT, software repository mining**

I. Introcution

II. Related Work

III. Methodology

The methodology to answer *RQ1* and *RQ2* is mostly the same and can be divided into 5 steps: (1) data collection, (2) code snippet extraction, (3) code snippets filtering, (4) code smell detection, and (5) code smell analysis. The main difference between the two research questions is the data collection step. For *RQ1*, the data used is the DevGPT dataset provided by the MSR2024 challenge [1]. For *RQ2*, a dataset was created using StackOverflow's API following the methodology described in section III-A.

All of the code and algorithms to extract and process data was written in Python. The code is available on GitHub [1].

A. Data Collection

1) DevGPT Dataset: All of the provided snapshots were used in the study, as well as all of the JSON files in each of the snapshots. The JSON files contain information about the conversations between the developers and the ChatGPT bot.

2) StackOverflow Dataset: In order to build a dataset of StackOverflow's code snippets, the following approach was used:

1) **Fetching Questions from StackOverflow's API:** The first step was to query StackOverflow's API to get a list of Python related questions asked between November 23rd 2018 and November 23rd 2023. These dates were chosen as they represent a recent period of time that would correspond to the kind of code snippets found in the DevGPT dataset.

   Furthermore, the questions had to have at least five answers with one of them being accepted, and having been viewed 1000 times. These criteria were chosen to ensure that the questions were relevant and that the code snippets found in the answers would be of relatively good quality.

2) **Retrieving the answers to the questions:** The second step was to fetch the answers to the questions retrived in step 1. To do so, the *question_id* of each question was used to query StackOverflow's API and retrieve the answers associated with the question. The answers were then stored in a JSON file.

B. Code Snippet Extraction

1) DevGPT Dataset: The DevGPT dataset contains conversations between developers and the ChatGPT bot. Each JSON file contains a data structure called `ChatgptSharing`, itself containing a list of `Conversation`. Each `Conversation` contains the prompt given by the developer, the answer given by ChatGPT, and a data structure called `ListOfCode` containing the code snippets included in ChatGPT's answer.

This `ListOfCode` itself provides the code snippet emitted by ChatGPT as well as the programming language used in the snippet.

In order to build a dataset of all of the code snippets generated by ChatGPT, a program was written to parse all of the JSON files of all of the snapshots and extract all the `ListOfCode` data structures.

2) StackOverflow Dataset: The answers retrieved from StackOverflow's API consisted of HTML bodies containing HTML elements to delimit the code snippet. Therefore, the HTML bodies were parsed using the following Regex to extract the code snippet itself: `<pre><code>(.*?)</code></pre>`.

The snippet was then parsed from its HTML format to a Unicode format and stored in a JSON file.

C. Code Snippets Filtering

Not all code snippets extracted were used for the code smell detection, since not all of them were suited for the analysis. The definition of a *suitable* code snippet is as follows:

[1] https://github.com/ThomasTrepanier/log6307-final-project

1) The code snippet must be in Python. This is because the code smell detection tool used, *PyScent*, is a Python tool.

2) The code snippet must be at least 5 lines long, and contain either a function (defined with the `def` keyword) or a class (defined with the `class` keyword). This is to filter out short answers or code snippets that were not actually Python code, such as JSON objects or library import statements.

3) The code snippet must be written entirely in english. This is because non-unicode characters are not supported by *PyScent*.

4) The code snippet must not contain any syntax errors. This is because *PyScent* is not able to detect code smells in code that does not compile. To remove the snippets with syntax errors, a program was written to delete export each code snippet to a `.py` file and try to compile it. If a `SyntaxError` or `TypeError` was thrown by Python, the file was assumed to be erronous and deleted from the dataset.

D. Code Smell Detection

The last step was to detect code smells in the code snippets. To do so, the *PyScent* tool was used. *PyScent* is a Python tool that detects code smells in Python code and the source code can be found on Github [2].

PyScent detects the 11 following code smells:

- **Long Method:** A method with over 50 statements.
- **Long Parameter List:** A method that takes more than 5 arguments.
- **Long Branches:** An if statement with more than 12 branches.
- **Too Many Attributes:** A class that has more than 7 attributes.
- **Too Many Methods:** A class that has more than 20 public methods.
- **Useless Try/Except Clauses:** A try/except clause that catches too many exception (using the `Exception` or `Standard Error`), or whose catch clauses are empty.
- **Shotgun Surgery:** A Class that calls more than 5 methods from other classes.
- **Class Cohesion:** A class that has cohesion score lower 30%. The cohesion score is computed using the Cohesion project on Gitbut [3].
- **Code Complexity:** A block that has a cyclomatic complexity score lower than 'C'. The cyclomatic complexity score is computed using the Radon project [4].
- **Long Lambda:** A lambda function that has more than 60 characters.
- **Long List Comprehension:** A list comprehension that has more than 72 characters.

To analyse the code snippets, *PyScent* needs them to be in `.py` files under one directory. Therefore, a program was written to export all of the code snippets to `.py` files and run *PyScent* on them.

