How Good is your Comment?
A study of Comments in Java Programs

Dorsaf Haouari, Houari Sahraoui, Philippe Langlais
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, Canada
Email: {haouarid, sahraouh, felipe}@iro.umontreal.ca

***Abstract*—Comments are very useful to developers during maintenance tasks and are useful as well to help structuring a code at development time. They convey useful information about the system functionalities as well as the state of mind of a developer. Comments in code have been the focus of several studies, but none of them was targeted at analyzing commenting habits precisely. In this paper, we present an empirical study which analyzes existing comments in different open source Java projects. We study comments from both a quantitative and a qualitative point of view. We propose a taxonomy of comments that we used for conducting our analysis.**

I. Introduction

Software maintenance is performed through complex tasks which consume the majority of software development resources [11]. Among the activities involved in these tasks, program understanding represents certainly the most central and crucial one. Indeed, many studies showed that due to lack of up-to-date documentation, more or less half of the effort of maintenance is dedicated to software analysis and understanding (see for example, the studies reported in [9] and [8]).

In order to understand a program, developers usually read existing comments in the source code. These comments are very important artefacts used during development and maintenance tasks [2]. They explain the used code structures and express developers' thoughts and notes in (more or less) natural language. Therefore, they have the potential of increasing the understandability of a program.

Much work targeted comments from, among others, the perspectives of program comprehension, maintenance, and reliability. These studies generally do not consider the comment content. The very few that analyze the content typically target specific software and very specific tasks. For example, the study presented in [7] focuses on operating system code in order to identify reliability problems that could be solved by existing tools.

We believe that studying comment content for program comprehension and documentation would bring useful information on how to benefit from these comments. Consequently, we propose to empirically analyze comment content in samples of programs to understand the kind of information conveyed by comments. More precisely, we are first interested in knowing which program construct types are documented more often than others. This allows to derive general trends on commenting habits that could be exploited in program documentation.

The second aspect that interests us is the nature and the relevance of the comments. As comments are used for different purposes (documentation, communication, personal notes, etc.), it is important to know how often comments are written for a particular purpose. Moreover, the presence of a comment does not necessarily mean that this comment is useful and could be used for documentation or comprehension purpose.

In this context, we conducted an empirical study in order to analyze existing comments in Java programs. Our study involved 49 programmer subjects (after the data validation, the data of 39 of them were studied) and three open source projects. Subjects analyzed the content of a random sample of comments. To this end, we defined a taxonomy of comments to guide this analysis. Our study showed that programmers comment some constructs more often than others. In the majority of cases, comments are intended to explain the code that follows them. The second more widely used category of comments are dedicated to communication between programmers and personal notes (we call them working comments).

The rest of this paper is structured as follows. Section II summarizes the related work on comment analysis. Our first study is concerned with the quantitative aspects of comments and is described in Section III. The results and discussion of the qualitative aspects are presented in Section IV. The obtained results are compared in Section V to the subjective evaluation results provided by the subjects. In Section VI, we highlight the possible threats to the validity of our study. Finally, Section VII gives conclusive remarks.

II. Related Work

Several studies targeted comments in code. Some of them investigated the usefulness of comments for program understanding. For instance, Souza and al. [2] conducted a

survey in order to know which artifacts are the most useful during maintenance tasks. They found for two different programming paradigms, that after the source code, the comments are the most important artifacts used to understand a system; more important than design documents for example.

Other works studied the evolution of comments and source code [5] [3]. Malik and al. [5], for example, were interested in understanding the rationale for updating a function's comment. They found that the characteristic of the change is the most influential dimension in explaining the comment update phenomenon.

Another category of contributions exploited the contained information in comments for various maintenance tasks. For example, Tan and al. [13] studied the consistency between comments and code. When an inconsistency is found, it is classified as a bug (the code is wrong) or as a bad comment (the comment is wrong).

