A New Dataset for Source Code Comment Coherence

Anna Corazza
DIETI,
Univ. di Napoli Federico II
anna.corazza@unina.it

Valerio Maggio
Fondazione
Bruno Kessler
vmaggio@fbk.eu

Giuseppe Scanniello
Dept. of Mathematics, Information
Technology, and Economics
Univ. della Basilicata
giuseppe.scanniello@unibas.it

Abstract

English. Source code comments provide useful insights on a codebase and on the intent behind design decisions and goals. Often, the information provided in the comment of a method and in its corresponding implementation may be not coherent with each other (i.e., the comment does not properly describe the implementation). Several could be the motivations for this issue (e.g., comment and source code do not evolve coherently). In this paper, we present the results of a manual assessment on the coherence between comments and implementations of $3,636$ methods, gathered from 4 Java open-source software. The results of this assessment has been collected in a dataset that we made publicly available on the web. We also sketch here the protocol to create this dataset.

Italiano. *I commenti al codice sorgente forniscono informazioni utili sull'implementazione del codice e sulle intenzioni relative alle decisioni e agli scopi del progetto. Spesso, le informazioni presenti nel commento di un metodo e nella sua implementazione possono non essere coerenti (nel senso che il commento non dà una descrizione adeguata dell'implementazione). Ci possono essere diverse spiegazioni per questo (ad esempio, commenti e codice sorgente non sono stati modificati in modo coerente). In questo articolo, presentiamo i risultati di una valutazione manuale della coerenza tra commenti e implementazione di* $3,636$ *metodi, raccolti da* 4 *applicazioni open source in Java. I risultati di questa valutazione sono stati raccolti in un dataset che abbiamo pubblicato sul web. Accenniamo anche al protocollo seguito per la preparazione del dataset.*

1 Introduction

Natural language is used in different ways in the development process of a software system and therefore techniques of natural language processing (NLP) and information retrieval (IR) are more and more frequently integrated into software development and maintenance tools. Many automated or semi-automated techniques have been proposed to aid developers in the comprehension and in the evolution of existing systems, e.g., (Corazza et al., 2016; Scanniello et al., 2010).

Natural language information is provided in source code comments and in the name of identifiers. In the former case, standard natural language is usually adopted, although quite technical. Comments are written in English, even when developers have different mother-tongues. On the other hand, identifiers are typically constructed by composing multiple terms and abbreviations. Therefore, more sophisticated techniques are necessary to extract the lexical information contained in each identifier (Corazza et al., 2012).

Most of these techniques assume that the same words are used whenever referring to a particular concept (Lawrie et al., 2010). In many cases, this represents an oversimplification: methods are often modified without updating the corresponding comments (Salviulo and Scanniello, 2014). In these cases, comments might convey information unrelated or inconsistent with the corresponding implementation. Nevertheless, comments are extremely important because they are

expected to convey the main intent behind design decisions, along with some implementation details (e.g., types of parameters and of returned values).

Therefore, more sophisticated models are necessary to determine if there is coherence between the lexicon provided in comments and in its corresponding source code. Hence, there exists coherence between a lead comment of a method and its source code (also simply coherence, from here on) if that comment describes the intent of the method and its actual implementation.

In this work, we focus on the lead comment of methods. This kind of comments precedes the definition of a given method and is supposed to provide its documentation and details about the implementation. We discuss here a dataset we made publicly available on the web.[1] It contains annotations about the coherence of $3,636$ methods collected from 4 implementations of 3 open source projects written in Java. The defined protocol used for its creation is also sketched, to give researchers the opportunity to possibly extend it. Further details on this protocol can be found in (Corazza et al., 2015).

For the assessment of the quality of the annotation with special focus on computational linguistics applications, a few indexes have been considered (Eugenio and Glass, 2004; Artstein and Poesio, 2008; Mathet et al., 2015), among which the *kappa index* (Cohen, 1960) is the most widely adopted because of its favorable characteristics. The inter-annotator agreement has therefore been assessed by this parameter.

