Directing Generative AI for Pharo Documentation

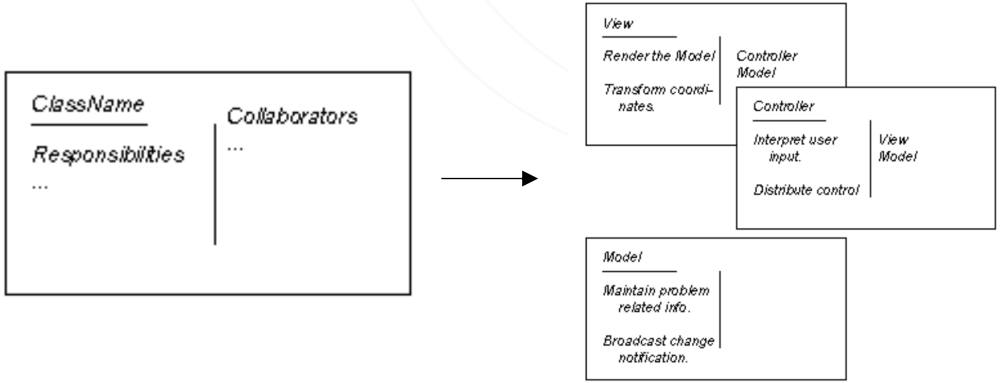
How can we effectively use AI to help us write documentation?

Pascal Zaragoza
Nicolas Hlad

Sommaire

Context

Context : Documentation in Pharo 12



Why documentation matters

- ~58% of a developer's time is spent on code comprehension [1].
  - Bad documentation = more time lost
  - Good documentation = less time lost

Code documentation in Pharo

- Package, class and method-level comments
- Class-Responsibility-Collaborator definition

Class: WeakValueDictionary

I am a dictionary holding only weakly on my values. This mean that as long as my values are referenced (via a strong reference) by other objects, they will stay but in case no object is referencing them during a garbage collection, then my value will vanish and I will return nil instead.
Clients may expect to get a nil value for any object they request since they can be garbaged collected.

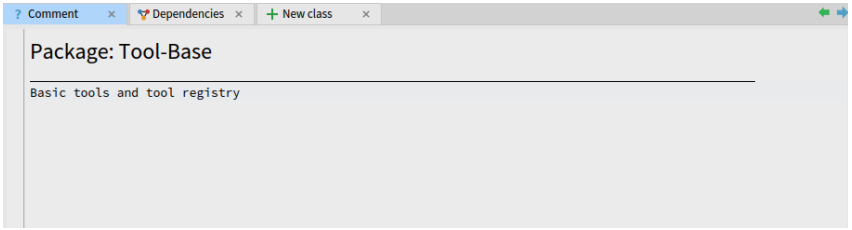
Implementation details:

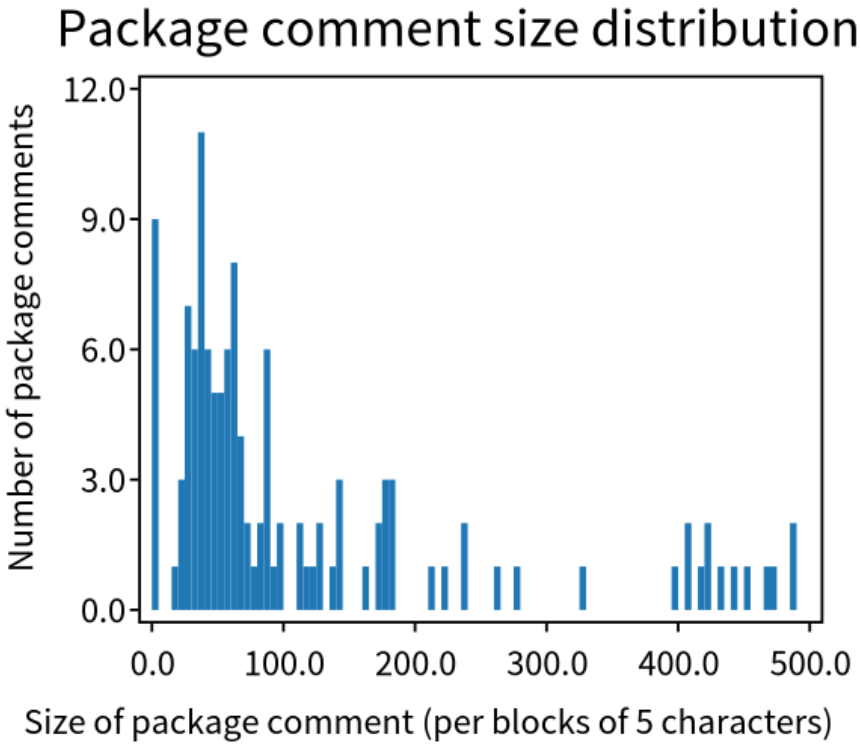
To store keys and values I am using a WeakValueAssociation. This association has a key and a value.
The key is the key the user is giving me, but if the user gives me a nil as value, I wrap it into a CollectionElement. This is explained because I need to do a distinction between nil values given by the user and nil values created by the garbage collection.
When the value of a WeakValueAssociation is a collection element wrapper on nil, then it means the user directly gave us a nil. In case the value of the WeakValueAssociation is nil, it means that we originally had a value that was garbaged collected.

Problem

Package documentation

- Only **16.7% of packages** have comments.
- 81.1% of classes have comments.
- 41.9% of methods have comments.
- Most package comments are very short (60.3% < 100 characters).





Conclusion: There is a strong need for improved and scalable documentation practices in Pharo.

