

# Class Comments Analysis

Seminar Software Composition, Bern University H2019

Ivan Kravchenko

Supervised by Pooja Rani

# Motivation

- Class comments provides high-level overview
- Helps to understand complex programs

# Problem

Different programming languages follow different programming convention

- Contain different information types
- Follow different style guidelines
- Tool support exist for writing proper comments

# Java

## Class comment example:

```
/**
 * A class representing a window on the screen.
 * For example:
 * <pre>
 *     Window win = new Window(parent);
 *     win.show();
 * </pre>
 *
 * @author Sami Shaio
 * @version 1.13, 06/08/06
 * @see java.awt.BaseWindow
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

# Python

```
class ExampleError(Exception):  
    """Exceptions are documented in the same way as classes.  
  
    The __init__ method may be documented in either the class level  
    docstring, or as a docstring on the __init__ method itself.  
  
    Either form is acceptable, but the two should not be mixed. Choose one  
    convention to document the __init__ method and be consistent with it.  
  
    Note:  
        Do not include the `self` parameter in the ``Args`` section.  
  
    Args:  
        msg (str): Human readable string describing the exception.  
        code (:obj:`int`, optional): Error code.  
  
    Attributes:  
        msg (str): Human readable string describing the exception.  
        code (int): Exception error code.  
  
    """
```

### Class comment example:

```
/**
 * A class representing a window on the screen.
 * For example:
 * <pre>
 *   Window win = new Window(parent);
 *   win.show();
 * </pre>
 *
 * @author Sami Shaio
 * @version 1.13, 06/08/06
 * @see java.awt.BaseWindow
 * @see java.awt.Button
 */
class Window extends BaseWindow {
    ...
}
```

```
class ExampleError(Exception):
    """Exceptions are documented in the same way as classes.

    The __init__ method may be documented in either the class level
    docstring, or as a docstring on the __init__ method itself.

    Either form is acceptable, but the two should not be mixed. Choose one
    convention to document the __init__ method and be consistent with it.

    Note:
        Do not include the 'self' parameter in the ``Args`` section.

    Args:
        msg (str): Human readable string describing the exception.
        code (:obj:`int`, optional): Error code.

    Attributes:
        msg (str): Human readable string describing the exception.
        code (int): Exception error code.

    """
```

```
class ExampleError(Exception):
    """Exceptions are documented in the same way as classes.

    The __init__ method may be documented in either the class level
    docstring, or as a docstring on the __init__ method itself.

    Either form is acceptable, but the two should not be mixed. Choose one
    convention to document the __init__ method and be consistent with it.

    Note
    ----
    Do not include the 'self' parameter in the ``Parameters`` section.

    Parameters
    -----
    msg : str
        Human readable string describing the exception.
    code : :obj:`int`, optional
        Numeric error code.

    Attributes
    -----
    msg : str
        Human readable string describing the exception.
    code : int
        Numeric error code.

