Herbelin Ludovic
Seminar Software Composition 2020

Software Comments Quality Metrics

Introduction

This document lists some of the software comment quality metrics, the goals and how effective they are based on a few papers. [refs]

The goal for this project is to analyze the class comments so the applicability will only be related to this specific type.

Metrics goals

Based on [Quality Analysis of Source Code Comments] the quality of the source code comments has 4 main criterias :

- **Coherence :** How code and comments relate, member comments up-to-date with the method name. Comments should provide additional information than method name.
- **Usefulness :** Comments should be perceived as helpful by the readers. If the code is not harder to read without the comment, it is useless.
- **Completeness :** Comment should appear all the time at certain positions, E.G. copyright, header for class, method and field of an API.
- **Consistency :** comments should be written in the same language (usually english), using the same coypright format, etc.

Coherence

The two following metrics for coherence are defined in the paper [Quality Analysis of Source Code Comments].

Levensthein distance

Using the distance between words in the comment and in the method, compare if the comment is too similar (no use) or too dissimilar (method should be renamed) to the method name.

Levenshtein distance : Words are considered similar if $d(w1, w1) < 2$ - Then compute the c_{coeff} for the comment $c_{coeff} = \frac{N_{similar}}{N_{words}}$ - Experimentally determined $c_{coeff} = 0$ -> not enough coherence, add info the emphasize relation with method identifier - Experimentally determined $c_{coeff} > 0.5$ -> too trivial, does not contain additional info

Applicability

This is could be used for example in a way using class names if they are consistent such as in Pharo `ASTCacheResetTest` which would be separated into the words `['AST', 'Cache', 'Reset', 'Test']` to see if the c_{coeff} is in an acceptable threshold.

Comment length

Experimentally (after normalization) comments with 1-2 words can be deleted. Comments with at least 30 words are too complex for the length to be used.

Applicability

This would be very much applicable, although the thresholds would probably need to be modified since class comments are usually longer than inline or method comments.

Usefulness

As stated in the paper [Automatically Assessing Code Understandability- How Far Are We], automatically understanding code is a very tedious task. They also demonstrated using surveys that the 120+ metrics they compiled from other studies are not representative of the actual code understandability.

Thus, usefulness is very hard to measure since it depends a lot on the personal level of knowledge of the developer using metrics.

Completeness

Code / comment consistency

The following metrics defined in the [Automatic Quality Assessment of Source Code Comments: The JavadocMiner] paper all relate to the completeness of the comments in the project. They measure whether there are enough comments and if they document all aspects of the code.

1. **Documentable Item Ratio Heuristic (DIR)** : Must document all aspects of method @throws, @return, @parameter, etc.
2. **Any Javadoc Comment heuristic (ANYJ)** : Ratio of nb of identifiers (?) to total number of identifiers
3. **Sync Heuristics** : Return types, Parameters, Exceptions must be valid (up-to-date), IE having correct name of the type, parameters names and exceptions names
    - **RSYNC** : Return type
    - **PSYNC** : Parameters
    - **ESYNC** : Exceptions

Applicability

The **ANYJ (1.)** could be used for class comments, **DIR (2.)** and the **SYNC (3.)** not so much for class comments since those informations are not contained in the class comments.

Consistency

Since we are only analyzing class comments at this time, this would fall kind of in the comment length that was seen previously.

We want to detect if comments are present all the time at specific places, which leaves only the class comments for us.

Various metrics for Natural Language quality

This does not fall specifically in either of the 4 criterias, maybe a bit in the consistency and coherence. Those metrics are mostly about measuring an "ease of reading" for the developer.

The paper [Automatic Quality Assessment of Source Code Comments: The JavadocMiner] defines the following 4 metrics to measure natural language quality.

1. **Token, Noun, Verb count heuristics** : detect well formed sentences in the language.
2. **Words Per Javadoc Comment (WPJC)** : detect members, classes, etc. that could be under documented -> similar to the comment length seen previously
3. **Abbreviation Count Heuristic (ABB)** : detect number of abreviations (to avoid)
4. **Readability heuristics** : Fog index, Flesch Reading Ease Level, Flesch-Kincaid Grade Level Score -> in the paper studies infeasible for source code comments ?

All of these could be used for class comments, as they are general metrics for the text written.