Our approach towards generating package comments

Overview of the Comment Generation Approach

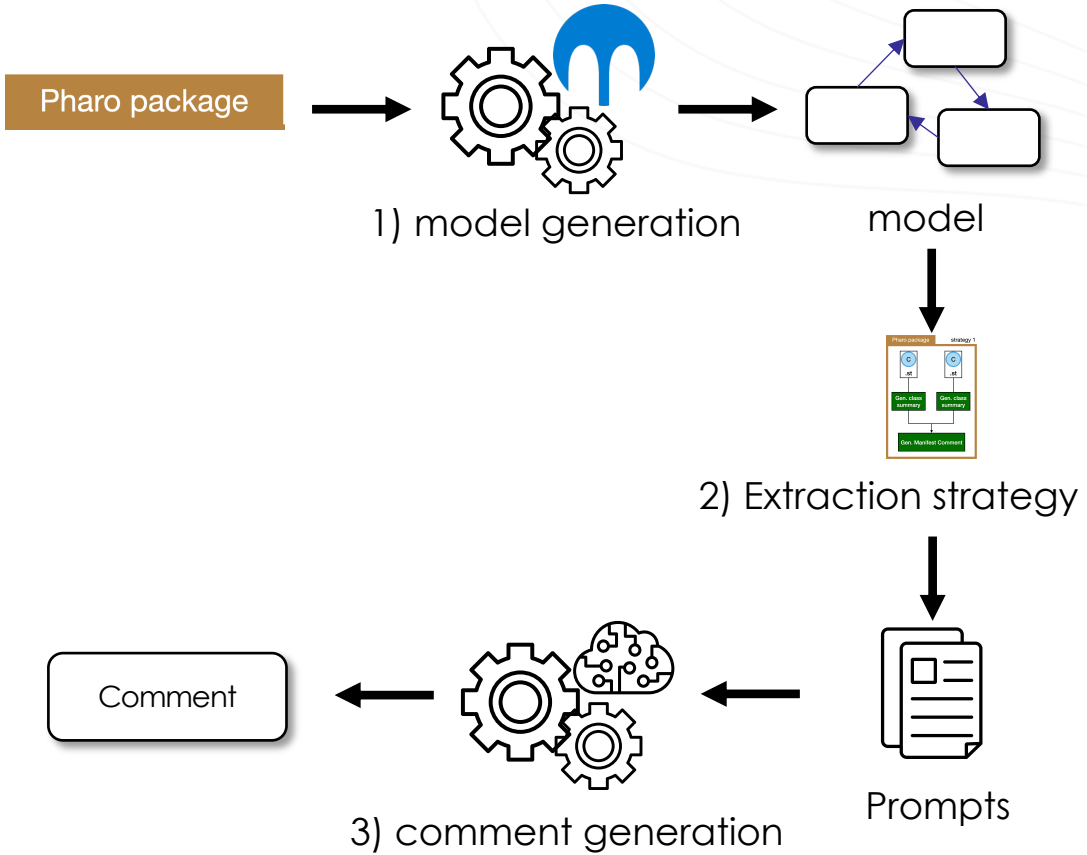
Goal: Improve Pharo package documentation using LLMs.

Method: Retrieval-Augmented Generation (RAG).

Focus: Evaluate how different information sources affect generated comment quality.

3-Step Process:

- Generate a model representation of the package (using Moose).
- Data extraction/retrieval from the model.
- Comment generation via LLM.
  (mistral-small-2503)



https://github.com/pzaragoza93/AutoCodeDocumentator

Example Prompts

```
packageCommentSystemPrompt
   "System prompt used for creating a summary for a class "

   ^ 'You are a useful assistant tasked with writing a comment of what a Pharo package does.
   Pharo is a SmallTalk-based programming language.
   You will be given summaries of a set of class from this package.
   You must generate a general comment for what the package does based on the summaries of its classes.

# Template for writing a comment
Here is the the following template inspired by Class Responsibility Collaborator (CRC) design:

For the Package part:  State a one line summary. For example, "I represent a paragraph of text".

For the Responsibility part: Three sentences about my main responsibilities - what I do, what I know.

For the Collaborators Part: State my main collaborators and one line about how I interact with them. Collaborators are packages and classes that
are external to the package. You should not list the classes within this package. Include the main classes for this package as an entry point.

# How to format your class comment
Use microdown for the creation of the comment. Microdown a shortened version of markdown. You dont need to specify that its a microdown format.

If you give an example of code you can highlight this example code using the triple tick annotation for example:
```language=Pharo&caption=Beautiful&anchor=Fig1\n<insert your code>\n```
In this example, we define an <insert you code> in the Pharo language with the caption "Beautiful" and an anchor of Fig1.

In Pharo, Microdown supports hyperlinks to:
- classes e.g., `Point`,
- methods e.g., `Point class`, `Point>>#setX:setY:`, and
- packages e.g., `#''Microdown-Tests''` (for packages).
If you reference a class package or method use the appropriate hyperlink.'
```
```

Example Prompts

```
packageCommentHumanPrompt: aListOfClassSummaries
   "a human prompt for generating a class comment using the class source text as input"

   | prompt |
   prompt := String streamContents: [ :stream |
                stream
                   nextPutAll:
                      'The following is a set of class summaries of the package which are separated by # ---';
                   cr;
                   nextPutAll: '# Begin class summaries';
                   cr.

                aListOfClassSummaries do: [ :eachSummary |
                   stream
                      nextPutAll: eachSummary;
                      cr;
                      nextPutAll: '# ---';
                      cr ].

                stream
                   cr;
                   nextPutAll: '# End class summaries'.
          stream cr; nextPutAll: 'Now write a comment on the package that has these classes while respecting the formating and styling:' ].
   ^ prompt
```

Overview of the Comment Generation Approach

Package: Tool-Base

I represent a package that provides tools and utilities for managing and interacting with the Pharo development environment.

