Automatically assessing quality of class comments

Ludovic Herbelin, Seminar Software Composition, MCS 2020

Supervisor : Pooja Rani

Introduction and motivation

Comments overview

- Comments are one of the main sources of documentation of a project
- They should help contribute to the code’s understandability
- Documented code has been proven to be easier to understand than undocumented ones (D. Steidl, 2013)
- Problem: Documentation is often given a lower priority

Comment quality

What makes a good comment ?

What makes a good comment ?

```
// Async edge case #6566 requires saving the timestamp when event listeners are
// attached. However, calling performance.now() has a perf overhead especially
// if the page has thousands of event listeners. Instead, we take a timestamp
// every time the scheduler flushes and use that for all event listeners
// attached during that flush.
// Async edge case fix requires storing an event listener's attach timestamp.
export let currentFlushTimestamp = 0
```

Too long !

```
// run the thread
new Thread(runnable).start()
```

Trivial !

```
// if we had a previous association
// restore and throw an exception
if(previous != null)
    taskVertices.put(id, previous)
```

Okay

Automatically analyzing comment quality

- Comment’s usefulness is related to the code’s understandability
- Need to assess and relate the natural language of the comment and machine language of the code

Our work

Goal : analyze quality of source code comments

Focus on related metrics

Applied on Pharo and Python datasets

Quality attributes of a comment

- Coherence : How the code relates to the comment
- Completeness : Are there enough comments, is everything documented ?
- Natural Language Quality

Work pipeline

Discarded Metrics

- SYNC Heuristics / Documentable Item ratio
- Polysemy Heuristics
- API External Documentation Quality

Previous work

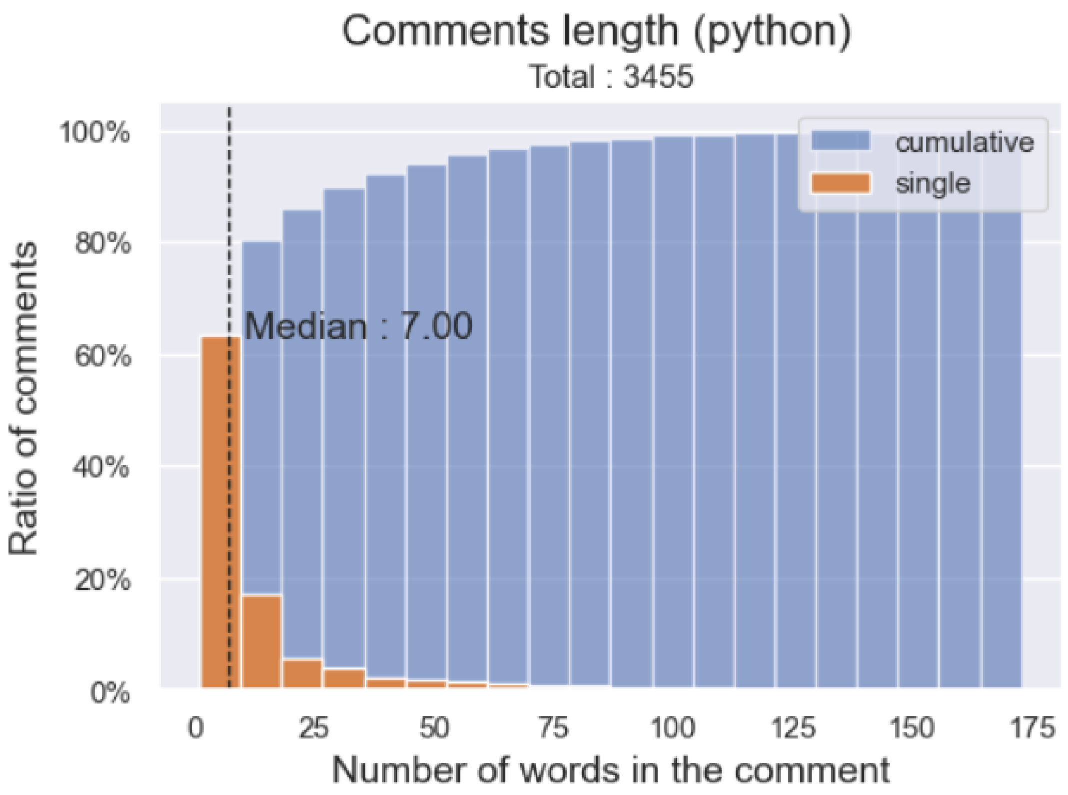
1. Automatic Quality Assessment of Source Code Comments: The JavadocMiner (N.Khamis, 2010)
2. Quality analysis of source code comments (D. Steidl, 2013)
3. Automatically Assessing Code Understandability (S.Scalabrino, 2017)