The previous categories of studies were not concerned with the content and relevance of comments. To understand what the issues mentioned in comments are, Padioleau and al. [7] studied comments in operating system code. They found that many of the problems mentioned in comments could be easily solved by existing (or easy to develop) tools. Ying and al. [15] focused particularly on categorizing task comment contents.

Schreck and al. [10] defined several metrics to assess comment quality features. The metrics proposed cannot evaluate the semantic consistency between code and comments.

We are concerned with the ability of comments to document the code. In this context, many tools were designed to facilitate the generation of documentation accompanying a code. Tools like `javadoc` [1] and `Doxygen` [2] are among popular ones. This kind of tools which deals with the extraction of comments from existing programs can be seen as posterior approaches [6] to code understanding. Other tools are dedicated to prevenient [6] scenarios, where a developer documents the program before, or side by side with coding. Tools such as "Verbal source code descriptor" [14] or "Commenting by voice" [1] help developers to comment their code by voice, at the same time as they write it.

In a different but related context, Sridhara and al. [12] proposed an automatic approach for generating comments from source code for Java methods. They identify the important code statements in a method to be commented and transform them into natural language by applying a text generation technique. Similarly, Haiduc and al. [4] summarize source code and generate, mainly, extractive summaries for source code constructs. They extract lexical information from identifiers in code and comments, then, apply text retrieval techniques in order to determine the most relevant terms which will form the summary.

[1]http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[2]http://www.stack.nl/ dimitri/doxygen/

Table I
MAIN CHARACTERISTICS OF THE PROJECTS USED IN THIS STUDY.

	`DrJava`	`SHome3D`	`jPlayMan`
|packages|	14	9	9
|classes + interfaces|	591	180	40
|methods|	9 200	3 600	664
|lines of codes (LOC)|	145 511	66 484	20 869
|lines of comments (CLOC)|	50 666	16 509	5 814
|nb of comments (NCOM)|	14 954	5 636	1 357
comment density			
CLOC / LOC	34.8%	24.8%	27.9%
NCOM / LOC	10.3%	8.5%	6.5%

All these studies contribute to show the importance of comments in code, and the diversity of information they convey. Paradoxically, we are not aware of a study which analyses comments at large and by themselves, both from a quantitative and a qualitative perspective. This work is an attempt to fill this gap.

III. Study 1: Comment Distribution and Frequencies

A. Setting

Our goal is to study how comments are distributed among the code constructs, and what the most often commented constructs are.

Our study of comments harvested three open source projects written in Java by more than 50 developers. These projects are:

- `DrJava`[3], a lightweight development environment for writing Java programs.
- `SweetHome3D`[4], an interior design application.
- `jPlayMan`[5], an application for the creation and management of music playlists.

We selected these projects with the goal of having different projects and development team sizes. The size metrics of the projects we studied are reported in Table I.

The size of projects varies from 40 to 591 for classes and from 20 to 145 KLOC. For the three projects, more or less 30% of lines of code are comments, which clearly indicates that commenting is not anecdotal in those projects. The largest program, *i.e.*, `DrJava`, is the most commented with one comment for ten lines of code (NCOM/LOC = 10.3%). This is not surprising considering the large number of developers involved in the project. The other two projects have slightly fewer comments per line of code.

To determine the distribution and the frequencies of the comments with respect to the Java construct types, we parsed the projects' Java code using a parser generated from `sableCC` [6]. From the strict location standpoint, each

[3]http://www.drjava.org/

[4]http://www.sweethome3d.com/

[5]http://jplayman.sourceforge.net/

[6]http://sablecc.org/

comment was associated to the construct type contained in the line that follows it. As this analysis is fully automated, we did not need to work on a limited sample and considered all the comments in the three projects.

B. Results and Discussion

Table II reports the distribution of comments represented by the ratio between comments preceding a given construct (we considered many of them) and the total number of comments observed. It is important to note that it is not because a comment precedes a construct (such as a method) that it does concern this construct. In this first study, we made this reasonable assumption. We will come back to this point in our second study.

