Comments in GitHub Projects

Josef Gugglberger & Till Volkmer

Seminar Paper

Supervisor:
Prof. Dr. Michael Felderer
Department of Computer Science
Universität Innsbruck

Innsbruck, January 20, 2019

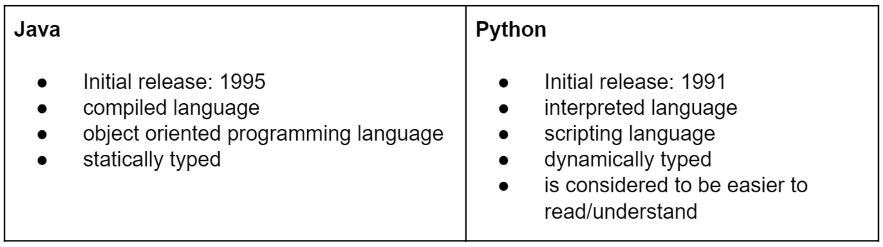
1 Abstract

Comments in source code help to comprehend the function and intention of software projects with respect to development and maintenance. Transparency is important, especially in open source projects, which might not have comprehensive written documentation. Comments in-code enable an accurate contribution to open source projects. Developers can share relevant and important information directly with leaving accurate and sufficient comments.

This research assignment focuses on the differences of in-code comments between Java and Python projects and should reveal differences in the amount, the types and the quality of comments. From these three main measurement points we investigate for a correlation between the characteristics of commenting and the influence on the popularity and of the respective coding projects.

2 Introduction

Java and Python are two of the most popular programming languages and share some key properties. Despite the earlier introduction of Python, Java was for many years more popular than Python. Just recently Python took the lead, as it is considered to be a very productive language compared to compiled programming languages like Java. With the general understanding that Python usually requires fewer lines of code to reach the same output and is easier to oversee and to comprehend.



Java	Python
• Initial release: 1995 • compiled language • object oriented programming language • statically typed	• Initial release: 1991 • interpreted language • scripting language • dynamically typed • is considered to be easier to read/understand

Table 0.1: Java - Python comparison

GitHub [1] provided this research with the input data, by offering a platform to enable collaboration on open source projects. Popular projects have several thousand contributors, to achieve traceability of software projects, comments have a key role. For this research, the selection of projects was made depending on the popularity, which was measured by the number of stars the project has. Overall we analyzed 175 projects with a total of 21,8 million lines (code and comments) and which contain 3,9 million lines of comments. On average the projects had a size of 125.000 lines of code and comments.

3 Background and related work

The study from Daniela Steidl et al. in [2] investigates the usefulness of comments for system understanding and searches for possibilities to include comments into quality assessment of the software. This study sets a good basis and sets up a useful comment classification scheme. Our research project is limited to Python and Java and open source projects on the platform GitHub. Our project searches for differences in the commenting behavior between the two programming languages. With the assumption that the fundamental differences in syntax, background and favored field of application of Java and Python have an impact on the commenting. Therefore we compare the quantity of comments, the correlation between the popularity of the projects and the amount of comments in the comments.

Guzman et al. showed an empirical study of git commit messages in [3]. The authors analyzed the effect of different factors like day-time and the used programming language on the sentiment of commit messages. This approaches gave us the motivation to examine the sentiment in source code comments.

4 Research method

To following section aims to present our research methodology. First, we defined the research question in sub-section 4.1. Next, sub-section 4.2, describes how we collected the data that we analyze in sub-section 4.3. Finally sub-section 5 presents and discusses the gathered results.

4.1 Research questions

Research question 1

Is there a correlation between the quantity of comments in source code and the language used?

Metrics: lo comments / lo code. The comment to code ratio is compared between Java and Python.

H0: The amount of comments is not affected by the respective programming language.

- H0 : avg(lo comment/code Python) = avg(lo comment/code Java)
- Ha : avg(lo comment/code Python) < or > avg (lo comment/code Java)

Research question 2

Is there a correlation between the popularity on GitHub and the amount of comments?

More popular projects should have more extensive commenting, thus more people are contributing to the project.

Metrics: We search for a correlation between the number of stars for a project and the amount of comments. With the hypothesis, that a project which is more popular i.e. has more stars also has a more detailed commenting which benefits the contribution to the project

H0: Popularity has no impact on the amount of comments.