    """
```

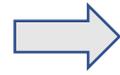
# Problem

We investigate class comments

- What information they contain?
- How they're influenced by the style guidelines?
- What tools support exist for writing class comments?

# Methodology

Select programming language



Select projects



Extract code comments



Analyze

- Popular
- Documentation guidelines
- Big developer community

- Open-source project
- Decent code/comment ratio
- Have style guidelines

- Separate comments
- Class comments
- Gather style guideline

- Manually analyze
- Categorize into existing taxonomy

# Related work on comments

## Classifying code comments in Java open-source software systems

Luca Pascarella  
Delft University of Technology  
Delft, The Netherlands  
L.Pascarella@tudelft.nl

Alberto Bacchelli  
Delft University of Technology  
Delft, The Netherlands  
A.Bacchelli@tudelft.nl

**Abstract**—Code comments are a key software component containing information about the underlying implementation. Several studies have shown that code comments enhance the readability of the code. Nevertheless, not all the comments have the same goal and target audience. In this paper, we investigate how six diverse Java OSS projects use code comments, with the aim of understanding their purpose. Through our analysis, we produce a taxonomy of source code comments; subsequently, we investigate how often each category occur by manually classifying more than 2,000 code comments from the aforementioned projects. In addition, we conduct an initial evaluation on how to automatically classify code comments at line level into our taxonomy using machine learning; initial results are promising and suggest that an accurate classification is within reach.

### I. INTRODUCTION

While writing and reading source code, software engineers routinely introduce code comments [6]. Several researchers investigated the usefulness of these comments, showing that thoroughly commented code is more readable and maintainable. For example, Woodfield *et al.* conducted one of the first experiments demonstrating that code comments improve program readability [35]; Tenny *et al.* confirmed these results with more experiments [31], [32]. Hartzman *et al.* investigated the economical maintenance of large software products showing that comments are crucial for maintenance [12]. Jiang *et al.* found that comments that are misaligned to the annotated functions confuse authors of future code changes [13]. Overall, given these results, having abundant comments in the source code is a recognized good practice [4]. Accordingly, researchers proposed to evaluate code quality with a new metric based on code/comment ratio [21], [9].

Nevertheless, not all the comments are the same. This is evident, for example, by glancing through the comments in a source code file<sup>1</sup> from the Java Apache Hadoop Framework [1]. In fact, we see that some comments target end-user programmers (*e.g.*, Javadoc), while others target internal developers (*e.g.*, *inline* comments); moreover, each comment is used for a different purpose, such as providing the implementation rationale, separating logical blocks, and adding reminders; finally, the interpretation of a comment also depends on its position with respect to the source code.

Defining a taxonomy of the source code comments that developers produce is an open research problem.