Table II
DISTRIBUTION OF COMMENTS OVER THE CONSTRUCT TYPES FOR THE THREE STUDIED PROJECTS.

construct type	DrJava	SHome3D	jPlayMan	avg.
package declaration	3.9	3.2	3	3.4
import declaration	0.1	0	0	0.03
class declaration	4.6	5.2	5.2	5
interface declaration	0.6	0.4	0.2	0.4
field declaration	10.3	1.1	3.5	5
constant declaration	2.2	0	0	0.7
method	27.2	36.3	36.6	33.4
abstract method	4.7	2.2	0.7	2.5
constructor	3.1	3.3	4.7	3.7
local variable	8.8	13.8	14.9	12.5
assignment	3.4	4.4	4.6	4.1
method invocation	17.5	19.5	10.1	15.7
for	0.6	1.7	1.4	1.2
while	0.4	0	0.2	0.2
if	3.5	4	6.1	4.5
return	1.3	1	2.2	1.5
tryCatch	0.8	0	0.3	0.4
enum	0.1	1.4	0.8	0.8
other	6.9	2.5	5.5	5

We observe three main trends from Table II.

- The greatest proportion of comments (33.4%) precedes methods. This was somehow expected since there are comparatively many more methods than say, classes or interfaces.
- Comments preceding local variable declarations and method invocations represent 12.5% and 15.7% of comments respectively. Here again, variable declarations and method invocations are very frequent constructs in Java programs, which explains these second and third positions. Still, the portion of comments dedicated to method invocations somehow surprised us.
- The forth type of constructs that gathers the most comments is class declaration (5%), although this construct is less frequent than for example import declarations that totalize only 0.03% of the comments.

The absolute distribution of comments gives a fast picture of where the comments are located in the code, but the figures are clearly biased toward the frequency of the constructs considered: comments are most likely to precede frequent constructs. Still, it captures to some extent some commenting habits. For instance, although methods are less numerous (12 501) than method invocations (34 365), they are more often preceded by a comment than the latter. Also, such a distribution shows which parts of codes are not often commented.

In order to balance these observations, we computed the percentage of a given construct, which is preceded by a comment. We call the resulting frequencies the commenting rate distribution. It is reported in Table III.

Table III
COMMENTING RATE PER CONSTRUCT TYPE(%).

construct type	DrJava	SHome3D	jPlayM	avg.
package declaration	98.8	100	100	99.6
import declaration	0.4	0	0	0.1
class declaration	84.6	98.3	100	94.3
interface declaration	97.9	100	100	99.3
field declaration	54.1	3.5	13.3	23.6
constant declaration	87.2	0	0	29.1
method	48.6	58.9	75.9	61.1
abstract method	85.6	95.4	100	93.7
constructor	61.3	58.8	80	66.7
local variable	18	18	15.4	17.1
assignment	8.3	8.5	9	8.6
method invocation	11.1	12.1	7.6	10.3
for	11.3	16.4	11.4	13
while	21.8	2.3	12	12
if	9.7	6	6.6	7.4
return	4.9	3.3	4.9	4.4
tryCatch	5	0	1.3	2.1
enum	25	14.6	26.2	21.9

As far as comments are concerned, we can distinguish three kinds of constructs:

- Abstract methods, class declarations, interface declarations and package declarations are almost always preceded by comments (93.7% to 99.6% of the cases on average over projects).
- Methods and constructors are preceded by comments in 61.1% and 66.7% of the cases respectively. The fact that constructors do not get commented much more often comes somehow at a surprise. A possible explanation is that developers can consider constructors as self documented since they perform a very specific task.
- Expectedly, the import declarations are the less commented constructs (0.1%), as are to a lesser degree try-catch idioms (2.1%).