We expect that making freely available this dataset could give impulse to the research for approaches to assess the coherence between the implementation of a method and its lead comment (simply coherence, from here on). In fact, although no approach has been yet proposed in this regard, they could be of great help for software maintenance and evolution activities.

The paper is structured as follows. In Section 2, we discuss the methodology used to create our dataset. A description of the main characteristics of the dataset is given in Section 3, while in Section 4 the annotation is assessed. Some final considerations conclude the paper.

[1] www2.unibas.it/gscanniello/coherence/

2 Dataset Construction

To create our dataset, we adopted the perspective-based and the checklist-based review methods (Wohlin et al., 2012). The perspective is the one of the Researcher aiming at assessing the coherence between the lead comment of a method and its implementation. The process of creation is based on the following elements:

1. **Working Meetings.** We used meetings to determine the goals of our research work and the process to create the dataset.
2. **Dataset Creation.** We instantiated the defined process to create our dataset.
3. **Outcomes.** We gathered results during and after the creation of our dataset.
4. **Publishing Results and Dataset.** We shared our experience with the community in (Corazza et al., 2015) and released the dataset on the web.

The construction of the dataset has been completed in two main consecutive phases by using an ad-hoc web system implemented for the purpose:

- **Verify coherence**. Annotators verify by means of a checklist the coherence between the lead comments of a set of methods and their corresponding implementation.
- **Resolve conflicts**. The intervention of experts is required whenever the judgements of the annotators differ. In our case, two of the authors, with a background in software engineering, assumed the role of experts and examined the problematic cases. For each conflicting method, the experts should reach an agreement about the coherence or the non-coherence. Methods on which experts do not get a consensus are automatically discarded.

3 Dataset Description

Some descriptive statistics (e.g., number of classes and methods) of the software systems in our dataset are shown in Table 1.

- **CoffeeMaker**[2] is a software to manage inventory and recipes and to purchase beverages. We chose this software because it has

[2] agile.csc.ncsu.edu/SEMaterials/tutorials/coffee_maker/

Table 1: Descriptive statistics of the software systems in the dataset: N_f stands for the number of files, N_c for the number of classes, N_m for the number of methods and N_m^* for the number of methods with lead comments.

System	Version	N_f	N_c	N_m	N_m^*
CoffeeMaker	-	7	7	51	47 (92%)
JFreeChart 6.0	0.6.0	82	83	617	485 (79%)
JFreeChart 7.1	0.7.1	124	127	807	624 (77%)
JHotDraw	7.4.1	575	692	6414	2480 (39%)

a simple and clear design (being it developed for educational purposes).

- **JFreechart**[3] is a Java tool supporting the visualization of data charts (e.g., scatter plots and histograms). We included two versions of this software. As reported in Table 1, both these versions contain almost the 80% of methods with lead comments. This suggests an extensive use of comments, which is the main reason why we decided to include this software in the dataset.

- **JHotDraw**[4] is a framework for technical and structured graphics. Even if the source code of JHotDraw is scarcely commented (see Table 1), it is well-known in the software maintenance community due to its good Object-Oriented design (Scanniello et al., 2010).

In Figure 1, we report the implementation and the lead comment of `setTickLabelsVisible` (extracted from JFreeChart ver. 0.7.1, included in our dataset). According to our definition of coherence, we can assert that this method is coherent. On the other hand, the `save` method reported in Figure 2 provides a very poor and inadequate description of the design intent of the method, thus reflecting a lack of coherence with the underlying implementation.

Three annotators were involved in the dataset creation process. Two of them hold a Bachelor degree in Computer Science, and have very similar technical backgrounds. On the other hand, the third annotator can be considered more experienced than the other two since he holds a Master degree in Computer Science. We distributed the effort among the annotators so that each software

[3] `www.jfree.org/jfreechart/`

[4] `www.jhotdraw.org/`

```
/**
 * Sets the flag that determines whether or not
 * the tick labels are visible.
 * Registered listeners are notified of a
 * general change to the axis.
 *
 * @param flag The flag to set.
 */
public void setTickLabelsVisible(boolean flag) {
    if (flag!=tickLabelsVisible) {
        tickLabelsVisible = flag;
        notifyListeners(new AxisChangeEvent(this));}
}
```

