

Generating class comments in Pharo automatically

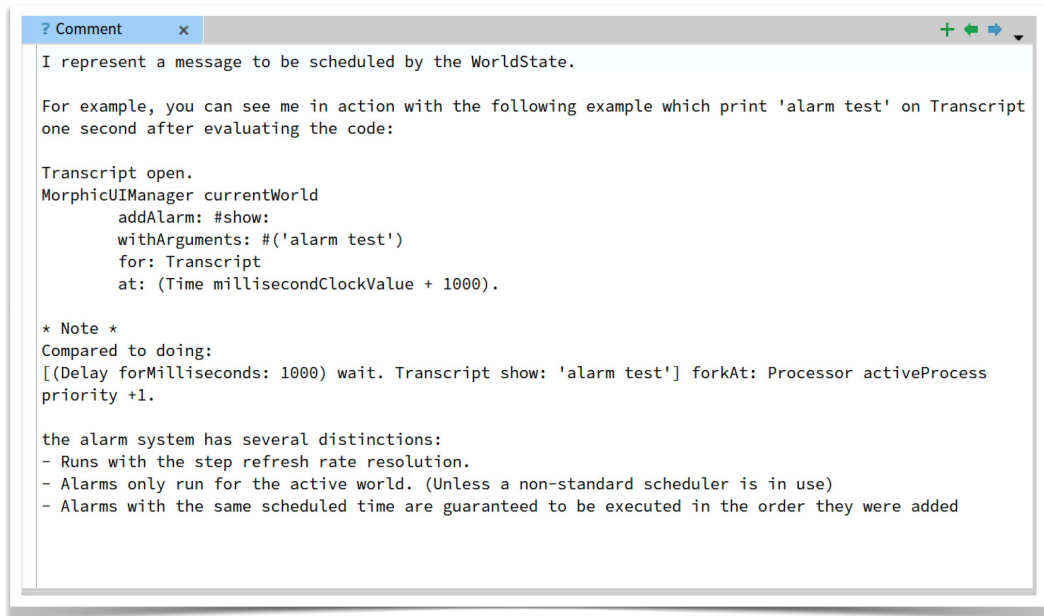
Lino Hess

Bachelor Thesis 1st presentation

Supervised by Pooja Rani

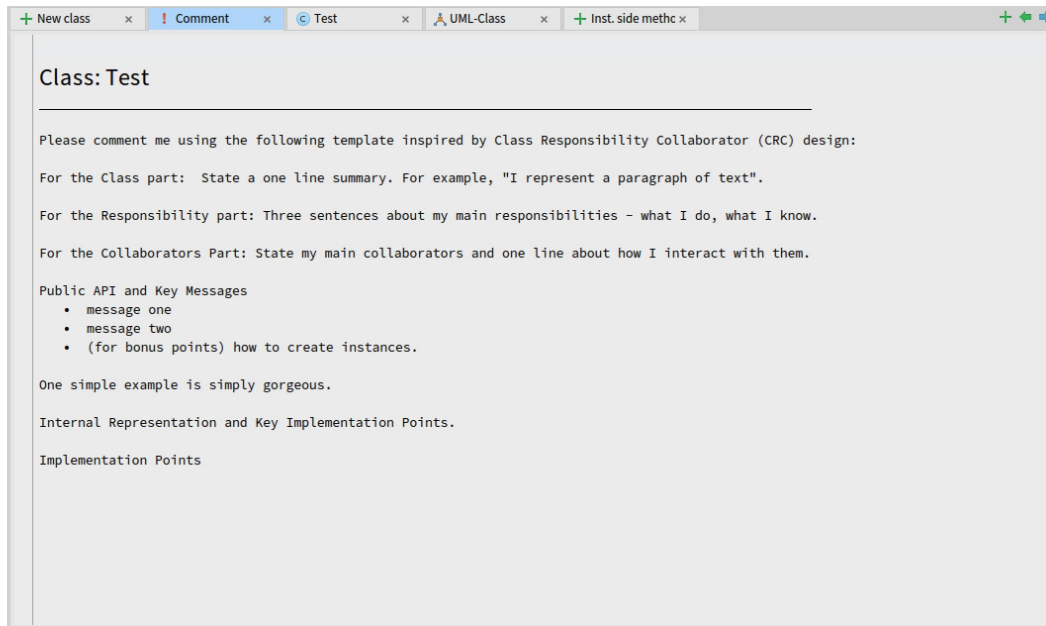
17.November 2020

Pharo class comment

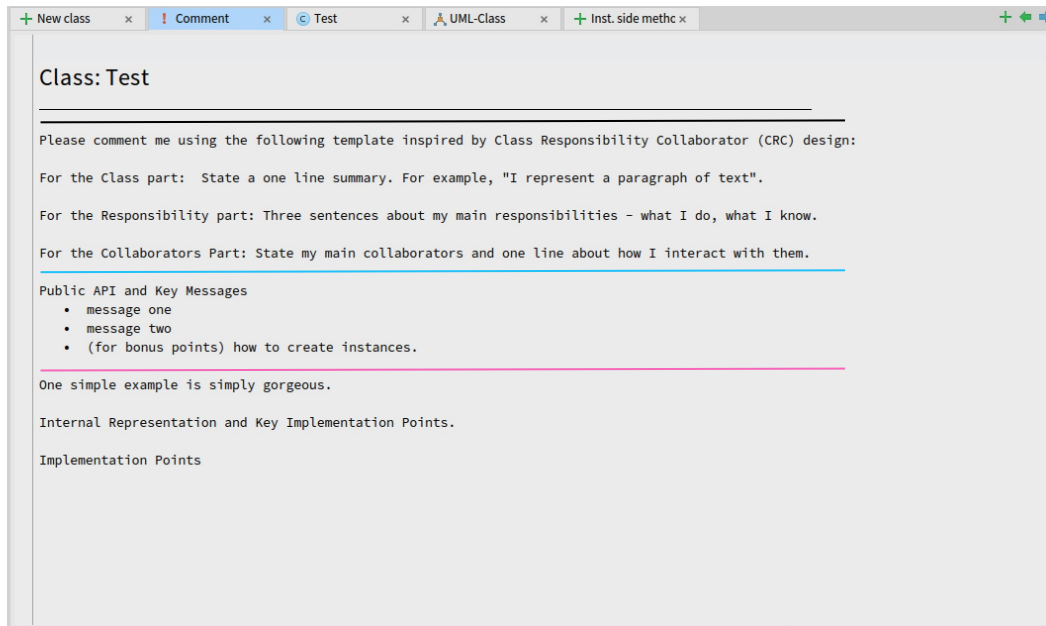


```
? Comment x + ← → ▾  
I represent a message to be scheduled by the WorldState.  
  
For example, you can see me in action with the following example which print 'alarm test' on Transcript  
one second after evaluating the code:  
  
Transcript open.  
MorphicUIManager currentWorld  
  addAlarm: #show:  
    withArguments: #('alarm test')  
    for: Transcript  
    at: (Time millisecondClockValue + 1000).  
  
* Note *  
Compared to doing:  
[[Delay forMilliseconds: 1000) wait. Transcript show: 'alarm test']] forkAt: Processor activeProcess  
priority +1.  
  
the alarm system has several distinctions:  
- Runs with the step refresh rate resolution.  
- Alarms only run for the active world. (Unless a non-standard scheduler is in use)  
- Alarms with the same scheduled time are guaranteed to be executed in the order they were added
```

Pharo comment template



Pharo comment template



The screenshot shows a Pharo IDE window with several tabs: 'New class', 'Comment', 'Test', 'UML-Class', and 'Inst. side methc'. The 'Comment' tab is active and displays a template for a class comment. The template is structured as follows:

```
Class: Test
```

Please comment me using the following template inspired by Class Responsibility Collaborator (CRC) design:

For the Class 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.

Public API and Key Messages

- message one
- message two
- (for bonus points) how to create instances.