We also observe in Table III that the commenting rate of several constructs varies drastically from one project to another. This is, for instance, the case of the field declarations which are commented half the time in the DrJava project, but in a very few occasions in the other two. This indicates that although there is a general consensus on commenting some constructs, there are some "cultural" differences on the others.

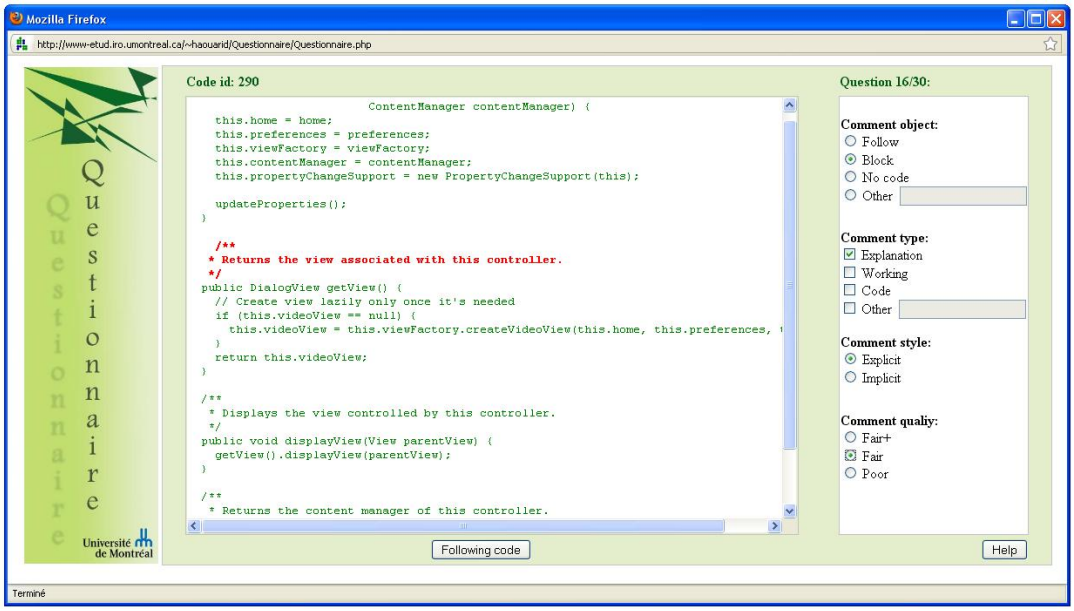

Figure 1. An Example of the Web Form for a Comment.

If those figures (absolute and relative ones) help to portray the global commenting habits of programmers, they do not characterize the quality of the comments, neither do they tell us if a comment is related to the constructs in its vicinity. To understand these aspects, we performed a second study, which is the purpose of the next section.

IV. Study 2: Comment Content and Relevance

A. Setting and Methodology

Unlike the distribution and the frequencies, the study of the comment content and relevance cannot be fully automated. For this reason, we decided to use programmer subjects in this study.

We used the stratified sampling technique for gathering a representative set of code fragments (comments with adjacent lines of code). We divided the comments of the three projects of Section III into subgroups (strata) corresponding to the constructs types. Then, we randomly selected from each subgroup a number of comments proportional to the size of the subgroup according to the empirical distribution of comments over the different constructs observed in Section III. Each code fragment was transformed into a web form where a participant could inspect the code and fill a form about the content and the relevance of the comment (see Figure 1). We decided to present for each code fragment the chosen comment in red together with the 10 lines that precede it and the 20 lines that follow it. This choice emerged after examining many comments to decide what is the appropriate window that is necessary for the subjects to provide the required information about the comment. Another motivation is that the size of the code fragment allows the subjects to perform the annotation task without having to scroll too much over the code.

For the subject selection, we invited programmers of various records (academics, industrials, graduate students) by email to participate to the comment analysis effort. Among these persons, 49 accepted to participate. All the subjects are experienced in object-oriented programming and particularly in Java. To cross-validate the subjects' answers, we randomly created groups of three subjects and assigned to them 30 code fragments to classify. This excluded one of the participants (16 groups of three subjects). In total, 480 comments were included in the sample.