Figure 1: The lead comment and the implementation of the method `setTickLabelsVisible` of JFreeChart 0.7.1

```
// GEN-FIRST:event_save
// Code for dispatching events from components
// to event handlers.
private void save(java.awt.event.ActionEvent evt) {
try {
  String methodName = getParameter("datawrite");
  if (methodName.indexOf('(') > 0) {
        methodName = methodName.substring(0,
                methodName.indexOf('(') - 1); }
  JSObject win = JSObject.getWindow(this);
  Object result = win.call(methodName, new
           Object[]{getData()});
} catch (Throwable t) {
  TextFigure tf = new TextFigure("Fehler:␣" + t);
  AffineTransform tx = new AffineTransform();
  tx.translate(10, 20);
  tf.transform(tx);
  getDrawing().add(tf); }
}
```

Figure 2: Non-Coherent method in JHotDraw 7.4.1

would be separately evaluated by at least two annotators. This allowed us to have multiple judgements for each method in the dataset, and to calculate the rate of agreement among annotators.

4 Annotation Assessment

The whole dataset creation process occurred from January, 15^{th} 2014 to June, 20^{th} 2014, for a total of 800 man-hours. This gives an estimation of the effort required to conduct the study presented in this paper, and provides an indication to the researcher interested in extending our dataset.

The annotators provided indications on the coherence of methods by assigning them one out of three following possible values: *Non-Coherent*, *Don't Know*, and *Coherent*.

In this scenario, we use the *kappa index* (Cohen, 1960) to obtain an assessment of the agreement among annotators, thus estimating the reliability of their evaluations. In fact, if annotators agree on a large number of methods, we can conclude

that their annotations are reliable. The kappa index is designed for categorical judgments and refers the agreement rate calculation to the rate of chance agreement:

$$kappa = \frac{p_o - p_c}{1 - p_c}, \quad (1)$$

p_o is the observed probability of agreement, while p_c is the chance probability of agreement. Both probabilities are estimated by the corresponding frequencies. By a simple algebraic manipulation, Equation 1 can be written as:

$$kappa = 1 - \frac{q_o}{q_c}, \quad (2)$$

where $q_o = 1 - p_o$ and $q_c = 1 - p_c$ and correspond to the observed and the chance probabilities of *disagreement*, respectively. Usually the index assumes values in $]0, 1]$[5], as it can be expected that the observed disagreement is less likely than chance. A null value signals that observed disagreement is exactly as likely as chance, while the kappa index assumes negative values in the unwanted case where disagreement is more likely than chance. Perfect agreement corresponds to $k = 1$. Values greater than 0.80 are usually considered as a cue of good agreement. Values in the interval $[0.67, 0.80]$ are considered acceptable (Cohen, 1960).

The classical formulation of the kappa index considers a binary classification problem (e.g., *Non-Coherent* or *Coherent*). However in our case, the neutral judgement (i.e., *Don't know*) is also allowed. Therefore, possible disagreements include the case where one of the two answers is the neutral one. In this case, it is possible to differently weigh the possible disagreements among annotators. In fact, disagreements due to the neutral answers are less serious than disagreements where judgments are totally divergent (i.e., *Coherent* and *Non-Coherent*, in our case). To this end, Cohen (Cohen, 1968) presents a variant of the kappa index, where in case of a disagreement, different weights can be applied. In case the same weight is assigned to all possible disagreement combinations, the original (unweighted) formulation is obtained. The formulation of the *Weighted Kappa* (WK) is the one in the equation (2), but for the computation of q_o and q_c the contributions are weighted according to the importance given to the corresponding disagreement cases. By contrast, we refer to the original formulation of the kappa index as *Unweighted Kappa* (UK). We assign to the *Don't know* response a weight that is half the weight assigned to the *Not-Coherent* (or *Coherent*) one. This is the same schema reported by Cohen (Cohen, 1968). Weighted and Unweighted kappa indexes are reported in Table 2.

[5]The notation means that 1 is included in the interval, while 0 is not.

Table 2: Agreement Rate of Judges as computed by the Cohen's Kappa Index.