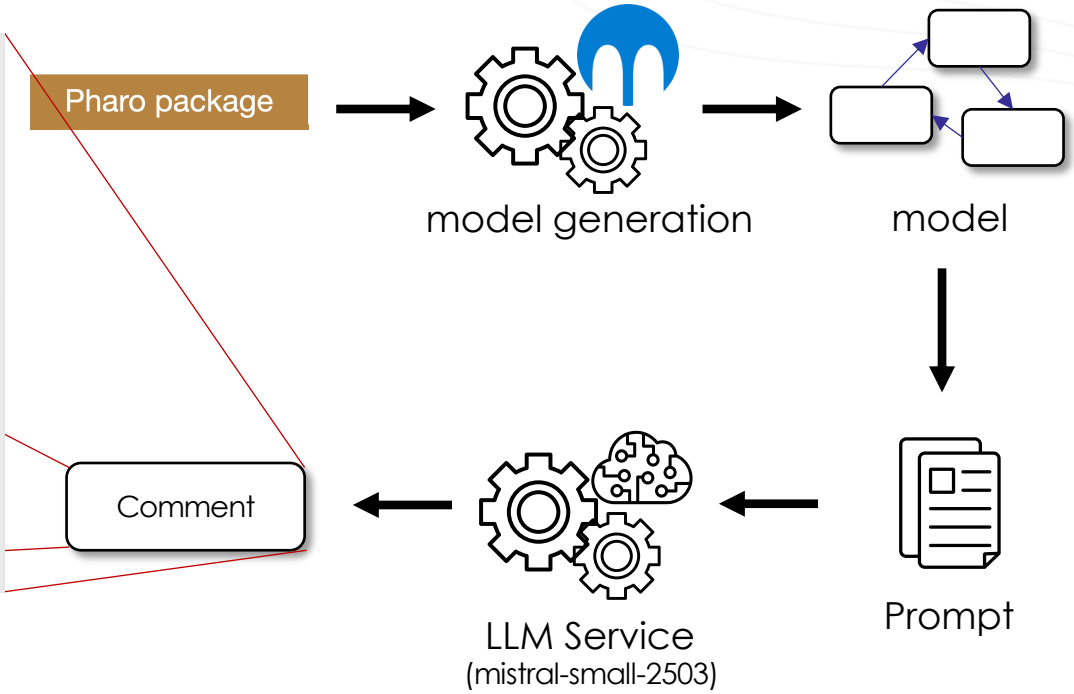
I am responsible for managing keyboard shortcuts, method references, navigation history, and various tools within the Pharo IDE. I provide functionality to define and query shortcuts, convert methods to text, classify methods, and manage navigation history. I also handle user interactions and confirmations for various actions.

I collaborate with the following:

- OSPlatform to determine platform-specific shortcuts.
- KMKeymap to define and manage keyboard shortcuts.
- DateAndTime to handle timestamps for navigation history and method modifications.
- OrderedCollection to store navigation entries.
- SystemNavigation to perform browsing and refactoring actions.
- UIManager to handle user interactions and confirmations.
- RBRefactoryChangeManager to manage refactoring changes.
- Dictionary and Bag to store mappings and collect protocols for method classification.

Key Classes:

- PharoShortcuts: Manages system-wide keyboard shortcuts.
- AbstractMethodReferenceConverter: Abstract wrapper for managing method references.
- SourceMethodConverter: Displays the source code of a provided message.
- NavigationHistory: Manages a collection of navigation entries.
- ToolShortcutsCategory: Defines global keyboard shortcuts for opening various tools.
- TimeStampMessageConverter: Formats and displays the timestamp of a method's last modification.
- AbstractTool: Provides generic methods for managing packages, classes, groups, and methods.
- AbstractMethodConverter: Wrapper for converting methods to text based on a specified filter.
- MethodClassifier: Classifies methods based on their names, prefixes, suffixes, and pragmas.



Strategy 1 – Naive Extraction

Input: Full source code of each class (.st files).

Process:

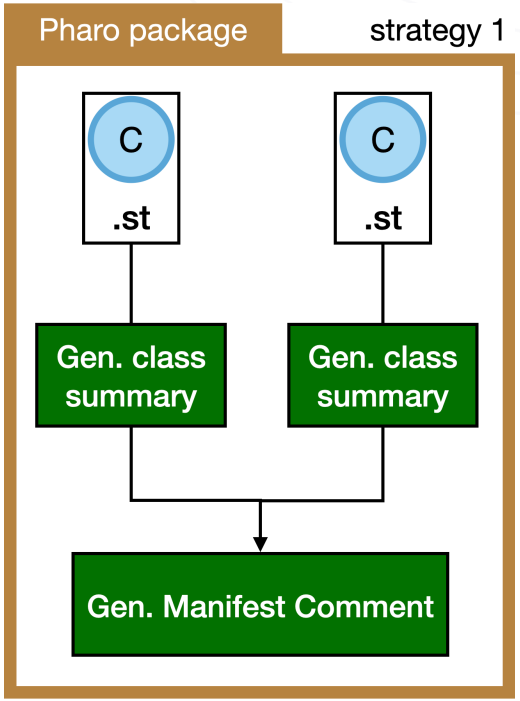
- Summarize class responsibilities, collaborators, and key implementations.
- Use LLM to generate CRC-based package comment from class summaries.

Pros:

- Rich context.
- Can infer detailed responsibilities and interactions.

Cons:

- Risk of **hallucinations** (e.g., non-existent classes).
- **Computationally expensive** due to large context size.



Strategy 2 – Comment-Based Extraction

Input: Existing class comments only.