One simple example is simply gorgeous.

Internal Representation and Key Implementation Points.

Implementation Points

- Class, Responsibility and Collaborators
- API and Key Messages
- Examples, implementations

Why do we want to generate comments?

- Possibility to spend less time on writing comments
- Create a uniform format to prevent inconsistent comments

Goal

Create a commenting tool written in Pharo

What are the challenges in generating comments automatically?

- Which approach to use?
- How do we define the heuristics to use?

Related work in Java

Automatic Generation of Natural Language Summaries for Java Classes

Laura Moreno¹, Jairo Aponte², Gopirasad Sridhara³, Andrian Marcus¹, Lori Pollock⁴, K. Vijay-Shanker⁵

¹Wayne State University
Detroit, MI, USA
{morenoe, amarcus}@wayne.edu

²Universidad Nacional de Colombia
Bogotá, Colombia
jlapontem@unal.edu.co

³IBM Research India
Bangalore, India
gsridha@in.ibm.com

⁴University of Delaware
Newark, DE, USA
{pollock, vjshay}@cs.udel.edu

Abstract—Most software engineering tasks require developers to understand parts of the source code. When faced with unfamiliar code, developers often rely on (internal or external) documentation to gain an overall understanding of the code and determine whether it is relevant for the current task. Unfortunately, the documentation is often absent or outdated.

This paper presents a technique to automatically generate human readable summaries for Java classes, assuming no documentation exists. The summaries allow developers to understand the main goal and structure of the class. The focus of the summaries is on the content and responsibilities of the classes, rather than their relationships with other classes. The summarization tool determines the class and method stereotypes and uses them, in conjunction with heuristics, to select the information to be included in the summaries. Then it generates the summaries using existing formalization tools.

A group of programmers judged a set of generated summaries for Java classes and determined that they are readable and understandable, they do not include extraneous information, and, in most cases, they are not missing essential information.

Index Terms—Source code summarization, program comprehension, documentation generation.

1. INTRODUCTION

Existing studies [1] revealed that developers often spend more time searching, browsing, and reading the code than editing it. Searching, browsing, and reading are essential activities needed to understand software, which in turn is needed for everyday software maintenance tasks. While browsing the source code, developers sometimes just glance at it to get a quick understanding and sometimes spend more time reading it in detail [1-3]. Skimming the code is performed in order to determine whether a specific part of it is relevant to the task at hand or not. When the code has good leading comments, developers can acquire a quick understanding of the code artifact. Unfortunately, more often than not, good comments are missing or outdated, and therefore, developers must spend much more time reading the code in detail, in order to gain even a superficial understanding.

One approach to overcome this problem is to automatically generate descriptive comments directly from the source code. While successfully applied for Java methods [4], generating comments for more complex code artifacts, e.g., classes, is significantly more difficult [5, 6]. Our focus here is on classes, as they are the primary decomposition unit in Object-Oriented (OO) programming languages, such as Java. In addition, the

OO paradigm supports reasoning at the object level and, consequently, code understanding and reuse at the class level.

Unfortunately, we cannot use existing comment generation tools for methods (e.g., [4]) and simply merge them to create a class summary. The reasons vary: (i) classes bundle together more than just methods – they also include data that the methods presumably operate on; (ii) adding together all method descriptions would result in very large summaries, which defeats their goal; (iii) not all methods are the same – some may be relevant to describe the behavior of the class instances, while some may not.