- H0 : correlation coefficient (stars, lo comments) = 0
- Ha : correlation coefficient (stars, lo comments) < or > 0

Research question 3

Is there a correlation between the sentiment in comments in source code and the language used?

Metric: Sentiment compound score provided by VADER [4]. This is an absolute value between -1 (very bad sentiment) and +1 (very good sentiment).

H0: Java is more often used by corporate developers, thus the language used in source comments is more serious.

4.2 Data collection procedure

The data extraction was implemented with Python. We decided to collect data from 175 open source repositories, whereas 77 projects have Java as their dominant language and the other 98 projects have Python as their main language. First, we cloned the repositories from GitHub to our local machine. A recursive algorithm traverses the directory structure in each repository and initiates the data extraction on each file ending with *.java* or *.py*. The extraction itself is done by the help of regular expressions. After a comment is located, it is sorted into one of the predefined categories listed in subsection 4.3. This classification is also done with regular expressions and simple string matching techniques. It should be noted that the matching technique with regular expressions here is just a simplification. Regular expressions are in general not expressive enough to analyze source code, for exact results a parser should be used. The result of this stage is saved to a CSV file, which is the basis for further analysis, described in the next section.

Java source code comments can either be defined with a double slash (single-line comment), or with a slash followed by a star which is closed in the reverse order (multi-line comment). For an example see listing 1.

Listing 2 shows two different comment styles in Python. Single-line comments are denoted by a single hashtag symbol. Multi-line comments inside triple double quotes are so called docstrings. Docstrings can occur as the first statement in a module, function, class, or method definition.

Listing 1: Commented Java Code

```
/*
    This is a Java multi-line comment
*/
public int foobar(){
    return 42; // this is a Java single line comment
}
```

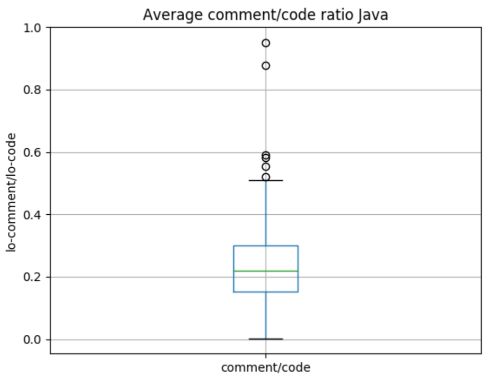
Listing 2: Commented Python Code

```
def foobar:
    """ this is a Python multi-line comment """
    return 42; # this is a Python single line comment
```

4.3 Data analysis procedure

Within the extraction procedure, each comment is sorted into one of five predefined categories. This enables not just to look for differences in code to comment quantities, but also analyze what types and their respective share of comments are within the analyzed projects. Therefore we created following five categories:

- **Header comments**. Give an overview of the content and possible copyright claims. Headers are typically at the top of a file, and can be identified by the position
- **Method comments**. Describe the functionality of a method
- **Class comments**. Describe the functionality of a class
- **To-Do comments**. Comment which contains a remaining ToDo or a remark
- **Other comments**. Comment which does not fall into any category above.



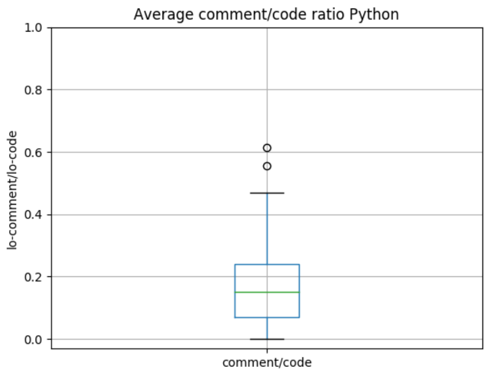


Figure 0.1: Average ratio of lo-comment/lo-code in analyzed projects.

The differences in the shares of comment categories, as they can be seen in figure 0.4, point to structural differences in the commenting behavior between Java an Python projects. Java has a comparatively higher share of header and class comments, whereby Python a higher share of inline comments. A higher share of header comments can point to a more detailed description of the intention of the particular file, the higher share of class comments leaves the implication, that classes written in Java more often need to be specified. More inline comments lead to the conclusion that more information with implementation decisions are required.

Figure 0.1 shows that the median of the ratio between lines of comments and lines of code is is higher in Java than in Python. Java has a ratio between lines of comment and lines of code of about 0.22, whereas Python has a ratio of about 0.15. In the analyzed Java projects there are more outliers, there are even two projects with a ratio over 0.8.