Process:

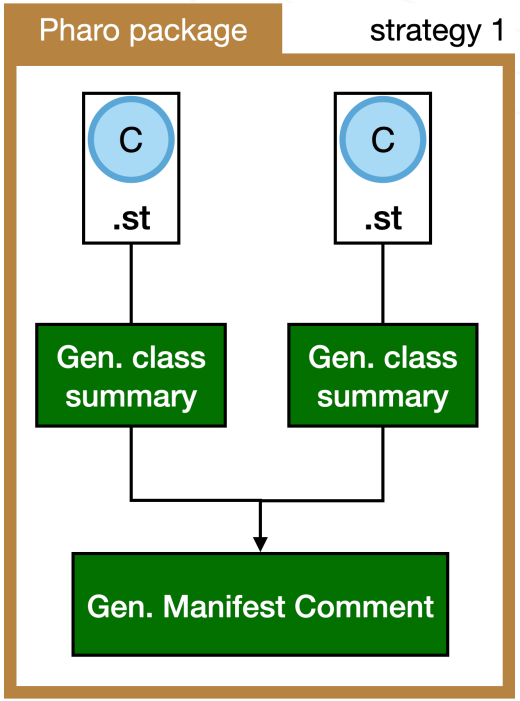
- Aggregate class comments.
- Generate package-level CRC comment using LLM.

Pros:

- Leverages human-authored summaries.
- Lower risk of hallucination.

Cons:

- Limited by **comment coverage** (incomplete/missing comments).
- Misses undocumented class behaviors or dependencies.



Strategy 3 – Comment & Outgoing Reference Extraction

Input: Class comments + method-level outgoing references.

Process:

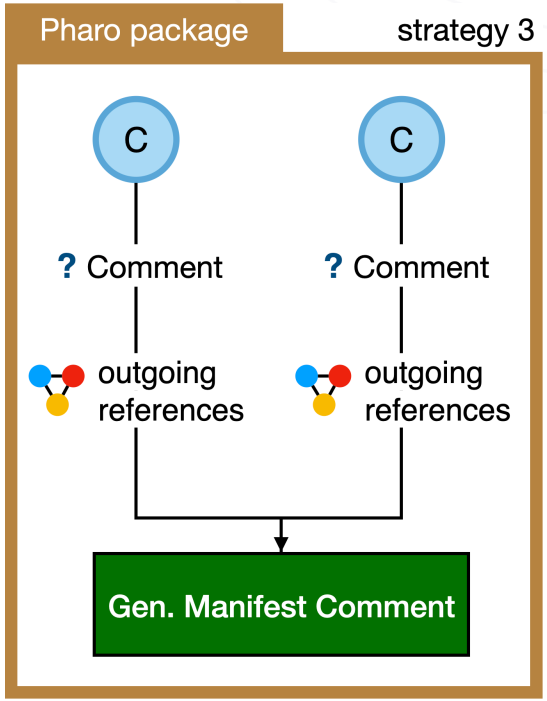
- Extract collaborators through reference analysis.
- Combine with existing class comments for CRC-based comment generation.

Pros:

- Balances authored insights with structural dependency data.
- Better handles **inter-class collaboration context**.

Cons:

- Dependent on reference accuracy and structure parsing.
- Limited by **comment coverage** (incomplete/missing comments).



https://github.com/pzaragoza93/AutoCodeDocumentator

Experimentation

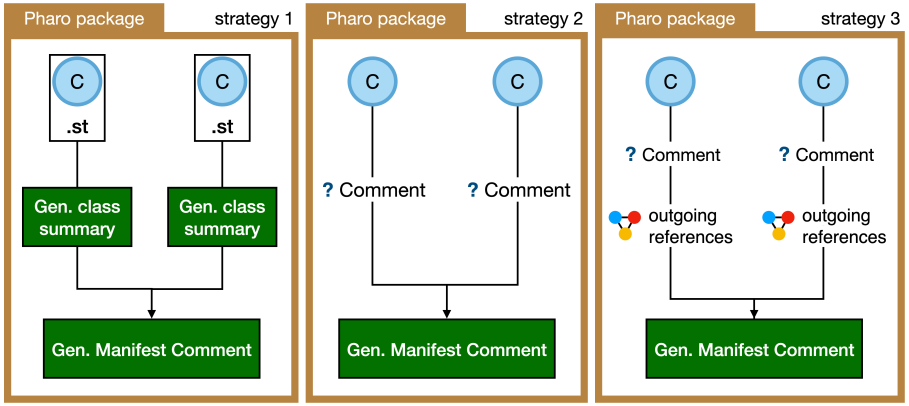
Experimentation

Purpose: Assess the impact of different LLM strategies on package comment generation.

Strategies Tested:

- Naive (source code based)
- Comment-based
- Comment + Dependency-based

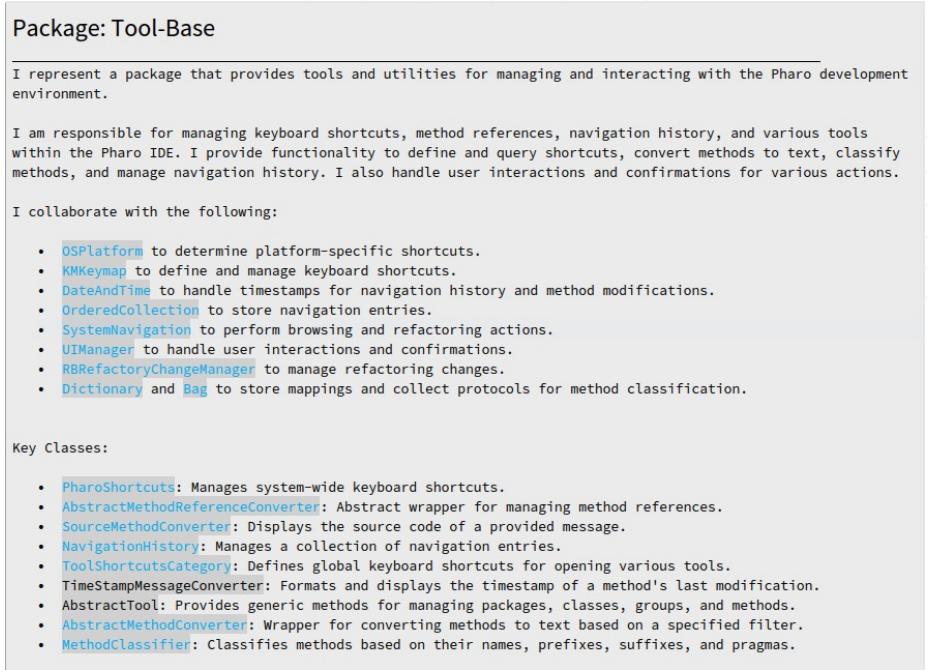
Focus: Identify strengths and weaknesses across strategies.