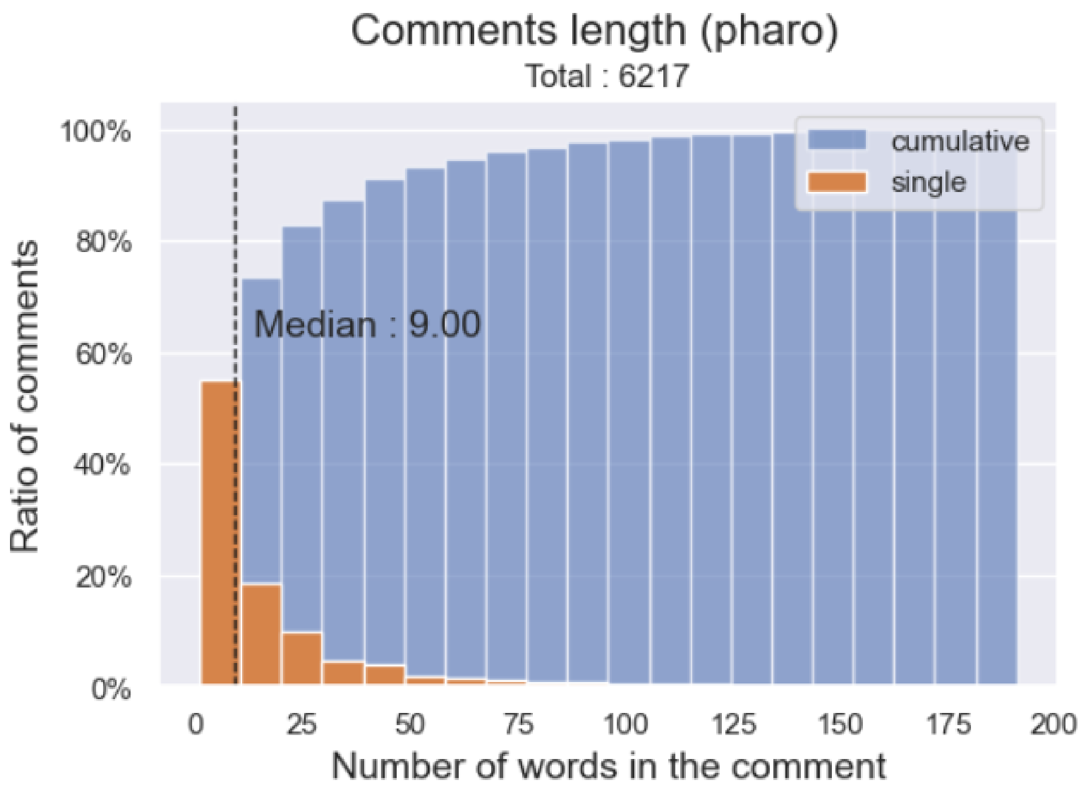
Metric: Comment completeness

Number of words in a class comment

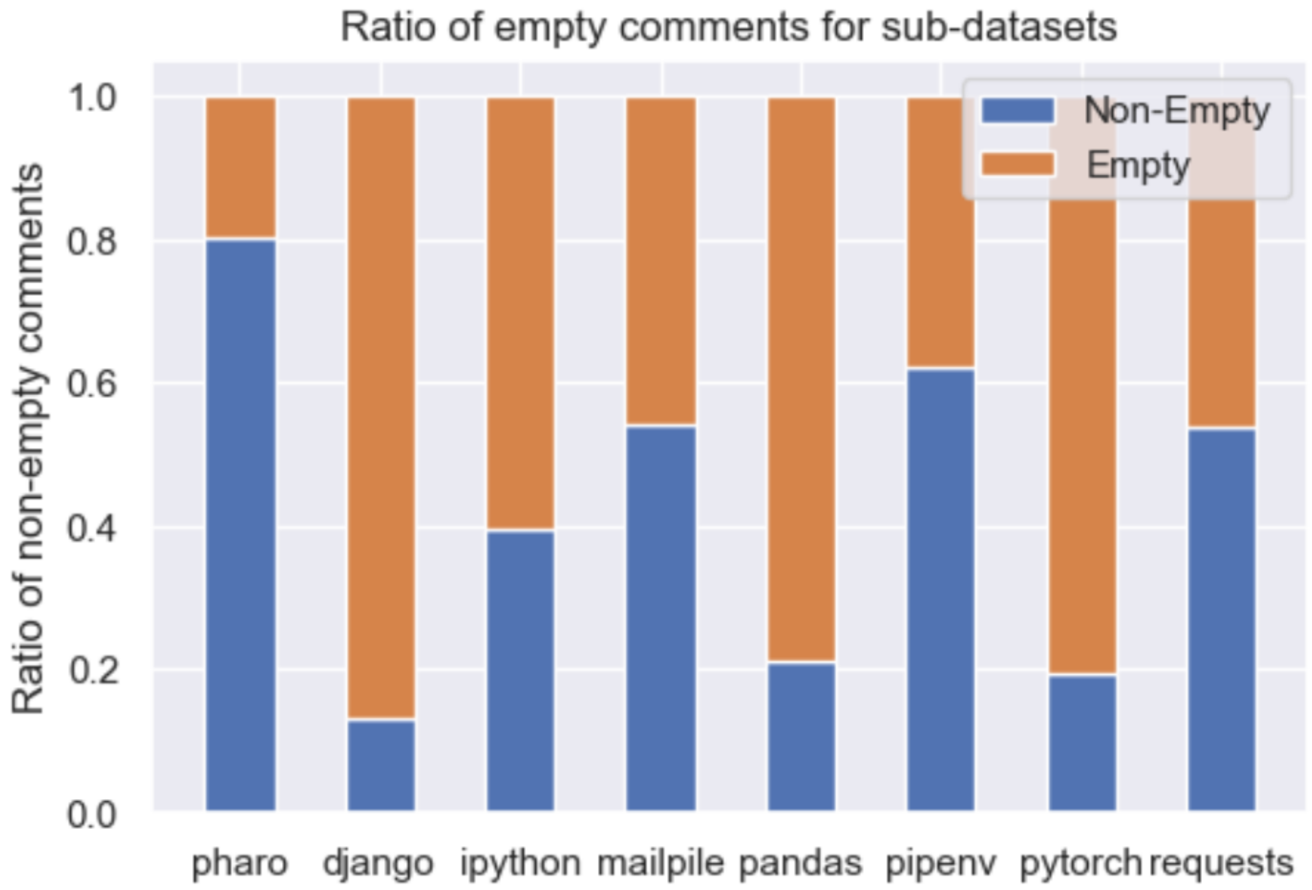
Comment should **at least contain 3 words** to be considered useful (D. Steidl, 2013)

Metric: Comment completeness





Ratio of empty comments



Insight

Python class comments tend to be shorter than Pharo ones