Figure 0.2 shows the box-plot of the average length of source code comments. The median of the average lengths is higher in Java, with about 150 characters per comment, in comparison to 82 characters per comment in Python.

4.4 Validity

Concerning the validity of our experiment, there are two points that should be noted: first, the sample size is relatively small and, second, we only examined the most popular projects. This selection may lead to results that are not representative because there is less diversity between the chosen projects as there would be with a random selection. Furthermore, there are many tutorial and example projects among the most popular repositories. The commenting behaviour in such projects may vary from real-life projects.

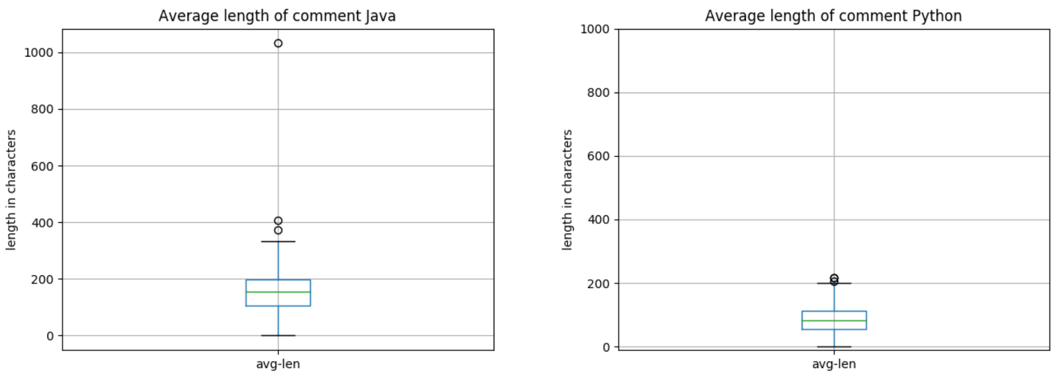


Figure 0.2: Average length a comment in analyzed projects.

5 Research results and discussion

To draw conclusions for this project, we had to look in-depth into the differences between Java and Python. First from a technical point of view, moreover to draw conclusions it was important to consider the behavioural differences of developers of the respective languages. The technical differences which affect the syntax and general rules of the languages affect the commenting behaviour, such

5.1 Research question 1

RQ1 was about the pure quantity differences, with creating bar charts to visualize the results and the calculation of overall average scores was used to reject the null hypothesis. The result showed a rounded difference of 10% in the comment to code ratio. On average a Java project had 25,7% lines of comments and a Python project 15,4%. With this result, H0 is rejected as there is a distinct difference in the amount of comments. Also the average length of a comment is larger, following box-plots indicate a clear difference. This let us draw the conclusion that the average comment is also more complex and requires more characters to explain the statement.

	lo code	lo comments	lo comments/ lo code
Java	11177942	2875282	0,257228
Python	6770754	1039213	0,153486

Figure 0.3: Lines of comment to lines of code ratio

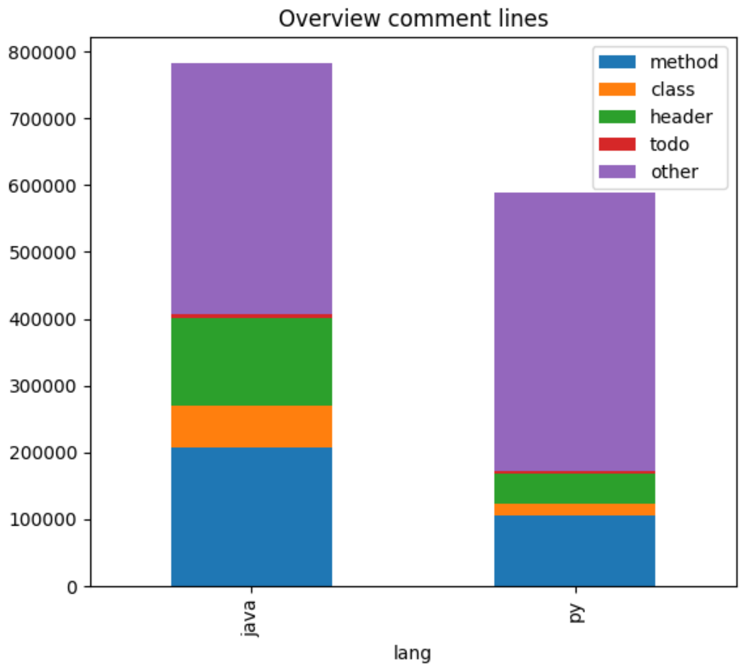


Figure 0.4: Comment category composition in analyzed projects.