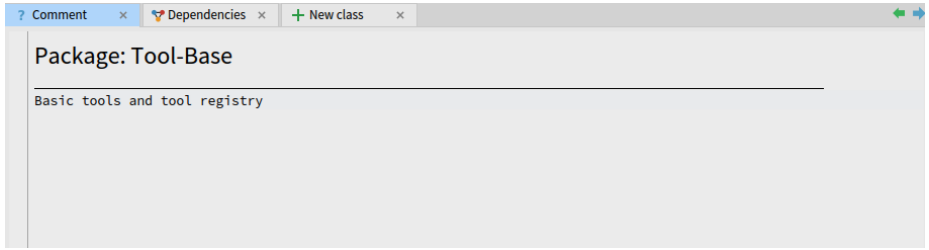


Research questions

- **RQ1**: Impact on CRC structure quality?
- **RQ2**: Accuracy of responsibility descriptions?
- **RQ3**: Accuracy of collaborator descriptions?
- **RQ4**: Overall quality vs. original comments?
- **RQ5**: Effect of package size on comment quality?

VS





Evaluation Dataset

Dataset: 21 Pharo packages

- Grouped by size: Small, Medium, Large (7 each)

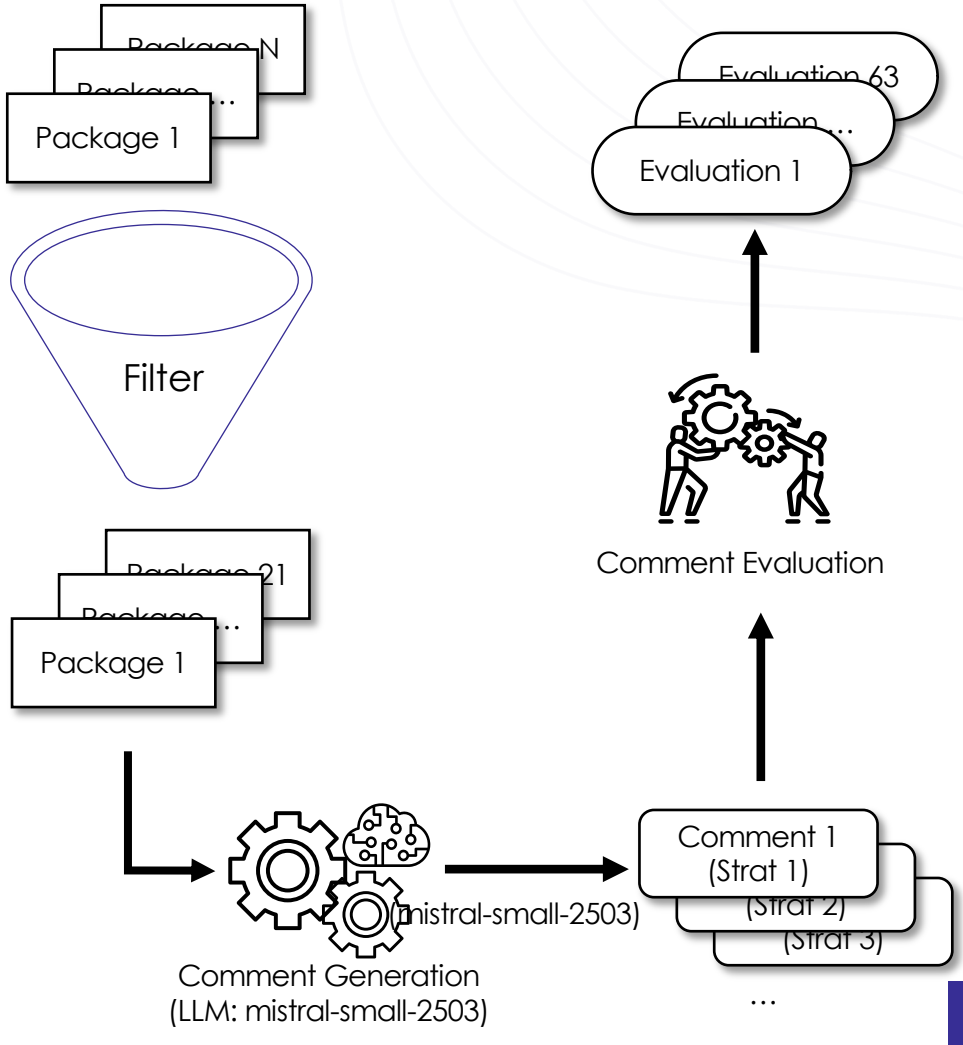
Filtering:

- Only packages with existing comments included.
- Excluded test and baseline packages.

Each package: Evaluated with all 3 strategies → 63 generated comments.

Large Language Model: mistral-small-2503

- Apache 2 Licence



Evaluation Method

Review Process:

- 6 Pharo users in 3 groups.
- Each user reviewed 7 packages and their 3 generated comments (21 comments per group).