Python datasets have a higher ratio of empty class comments : **80% vs 20%** for Pharo

Metric: Coherence Coefficient

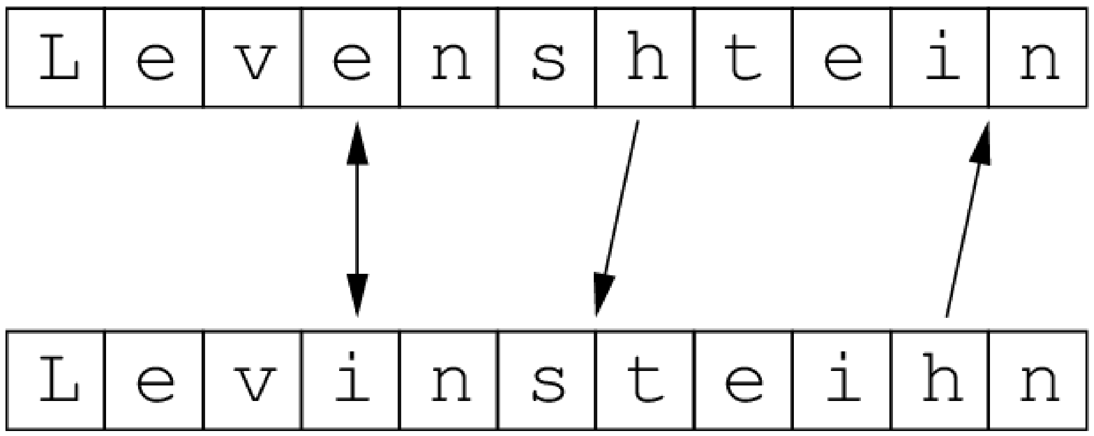
- Goal : compute how close the class name is to the comment, using edit distance
- Ratio of similar words to total words
- High coherence thresholds empirically defined : **0.5** and **0.75**
- Case when **Coeff = 0**