System	UK	WK
CoffeeMaker	0.913	0.913
JFreeChart 6.0	0.939	0.918
JFreeChart 7.1	0.983	0.977
JHotDraw	0.824	0.684

The agreement between annotators is good on the first three systems, and acceptable for JHotDraw. However, on this system the difference between the values for UK and WK is large, thus providing a more accurate indication on the agreement of the evaluations on this software.

At the end of the first step of the dataset creation process, the number of methods on which annotators did not agree was 302, corresponding to the 8.3% of the total number of methods from all the systems in the dataset. Most of these methods are those in JHotDraw, as suggested by the kappa index values (see Table 2). These methods were reviewed by two of the authors. An agreement was reached on all of these methods, which were then included in the dataset. The total number of methods in the dataset is reported in Table 3 (i.e., $2,883$).

5 Conclusions and future work

In this paper, we have presented the early steps of our research on the coherence between the lead comment of methods and their implementations. In particular, we have provided a description of the problem settings, along with the experimental protocol defined to create our dataset. We made it publicly available on the web. We also sketched the results of quantitative analysis conducted on a codebase of $3,636$ methods, gathered from 4 different open-source systems written in Java.

There could be many possible future directions for our research. For example, it would be interesting to conduct an empirical study to investigate

Table 3: Descriptive Statistics of the Dataset: C stands for Coherent, NC for Non-Coherent.

System	C	NC	Total	Don't Know (Not included)
CoffeeMaker	27	20	**47**	0
JFreeChart 6.0	406	55	**461**	24
JFreeChart 7.1	520	68	**588**	36
JHotDraw	762	1025	**1787**	693
Total	**1715**	**1168**	**2883**	

the effect of maintenance operations on the coherence for multiple versions of the same system. As a step forward in this future research direction, we already included in the dataset two versions of the JFreeChart system. Our results and those by Fluri et al. (Fluri et al., 2007) represent a viable starting point. Finally, we would like to exploit the collected data as an evaluation set to assess the performance of approaches able to discern method coherence.

References

Ron Artstein and Massimo Poesio. 2008. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, December.

J. Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

J. Cohen. 1968. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*.

A. Corazza, S. Di Martino, and V. Maggio. 2012. LINSEN: An efficient approach to split identifiers and expand abbreviations. In *Proceedings of International Conference on Software Maintenance*, pages 233–242. IEEE Computer Society.

A. Corazza, V. Maggio, and G. Scanniello. 2015. On the coherence between comments and implementations in source code. In *41st Euromicro Conference on Software Engineering and Advanced Applications, EUROMICRO-SEAA 2015, Madeira, Portugal, August 26-28, 2015*, pages 76–83. IEEE Computer Society.

A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello. 2016. Weighing lexical information for software clustering in the context of architecture recovery. *Empirical Software Engineering*, 21(1):72–103.

Barbara Di Eugenio and Michael Glass. 2004. The Kappa statistic: a second look. *Computational Linguistics*, 30(1).

B. Fluri, M. Wursch, and H.C. Gall. 2007. Do code and comments co-evolve? on the relation between source code and comment changes. In *Proceedings of the Working Conference on Reverse Engineering*, pages 70–79. IEEE Computer Society.

D Lawrie, D Binkley, and C Morrell. 2010. Normalizing Source Code Vocabulary. In *Proceedings of Working Conference on Reverse Engineering*, pages 3–12. IEEE Computer Society.

Yann Mathet, Antoine Widlöcher, and Jean-Philippe Métivier. 2015. The unified and holistic method gamma γ for inter-annotator agreement measure and alignment. *Computational Linguistics*, 41(3):437–479, September.

F. Salviulo and G. Scanniello. 2014. Dealing with identifiers and comments in source code comprehension and maintenance: Results from an ethnographically-informed study with students and professionals. In *Proceedings of International Conference on Evaluation and Assessment in Software Engineering*, pages 423–432. ACM Press.

G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico. 2010. Using the kleinberg algorithm and vector space model for software system clustering. In *Proceedings of International Conference on Program Comprehension*, pages 180–189. IEEE Computer Society.

C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering*. Computer Science. Springer.