Manual Scoring using 12 questions across 4 categories (3 questions for each category):

- CRC Structure (**RQ 1**)
- Responsibility Accuracy (**RQ 2**)
- Collaborator Accuracy (**RQ 3**)
- Comparison to Original (**RQ 4**)

Scale: 7-point Likert (strongly disagree to strongly agree)

Category	Question ID	Questions
CRC Methodology	crc_methodology accuracy q1	The CRC format (Class name, Responsibility, Collaborators) is clearly respected.
CRC Methodology	crc_methodology accuracy q2	The comment clearly explains both what the package does and who it interacts.
CRC Methodology	crc_methodology accuracy q3	The generated comment contains a structured title, description of purpose, and list of external dependencies.
Responsibility	responsibility accuracy q1	The generated responsibility description correctly describes the core responsibilities of the package.
Responsibility	responsibility accuracy q2	The generated description of the package's responsibility is clear.
Responsibility	responsibility accuracy q3	The generated description of the package's responsibility is succinct.
Collaborators	collaborator accuracy q1	The package's main collaborators are mentioned and described accurately.
Collaborators	collaborator accuracy q2	The generated comment DOES NOT omit important external dependencies.
Collaborators	collaborator accuracy q3	The reason for the interactions between its collaborators is clearly explained.
Comparison	comparison accuracy q1	The generated comment more complete the original comment.
Comparison	comparison accuracy q2	The generated comment is more clear than the original comment.
Comparison	comparison accuracy q3	The generated comment is more useful than the original comment.

Table 1: List of questions, their category and question ID used in the questionnaire.

Evaluation Method

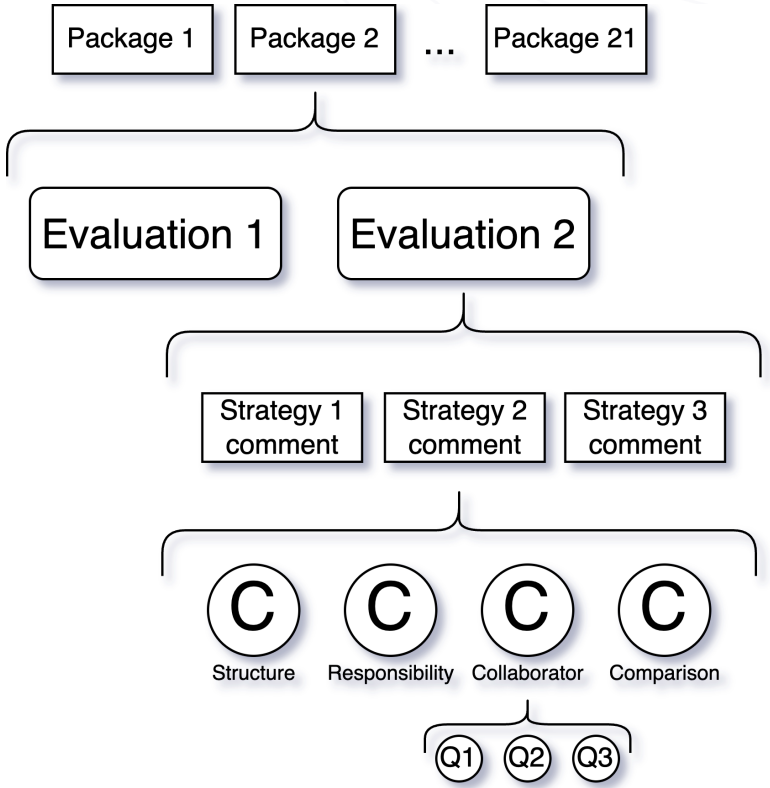
Review Process:

- 6 Pharo users in 3 groups.
- Each user reviewed 7 packages and their 3 generated comments (21 comments per group).

Manual Scoring using 12 questions across 4 categories (3 questions for each category):

- CRC Structure (**RQ 1**)
- Responsibility Accuracy (**RQ 2**)
- Collaborator Accuracy (**RQ 3**)
- Comparison to Original (**RQ 4**)

Scale: 7-point Likert (strongly disagree to strongly agree)



https://github.com/pzaragoza93/label-studio-pharo-evaluation

Results

Results regarding RQ 1 - 4

- Comparison between strategies across the 12 different statements :
  - No strategy offers a significatively better result (RQ 1, 2, 3, 4).
  - All strategies generate comments that are prefered over existing comments

Question	Strategy 1	Strategy 2	Strategy 3	ANOVA (p-value)
crc_methodology_accuracy_q1	6.438	6.188	6.188	0.618
crc_methodology_accuracy_q2	6.062	6.125	5.750	0.574
crc_methodology_accuracy_q3	6.188	6.125	6.062	0.962
responsibility_accuracy_q1	6.125	6.188	6.000	0.850
responsibility_accuracy_q2	6.188	6.188	5.625	0.180
responsibility_accuracy_q3	6.625	6.312	6.375	0.431
collaborator_accuracy_q1	4.500	5.062	4.438	0.614
collaborator_accuracy_q2	3.625	4.062	2.875	0.406
collaborator_accuracy_q3	4.625	5.000	4.000	0.395
comparison_accuracy_q1	6.188	6.625	6.188	0.463
comparison_accuracy_q2	6.375	6.562	6.188	0.414
comparison_accuracy_q3	6.125	6.562	6.250	0.492

Table 2: Average Likert score for each question across all 3 strategies.

Results regarding RQ 5

CRC Methodology	crc_methodology accuracy q2	The comment clearly explains both what the package does and who it interacts.
Collaborators	collaborator accuracy q1	The package's main collaborators are mentioned and described accurately.
Collaborators	collaborator accuracy q2	The generated comment DOES NOT omit important external dependencies.
Comparison	comparison accuracy q3	The generated comment is more useful than the original comment.