Haouari *et al.* [11] and Steidl *et al.* [28] presented the earliest and most significant results in comments' classification. Haouari *et al.* investigated developers' commenting habits, focusing on the position of comments with respect to source code and proposing an initial taxonomy that includes four high-level categories [11]; Steidl *et al.* proposed a semi-automated approach for the quantitative and qualitative evaluation of comment quality, based on classifying comments in seven high-level categories [28]. In spite of the innovative techniques they proposed to both understanding developers' commenting habits and assessing comments' quality, the classification of comments was not in their primary focus.

In this paper, we focus on increasing our empirical understanding of the types of comments that developers write in source code files. This is a key step to guide future research on the topic. Moreover, this increased understanding has the potential to (1) improve current quality analysis approaches that are restricted to the comment ratio metric only [21], [9] and to (2) strengthen the reliability of other mining approaches that use source code comments as input (*e.g.*, [30], [23]).

To this aim, we conducted an in-depth analysis of the comments in the source code files of six major OSS systems in Java. We set up our study as an exploratory investigation. We started without hypotheses regarding the content of source code comments, with the aim of discovering their purposes and roles, their format, and their frequency. To this end, we (1) conducted three iterative content analysis sessions (involving four researchers) over 50 source files including about 250 comment blocks to define an initial taxonomy of code comments, (2) validated the taxonomy externally with 3 developers, (3) inspected 2,000 source code files and manually classified (using a new application we devised for this purpose) over 15,000 comment blocks comprising more than 28,000 lines, and (4) used the resulting dataset to evaluate how effectively comments can be automatically classified.

Our results show that developers write comments with a large variety of different meanings and that this should be taken into account by analyses and techniques that rely on code comments. The most prominent category of comments summarizes the purpose of the code, confirming the importance of research related to automatically creating this type of comments. Finally, our automated classification approach

## Classifying code comments in Java open-source software systems

Luca Pascarella, Alberto Bacchelli

## Classifying Python Code Comments Based on Supervised Learning

Jingyi Zhang<sup>1</sup>, Lei Xu<sup>2(✉)</sup>, and Yanhui Li<sup>2</sup>

<sup>1</sup> School of Management and Engineering, Nanjing University,  
Nanjing, Jiangsu, China  
jyzhangchn@outlook.com

<sup>2</sup> Department of Computer Science and Technology, Nanjing University,  
Nanjing, Jiangsu, China  
{xlei, yanhui}@nju.edu.cn