5.2 Research question 2

This led us RQ2 let us look at the popularity of the projects on GitHub, which is measured with the help of the amount of stars the respective project has. With the initial thought, a well documented project with a high share of comments should be starred more often, as it is well made. The null hypothesis could be rejected by a very small correlation coefficient of 0,057, thus there is no distinct correlation of stars and the extent of the commenting.

5.3 Research question 3

RQ3 was about analyzing the sentiment in source code comments. We used the VADER [4] sentiment analysis package to retrieve a score of sentiment between -1 and 1. After hand-checking the results we can say that it does not make sense to do sentiment analysis of source code comments, because they are written in a very describing and objective fashion.

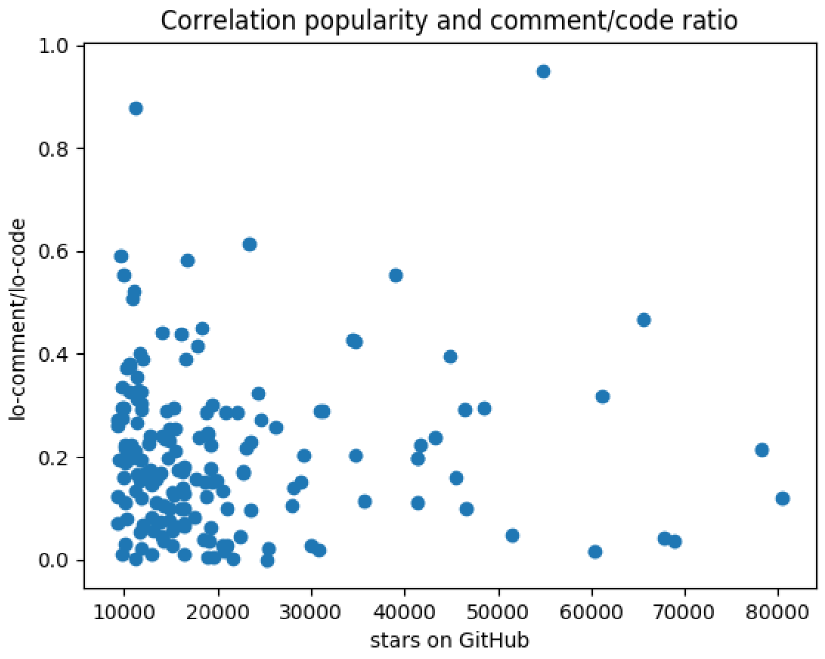


Figure 0.5: Correlation between stars and comments

6 Personal reflection on the project

The project gave us a deeper understanding of how comments in source code work, and how they are used differently among programming languages and also among projects of the same language. We also gained information on how to conduct a data mining experiment in practice, and how to deal with GitHub API. We also touched the field of sentiment analysis, which was not appropriate for our study, but nevertheless, it seemed new and interesting to us.

Along with all those topics we discovered Python libraries that were new to us: for example GitPython [5] and VADER [4]. However, in the presentations last week we saw that we missed using some packages that would have been beneficial for our project, for example, PyGithub [6], a wrapper to deal with the GitHub API, and some library to take care of the LOC count and similar measurements. We implemented some of those on our own instead.

As a takeaway and for future research on this topic, we should obtain and define the domain of the included projects, as the commenting behavior may be influenced by the purpose of the software. Also the maturity of the project, as older projects may be better elaborated and also who is maintaining the projects might influence the commenting in general.

Bibliography

[1] GitHub. https://github.com. Accessed: 2020-01-19.

[2] Daniela Steidl, Benjamin Hummel, and Elmar Juergens. Quality analysis of source code comments. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 83–92. Ieee, 2013.

[3] Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 352–355. ACM, 2014.

[4] VADER Sentiment. https://github.com/cjhutto/vaderSentiment. Accessed: 2020-01-19.

[5] GitPython. https://gitpython.readthedocs.io/en/stable/. Accessed: 2020-01-19.

[6] PyGitHub. https://github.com/PyGithub/PyGithub. Accessed: 2020-01-19.