Comparison of results between different package sizes (small, medium, large):

- Overall **small packages receive higher scores**
- Small packages have **clearer** comments
- Small**er** packages have collaborators that are **well-mentioned** & we are **not missing key collaborators**.
- Small**er** packages are more useful than existing comments

Question	small	medium	large	ANOVA (p-value)
crc_methodology_accuracy_q1	**6.583**	6.500	6.000	0.107
crc_methodology_accuracy_q2	**6.583**	5.833	5.625	0.040
crc_methodology_accuracy_q3	**6.583**	6.333	5.750	0.167
responsibility_accuracy_q1	**6.250**	5.833	6.042	0.671
responsibility_accuracy_q2	**6.333**	5.667	5.875	0.334
responsibility_accuracy_q3	**6.583**	6.333	6.375	0.696
collaborator_accuracy_q1	**5.833**	4.500	4.333	0.061
collaborator_accuracy_q2	4.750	**5.667**	2.458	0.001
collaborator_accuracy_q3	**6.083**	5.333	3.958	0.004
comparison_accuracy_q1	6.667	**6.833**	6.042	0.166
comparison_accuracy_q2	6.583	**6.833**	6.167	0.086
comparison_accuracy_q3	6.750	**6.833**	6.000	0.048

Table 3: Average Likert score for each question across all 3 strategies.

Conclusion

Conclusion, Limitations, and Future Directions (?)

Limitations

- Limited amount of evaluation per comment
- Needs more work on prompt tuning, document structure
- Weak solution for identifying collaborators

Conclusions:

- Generated comments are more complete, clear and useful than some human-made comments
  → Maybe use when there are no comments ?

Future Directions

- Use heuristics for identifying collaborators & GenAI for describing these collaborations
- Adapt to existing dynamic comment features (e.g. examples)
- Automate a pipeline for comment suggestion in existing Pharo projects

Package: Tool-Base

I represent a package that provides tools and utilities for managing and interacting with the Pharo development environment.

I am responsible for managing keyboard shortcuts, method references, navigation history, and various tools within the Pharo IDE. I provide functionality to define and query shortcuts, convert methods to text, classify methods, and manage navigation history. I also handle user interactions and confirmations for various actions.

I collaborate with the following:

- OSPlatform to determine platform-specific shortcuts.
- KMKeymap to define and manage keyboard shortcuts.
- DateAndTime to handle timestamps for navigation history and method modifications.
- OrderedCollection to store navigation entries.
- SystemNavigation to perform browsing and refactoring actions.
- UIManager to handle user interactions and confirmations.
- RBRefactoryChangeManager to manage refactoring changes.
- Dictionary and Bag to store mappings and collect protocols for method classification.

Key Classes:

- PharoShortcuts: Manages system-wide keyboard shortcuts.
- AbstractMethodReferenceConverter: Abstract wrapper for managing method references.
- SourceMethodConverter: Displays the source code of a provided message.
- NavigationHistory: Manages a collection of navigation entries.
- ToolShortcutsCategory: Defines global keyboard shortcuts for opening various tools.
- TimeStampMessageConverter: Formats and displays the timestamp of a method's last modification.
- AbstractTool: Provides generic methods for managing packages, classes, groups, and methods.
- AbstractMethodConverter: Wrapper for converting methods to text based on a specified filter.
- MethodClassifier: Classifies methods based on their names, prefixes, suffixes, and pragmas.

Conclusion, Limitations, and Future Directions (?)

Limitations

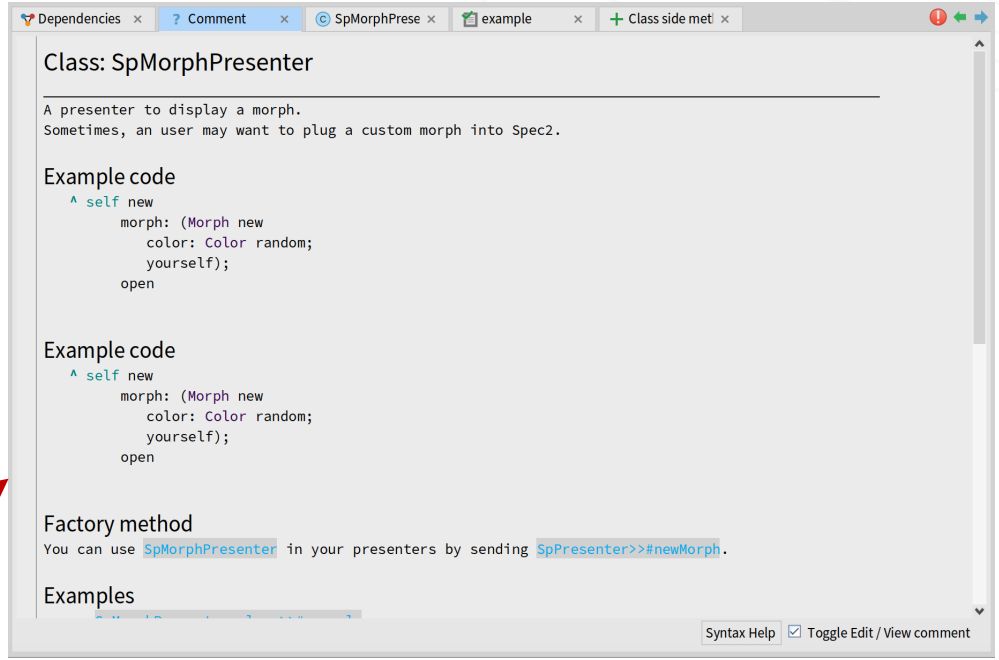
- Limited amount of evaluation per comment
- Needs more work on prompt tuning, document structure
- Weak solution for identifying collaborators

Conclusions:

- Generated comments are more complete, clear and useful than some human-made comments
  - → Maybe use when there are no comments ?