Class : *GLMReplacePresentationsStrategy*

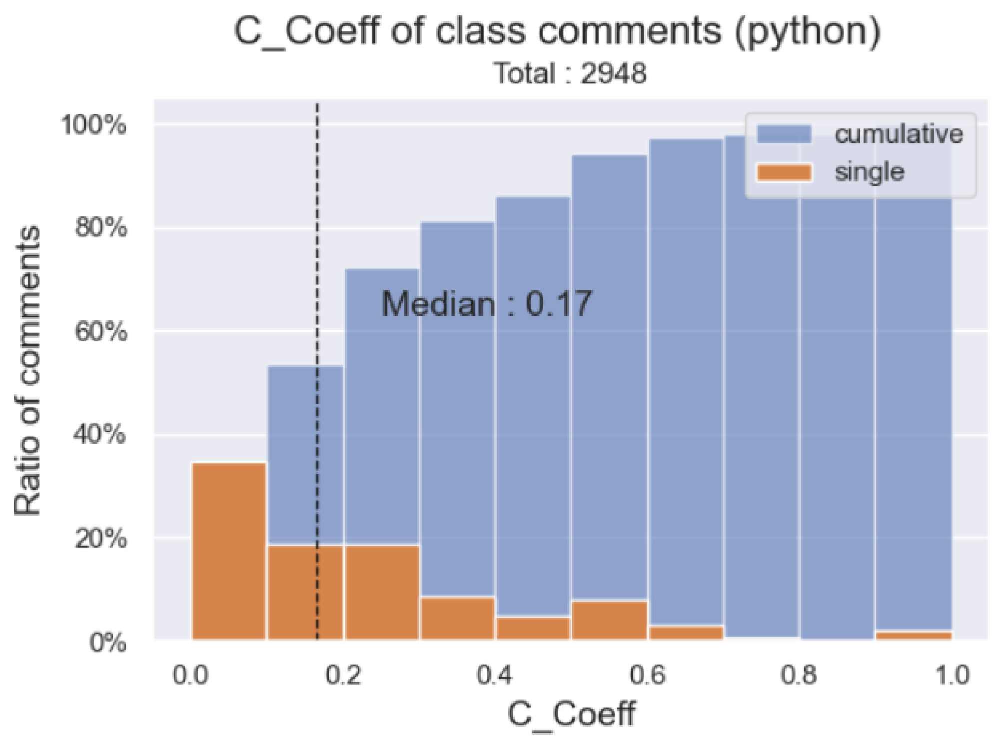
This strategy replaces the presentations from the pane of the destination port.