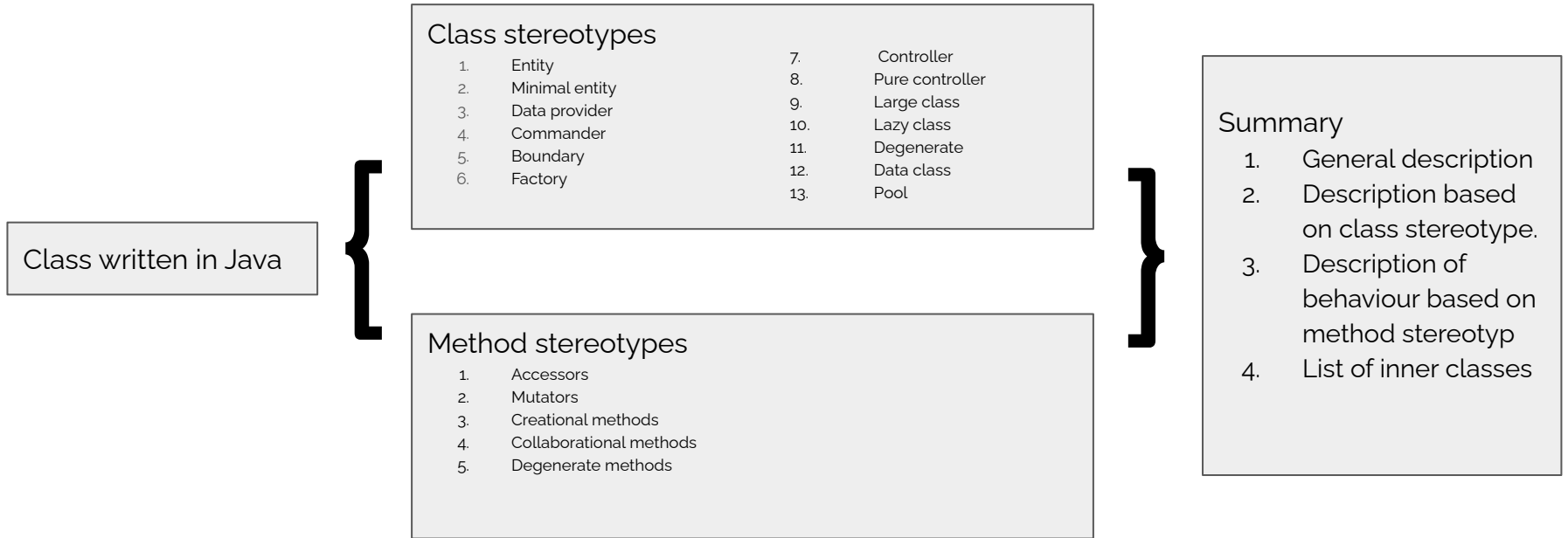
We propose in premiere a technique to automatically generate structured natural-language descriptions for Java classes, independent of their context and assuming that no documentation exists (i.e., if it exists, the comments are not currently used). The system takes a Java project as input, and for each class, it outputs a natural-language summary. The goal of the generated summaries is to support the quick understanding of a class by describing its intent and leaving aside its context and any algorithmic details. In this sense, the summaries are *informative* (i.e., provide a brief description of the class content), *abstractive* (i.e., include information that is not explicit in the class), and *generic* (i.e., attempt to cover only the important information of the class).

The intended audience is any developer, especially a novice, who is unfamiliar with the code and needs to quickly get the gist of the class to decide whether to peruse the source code or not. For example, the developer may be deciding whether to reuse a class X and wondering whether it would serve her needs, or, while reading the code of another class, she encounters an attribute of type X and wonders what it means. Developers sometimes write comments that describe the main responsibility of a class, to help other developers, regardless of their task. Our automatic summaries have the same goal. Although different maintenance tasks require different kinds of information from classes, our approach can serve as an initial step in the generation of specific-purpose summaries, which is outside the scope of this paper.

Our conjecture is that the type of methods and their distribution in a class is not accidental and denotes some design intent, which reflects the main goal of the class. Thus, our summarization technique first determines the stereotypes of the class [7] and each one of its methods [8]. The stereotype information is used in conjunction with predefined heuristics, to select the information that will be included in the summary.

- Moreno et Al.
- Focused mostly on the responsibilities of the classes

Heuristic-based process of Moreno et Al.