**Abstract.** Code comments can provide a great data source for understanding programmer's needs and underlying implementation. Previous work has illustrated that code comments enhance the reliability and maintainability of the code, and engineers use them to interpret their code as well as help other developers understand the code intention better. In this paper, we studied comments from 7 python open source projects and contrived a taxonomy through an iterative process. To clarify comments characteristics, we deploy an effective and automated approach using supervised learning algorithms to classify code comments according to their different intentions. With our study, we find that there does exist a pattern across different python projects: *Summary* covers about 75% of comments. Finally, we conduct an evaluation on the behaviors of two different supervised learning classifiers and find that Decision Tree classifier is more effective on accuracy and runtime than Naive Bayes classifier in our research.

**Keywords:** Code comments classification · Supervised learning  
Python

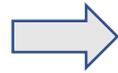
## Classifying Python Code Comments Based on Supervised Learning

Jingyi Zhang, Lei Xu, Yanhui Li

Java	Python
SUMMARY	Summary
EXPAND	Summary, Expand
RATIONALE	Summary
DEPRECATION	-
EXCEPTION	Exception
TODO	Todo
INCOMPLETE	-
COMMENTED CODE	-
DIRECTIVE	-
FORMATTER	-
LICENSE	Metadata
OWNERSHIP	Metadata
POINTER	partly Links
AUTOMATICALLY GENERATED	-
NOISE	noise
partly USAGE	Parameters
USAGE	Usage
-	Version
partly TODO, INCOMPLETE	Development Notes

# Methodology

Select programming language



Select projects



Extract code comments



Analyze

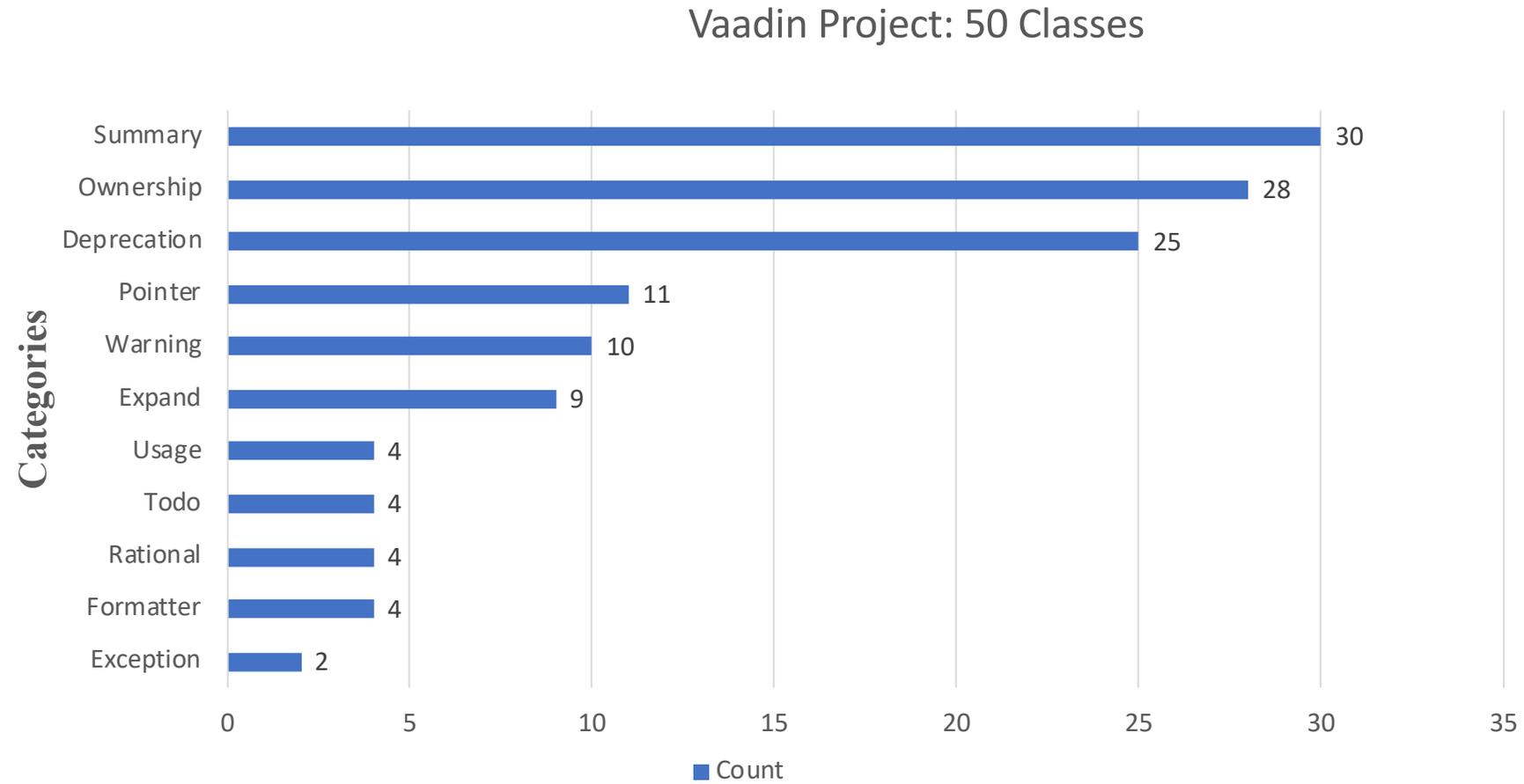
Java

- Apache Spark
- Apache Hadoop
- Eclipse
- Vaadin
- Guava
- Guice

- Class comments
- Style guidelines

- Representative sample set
- From each project

# Initial Results



# What are the different style guidelines?

Apache

Apache Spark  
Apache Hadoop

Google

Guava  
Guice

Oracle

Vaadin  
Eclipse

# How do they cover documentation?

- Extract comment related guidelines.
- Rules existing for writing comments:
  - What content should be written?
  - In what style the content should be written?

# Examples of style guidelines

## Oracle:

- Class/interface/field descriptions can omit the subject and simply state the object
- A class should use tags like @since, @version, @author

## Google:

- A summary fragment should not be complete sentence

## Hadoop:

- Do not use @author tags

# Tool support

- Checkstyle
- PMD
- Findbugs
- JaCoCo

Each tool has a set of rules to check for style guideline and code practices

# Tool support

<b>Tool</b>	<b>Comment formatting</b>	<b>Comment content</b>	<b>Comment size</b>	<b>Required tag</b>
Checkstyle	✓	✓	✓	✓
PMD	✓	✓	✓	—
Findbugs	—	—	—	—
JaCoCo	—	—	—	—

Everything is related to syntax rules, limited checks related to content

# Challenges

- How to define a class?  
(annotation, interface, inner class, enum, package-based data)
- Orphan comments and dangling comments

```
*      bind(PaymentService.class).to(CreditCardPaymentService.class);
*      bindConstant().annotatedWith(Names.named("port")).to(8080);
*    }
*  }
* </pre>
*
* @author crazybob@google.com (Bob Lee)
*/
// some comment
public abstract class AbstractModule implements Module {

    /**
     * Inner class with javadoc comment
     */
    class Module {
        int a;
        int b;
    }
}
```

- Taxonomy mapping from Java to Python
- Extracting style guidelines related to project

# Forthcoming plans

- Analyze remaining Java projects
- Create same dataset for python
- Analyze python comments
- Compare with python style guidelines
- How NLP can help to analyze guidelines that are not covered by the statistical analysis tools
- Comparison of differences between java and python class comments

?