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:

```java
/**
 * 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 comment example:

```java
/**
 * 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 {
    ...
}
```

```java
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.
    Attributes:
    msg (str): Human readable string describing the exception.
    code (int): Exception 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
- Popular
- Documentation guidelines
- Big developer community

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

Extract code comments
- Separate comments
- Class comments
- Gather style guideline

Analyze
- Manually analyze
- Categorize into existing taxonomy
Related work on comments

Classifying code comments in Java open-source software systems
Luca Pascarella, Alberto Baccelli

Classifying Python Code Comments Based on Supervised Learning
Jingyi Zhang, Lei Xu, Yanhui Li

---

Abstract. Code comments can provide a great data source for understanding programmers’ 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 code comments in Java open-source software systems
Luca Pascarella, Alberto Baccelli

Classifying Python Code Comments Based on Supervised Learning
Jingyi Zhang, Lei Xu, Yanhui Li
<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>Summary</td>
</tr>
<tr>
<td>EXPAND</td>
<td>Summary, Expand</td>
</tr>
<tr>
<td>RATIONALE</td>
<td>Summary</td>
</tr>
<tr>
<td>DEPRECATION</td>
<td>-</td>
</tr>
<tr>
<td>EXCEPTION</td>
<td>Exception</td>
</tr>
<tr>
<td>TODO</td>
<td>Todo</td>
</tr>
<tr>
<td>INCOMPLETE</td>
<td>-</td>
</tr>
<tr>
<td>COMMENTED CODE</td>
<td>-</td>
</tr>
<tr>
<td>DIRECTIVE</td>
<td>-</td>
</tr>
<tr>
<td>FORMATTER</td>
<td>-</td>
</tr>
<tr>
<td>LICENSE</td>
<td>Metadata</td>
</tr>
<tr>
<td>OWNERSHIP</td>
<td>Metadata</td>
</tr>
<tr>
<td>POINTER</td>
<td>partly Links</td>
</tr>
<tr>
<td>AUTOMATICALLY GENERATED</td>
<td>-</td>
</tr>
<tr>
<td>NOISE</td>
<td>noise</td>
</tr>
<tr>
<td>partly USAGE</td>
<td>Parameters</td>
</tr>
<tr>
<td>USAGE</td>
<td>Usage</td>
</tr>
<tr>
<td>-</td>
<td>Version</td>
</tr>
<tr>
<td>partly TODO, INCOMPLETE</td>
<td>Development Notes</td>
</tr>
</tbody>
</table>
Methodology

Select programming language
Java

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

Extract code comments
- Class comments
- Style guidelines

Analyze
- Representative sample set
- From each project
Initial Results

Vaadin Project: 50 Classes

Categories

- Summary: 30
- Ownership: 28
- Deprecation: 25
- Pointer: 11
- Warning: 10
- Expand: 9
- Usage: 4
- Todo: 4
- Rational: 4
- Formatter: 4
- Exception: 2

Count
What are the different style guidelines?

<table>
<thead>
<tr>
<th>Apache</th>
<th>Google</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Spark</td>
<td>Guava</td>
<td>Vaadin</td>
</tr>
<tr>
<td>Apache Hadoop</td>
<td>Guice</td>
<td>Eclipse</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Tool</th>
<th>Comment formatting</th>
<th>Comment content</th>
<th>Comment size</th>
<th>Required tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkstyle</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PMD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td>Findbugs</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>JaCoCo</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

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
• 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