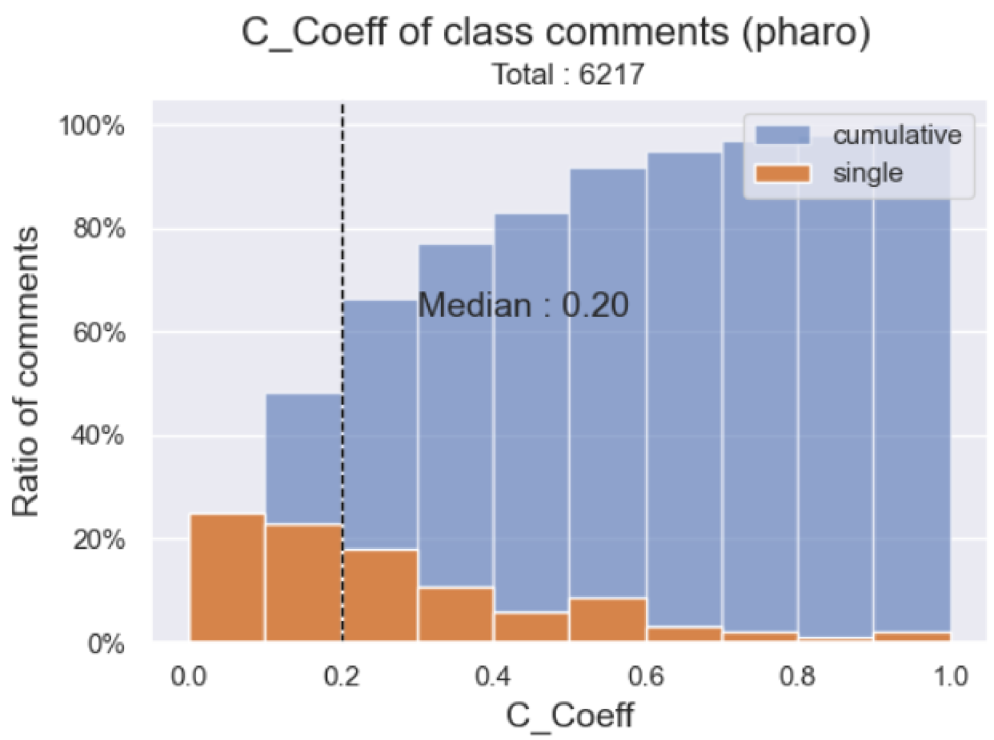
Edit distance

- Number of operations required to get from a string to another
- Usually costs for delete or insert is **1**, substitute is **2**
- Example cost : **2 + 1 + 1 = 4**



Coherence coefficient results





Coherence coefficient examples

Class : *ImageFieldTwoDimensionsTests*, **c_coeff = 0.60**

* Tests behavior of an ImageField and its dimensions fields. *

Class : *AdminViewProxyModelPermissionsTests*, **c_coeff = 1.00**

* Tests for proxy models permissions in the admin. *

Class : *GLMReplacePresentationsStrategy*, **c_coeff = 0.50**

This strategy replaces the presentations from the pane of the destination port.

Class : *ClyMethodContextOfFullBrowser*, **c_coeff = 0.80**

I am a context of selected methods in full browser

Insight

~80% of the comments are between **0.0** and **0.5**

Comments are close to the class name but not too much

Ratios

Number of **non-empty** comments : **2948**

Number of comments with **c_coeff = 0** (completely dissimilar) : **639 (21.68)** %

Number of comments with **c_coeff >= 0.5** (quite similar) : **406 (13.77)** %

Number of comments with **c_coeff >= 0.75** (really similar) : **83 (2.82)** %

Number of **non-empty** comments : **6217**

Number of comments with **c_coeff = 0** (completely dissimilar) : **662 (10.65)** %

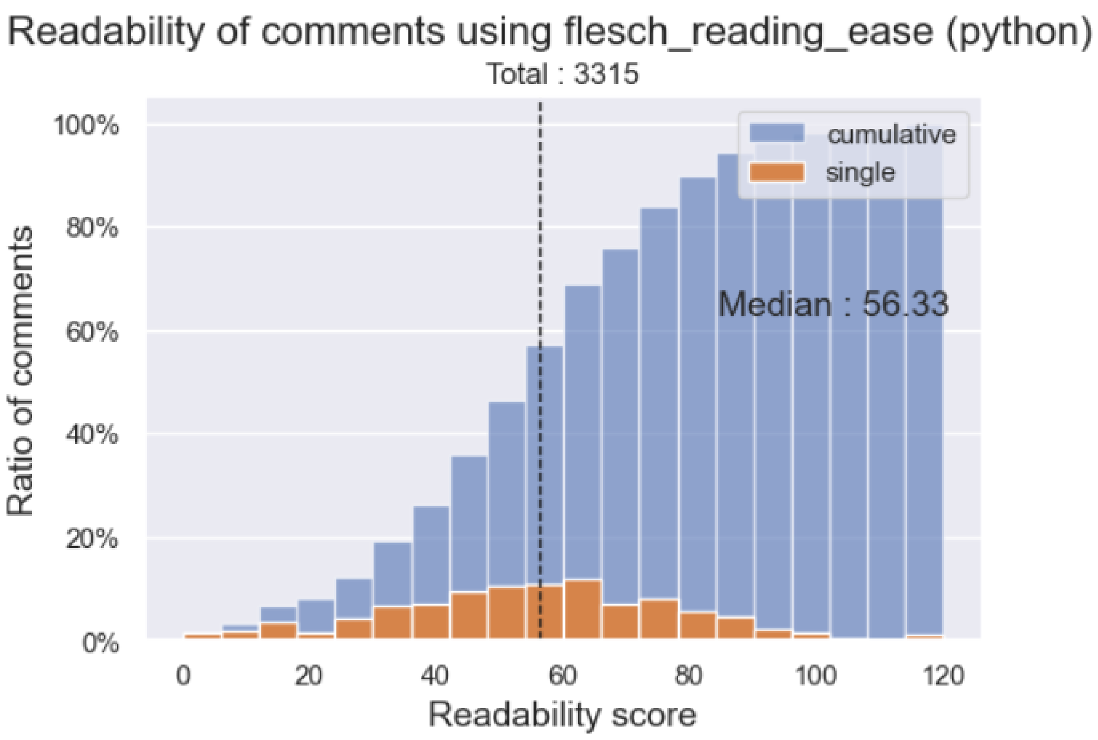
Number of comments with **c_coeff >= 0.5** (quite similar) : **1051 (16.91)** %