Future Directions

- Use heuristics for identifying collaborators & GenAI for describing these collaborations
- Adapt to existing dynamic comment features (e.g. examples)
- Automate a pipeline for comment suggestion in existing Pharo projects



References

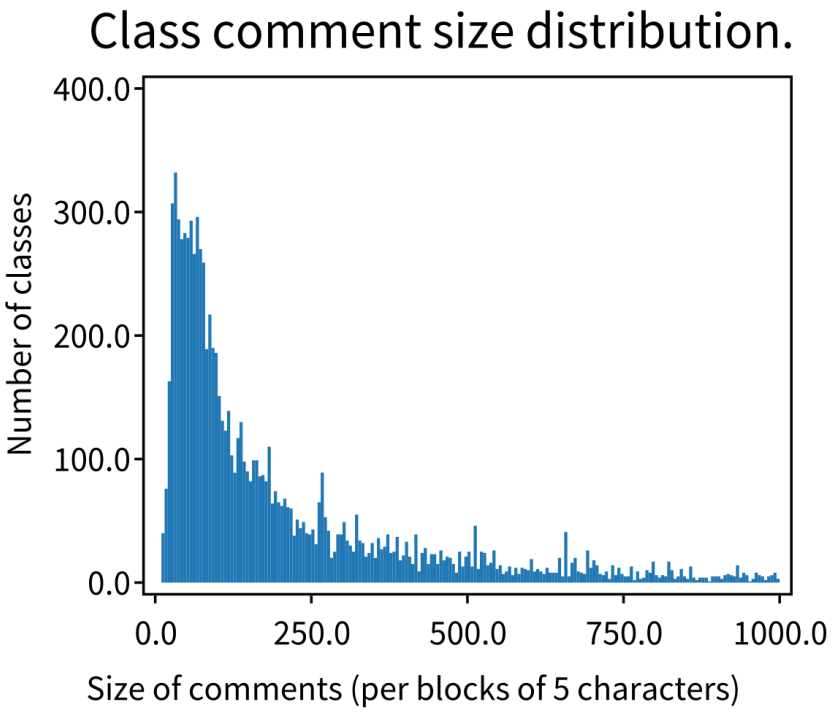
[1] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, S. Li, Measuring program comprehension: A large-scale field study with professionals, IEEE Transactions on Software Engineering 44 (2018) 951–976. doi:10.1109/TSE.2017.2734091.

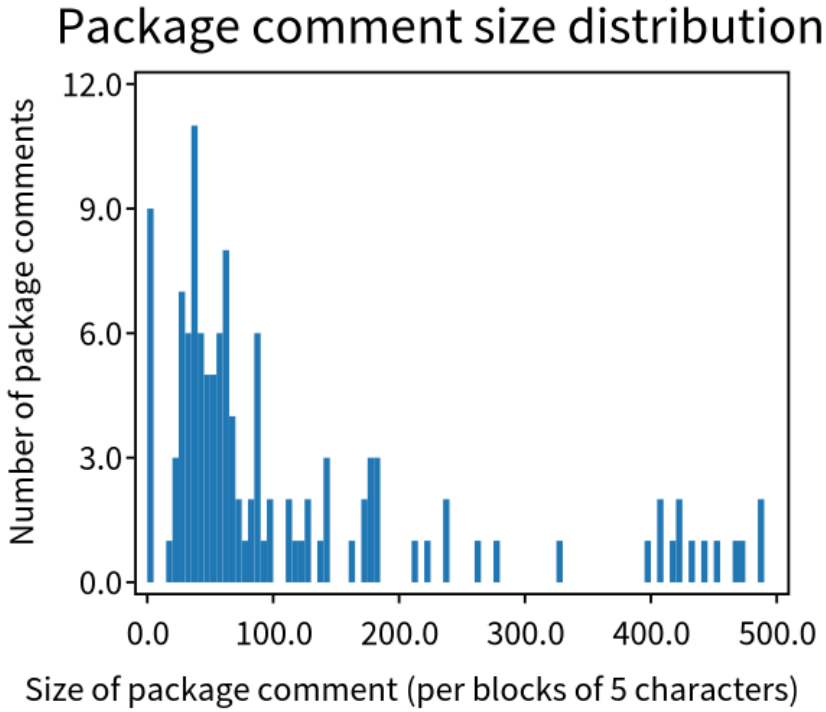
[2] K. Beck, W. Cunningham, A laboratory for teaching object oriented thinking, in: Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, OOPSLA '89, Association for Computing Machinery, New York, NY, USA, 1989, p. 1–6. doi:10.1145/74877.74879.

[3] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, Lost in the middle: How language models use long contexts, Transactions of the Association for Computational Linguistics 12 (2024) 157–173. doi:10.1162/tacl_a_00638.

[4] M. Barry, G. Caillaut, P. Halftermeyer, R. Qader, M. Mouayad, F. Le Deit, D. Cariolaro, J. Gesnouin, GraphRAG: Leveraging graph-based efficiency to minimize hallucinations in LLM-driven RAG for finance data, in: G. A. Gesese, H. Sack, H. Paulheim, A. Merono-Penuela, L. Chen (Eds.), Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK), International Committee on Computational Linguistics, Abu Dhabi, UAE, 2025, pp. 54–65.

Some stats


Class comment size distribution.
Number of classes
400.0
300.0
200.0
100.0
0.0
0.0
250.0
500.0
750.0
1000.0
Size of comments (per blocks of 5 characters)


Package comment size distribution
Number of package comments
12.0
9.0
6.0
3.0
0.0
0.0
100.0
200.0
300.0
400.0
500.0
Size of package comment (per blocks of 5 characters)