Our approach

- Heuristic based
- Corresponding to the template format

Related work in Pharo

EMSE manuscript No.
(will be inserted by the editor)

What do class comments tell us? An investigation of comment evolution and practices in Pharo

Pooja Rani · Sebastiano Panichella · Manuel Leuenberger · Mohammad Ghafari · Oscar Nierstrasz

Received: date / Accepted: date

Abstract Previous studies have characterized code comments in different programming languages, and have shown how a high quality of code comments is crucial to support program comprehension activities and to improve the effectiveness of maintenance tasks. However, very few studies have focused on the analysis of the information embedded in code comments. None of them compared the developer's practices to write the comments to the standard guidelines and analyzed these characteristics in the Pharo Smalltalk environment.

The class commenting practices have their origins in Smalltalk80, going back 40 years. Smalltalk traditionally separates class comments from source code, and offers a brief template for entering a comment for newly-created classes. These templates have evolved over the years, particularly in the Pharo environment. This paper reports the first empirical study investigating commenting practices in Pharo Smalltalk. As a first step, we analyze class comment evolution over seven Pharo versions. Then, we quantitatively and qualitatively analyze class comments of the most recent version of Pharo, to investigate the information types of Pharo comments. Finally, we study the adherence of developer commenting practices to the class template over Pharo versions.

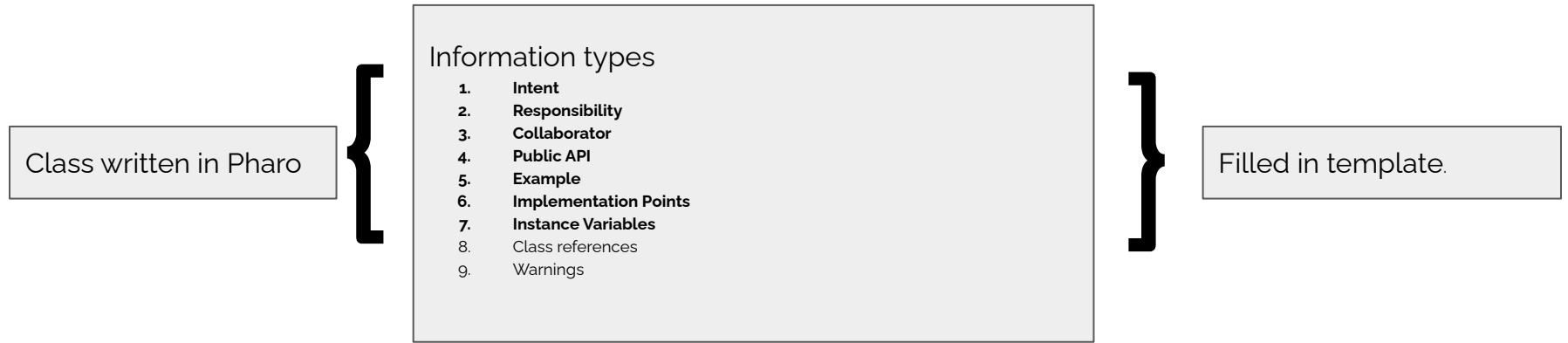
The results of this study show that there is a rapid increase in class comments in the initial three Pharo versions, while in subsequent versions developers added comments to both new and old classes, thus maintaining a similar ratio. In addition, the analysis of the semantics of the comments from the latest Pharo version suggests that 23 information types are typically embedded in class comments by developers and that only seven of them are present in the latest *Pharo class comment template*. However, the information types proposed by the standard template tend to be present more often than other types of information. Additionally, we find that a substantial proportion of comments follow the writing style of the template in writing these information types, but they are written and formatted in a non-uniform way. This suggests the need to standardize the commenting guidelines for formatting the text.

Pooja Rani, Manuel Leuenberger, Mohammad Ghafari, Oscar Nierstrasz
Software Composition Group, University of Bern, 3012 Bern, Switzerland
<http://scg.unibe.ch/en/>

Sebastiano Panichella
Zürich University of Applied Science
E-mail: punc@zhaw.ch

- Analyzed Class comments
- Found various information types embedded in class comments
- Many comments were written and formatted in a non-uniform way

Heuristic-based process in Pharo



Pipeline