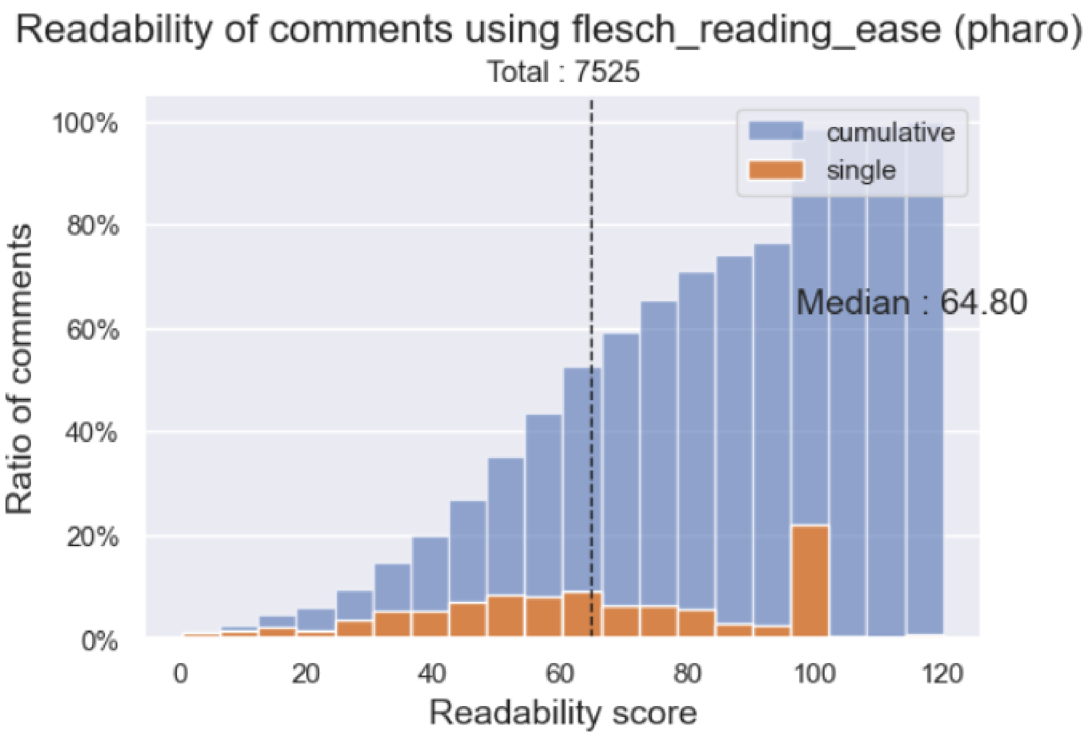
Number of comments with **c_coeff >= 0.75** (really similar) : **312 (5.02)** %

Metric: Readability

- Flesch reading ease **[0-120]** : lower score means harder to read
  - **0-30** : Understood by university graduates
  - **60-70** : 13-15 years old
- A score too high could mean the comment is oversimplified !

Flesch reading ease





Examples

MetacelloScriptEngine runs the execution of the script for one projectSpec -> **42.61**

I contain a fixed number of Slots. Instances of classes using this kind of layout have always the same size.* -> **80.40**

Insight

Comments are mostly quite easy to read

Could have less, more impactful (technical) comments

Insight

- Python dataset is sparser in the class comments compared to Pharo
- Overall similar distributions
- Most comments are close but not too close to the class name
- Improvements can be done towards the technicality

Summary and future work

Analyzing source code and comments is a difficult task

Integrate as a plugin in IDEs

Write meaningful comments

Plan the documentation part in the project tasks

Thank you for your attention

Bibliography

1. N. Khamis, R. Witte, and J. Rilling, “Automatic Quality Assessment of Source Code Comments: the JavadocMiner”
2. S. Scalabino, G. Bavota, et. Al., “Automatically Assessing Code Understandability: How Far Are We?”
3. J. Arthur, K. Stevens, “Assessing the Adequacy of Documentation Through Document Quality Indicators”
4. D. Steidl, B. Hummel, et. Al., “Quality Analysis of Source Code Comments”
5. Y. Shinyama, Y. Arahori, K. Gondow, “Analyzing Code Comments to Boost Program Comprehension”