The results of the code smell detection are presented in section IV.

IV. Results

In this section, we first present the result of the code smell analysis performed by Pyscent in section IV-A, then, we compare the results between ChatGPT and Stack Overflow in IV-B.

A. Code Smell Results

Following the code snippet analysis by Pyscent, a report was generated detailling the number of smells detected for each Code Smell described in section III. The report also presents the number of possible occurences of each Code Smell, so for example, there were 1247 methods detected in ChatGPT's code snippets, and 9 of them were detected as Long Parameter.

Furthermore, to be able to compare the results between ChatGPT and Stack Overflow, the percentage of each Code Smell was calculated. This percentage is the number of smells detected divided by the number of possible occurences of the smell.

The results of the analysis are presented in tables I and II.

TABLE I
ChatGPT's Code Smells

Smell Name	Smell Count	# of poss. occ.	%
Long Methods	0	1247	0.000%
Long Parameter	9	1247	0.722%
Long Branches	1	N/A	N/A
Too Many Attribute	11	250	4.400%
Too Many Methods	0	250	0.000%
Useless try/catch	1	101	0.990%
Shotgun Surgery	175	250	70.000%
Low Class Cohesion	0	250	0.000%
Code Complexity (< C)	17	1497	1.136%
Long Lambda	0	43	0.000%
Long List Comprehension	16	59	27.119%

For ChatGPT, we can see that the most common code smell is Shotgun Surgery, with a 70% occurence rate. This is not surprising, since ChatGPT is not aware of the code context of the developer. Therefore, it will often generate code snippets that are not related to the context.

We can also observe that ChatGPT tends to write long list comprehension. This might be because the training data it was fed contained a lot of long list comprehension. However, it does not seem to generate long lambdas, which might mean

[2]https://github.com/whyjay17/Pyscent
[3]https://github.com/mschwager/cohesion
[4]https://pypi.org/project/radon/

that it is aware of this code smell.

Finally, we can see that ChatGPT does not generate long methods, which could mean it is also aware of this code smell, but it could also be because ChatGPT does not tend to write long code snippets in general, so the code smell cannot appear.

TABLE II
STACK OVERFLOW'S CODE SMELLS

Smell Name	Smell Count	# of poss. occ.	%
Long Methods	5	3438	0.145%
Long Parameter	18	3438	0.524%
Long Branches	14	N/A	N/A
Too Many Attribute	4	592	0.676%
Too Many Methods	0	592	0.000%
Useless try/catch	19	204	9.314%
Shotgun Surgery	376	592	63.514%
Low Class Cohesion	14	592	2.365%
Code Complexity (< C)	41	4030	1.017%
Long Lambda	19	146	13.014%
Long List Comprehension	56	439	12.756%

From Table II, we can see that Stack Overflow's code snippets also tend to have a lot of Shotgun Surgery and long list comprehension, but not as much as ChatGPT. This is probably because Stack Overflow's code snippets are written by humans, who are more aware of those Code Smells.

However, Stack Overflow's code snippet present a higher occurence of useless try/catch and long lambdas compared to ChatGPT's. The first one might be because the developer does not put a lot of thought into the type of error handled by the try/catch to save time, and the second one could be because they want to impress the reader with a complex lambda.

Finally, it might be surprising to see that Stack Overflow's code snippets present Code Smells that are not present in any of ChatGPT's, such as long methods, low class cohesion and long lambdas, since we would expect developers to be more aware of Code Smells than ChatGPT.

B. Code Smell Comparison

Table III presents the comparison between ChatGPT and Stack Overflow's code smell percentages where the difference is significative ($> 0.5\%$). The percentage difference is calculated by substracting the percentage of Stack Overflow's code smells from the percentage of ChatGPT's code smells.

Therefore, a positive value means that there are more code smells of this type in Stack Overflow's code snippets, whereas a negative value means that ChatGPT uses this code smell more often.

Finally, an overall percentage difference is computed using the sum of the percentage difference for all of the code smells.

TABLE III
SO VS. CHATGPT CODE SMELLS

Smell Name	SO (%)	ChatGPT %	difference (%)
Too Many Attribute	0.68%	4.40%	-3.72%
Useless try/catch	9.31%	0.99%	8.32%
Shotgun Surgery	63.51%	70.00%	-6.49%
Low Class Cohesion	2.37%	0%	2.37%
Long Lambda	13.01%	0%	13.01%
Long List Comprehension	12.76%	27.12%	-14.36%
Overall			**-1.04%**

From Table III, we can see that ChatGPT's code snippets present more code smells than Stack Overflow's, with an overall difference of -1.04%. This means that the code quality of ChatGPT is more or less similar to that of Stack Overflow.

V. THREATS TO VALIDITY

VI. THREATS TO VALIDITY

VII. CONCLUSION

REFERENCES

[1] NAIST-SE. "DevGPT: Studying Developer-ChatGPT Conversations". MSR-2024 Challenge. https://github.com/NAIST-SE/DevGPT