A session of one hour was scheduled for each group. Sessions took place in our laboratory. A session typically started by a mini-tutorial presenting the annotation task with examples and the web application to use. After this, the participants were mainly left to themselves until they completed their tasks. At the end, the subjects filled another questionnaire in order for us to collect information about their profiles and commenting habits.

For each code fragment, we collected the information from the three subjects and used a majority voting system to decide for the final values. In the very few cases when no majority was found (three different answers), we excluded the comment from the sample. We also excluded from our corpus the annotation produced by subjects who did not complete their task. After gathering and validating the data, we obtained complete answers for 407 comments.

Table IV
EXAMPLES OF COMMENTS WITH THEIR CLASSIFICATION

`// if the document was an auxiliary file, remove it from the list`
`if (doc.isAuxiliaryFile())`
`  removeAuxiliaryFile(doc);`
	`follow`
	`explanation`
	`explicit`
	`fair`

`/** A state variable indicating whether the class path has changed.`
`Reset to false by resetInteractions. */`
`private volatile boolean classPathChanged = false;`
	`follow`
	`explanation`
	`explicit`
	`fair+`

`_tokens = new TokenList();`
`$cursor = _tokens.getIterator();`
`// we should be pointing to the head of the list`
`_cursor.setBlockOffset(0);`
	`follow`
	`explanation`
	`implicit`
	`poor`

`ServiceManager.setServiceManagerStub(`
`new StandaloneServiceManager(applet.getAppletContext(), codeBase, this.name));`
`// Caution: setting a new service manager stub won't replace the existing one`
	`other`: precedent
	`working`

`// _mainFrame.hourglassOff();`
`disableChangeListeners();`
`_mainFrame.toFront();`
	`nocode`
	`code`

B. Taxonomy of Comment Content and Relevance

Before running the study, an important issue was to determine how to characterize the content and the relevance of the comments in order to guide the subjects in their classification (form to fill for each code fragment). As, to our best knowledge, there is no work proposing such a general characterization, we first designed a taxonomy. We conducted a careful inspection of a random sample of comments and found a number of dimensions we feel are important for analyzing the content and relevance of comments. The resulting taxonomy is as follows.

- **Object of the comment** (*object*). This dimension characterizes the constructs in a code that are concerned by a comment. Intuitively, developers tend to put comments before the construct they want to describe. Therefore, we propose the following categories for characterizing the object of the comment:
  - `follow` indicates that the comment concerns the following instruction,
  - `block` stands for cases where the comment concerns the following block of instructions,
  - `nocode` corresponds to cases where a comment concerns no code in its vicinity, and
  - `other` indicates any other situation such as the code commented precedes the comment.
- **Comment type** (*type*). A comment type can be:
  - a code explanation (`explanation`), if it describes the functionalities of the related code,
  - a working comment (`working`), if it describes future tasks to be done (*e.g.* TODO items) or eventually some code, but without supplying information about its functionalities (*e.g.* preconditions in loops),
  - a commented code (`code`); old codes are often commented instead of being removed, or
  - any other type (`other`) such as licensing and credit comments.

  It has to be noted that a comment can be labeled by several type categories. It is, for example, the case where a first part of a comment explains the following construct (`explanation`) and the other part describes an action to perform in the future (`working`).
- **Style** (*style*). This dimension is specific to explicative comments (*type*≡`explanation`). We distinguish two categories `explicit` and `implicit` which respectively characterizes situations where the comment is written in terms of the instruction keywords and identifiers or in more abstract terms (see examples in Table IV).
- **Quality** (*quality*). This dimension is specific to explicative comments as well, and involves three categories:
  - `fair+` which designates comments that are presenting the functionalities of the related code, as well as other information,
  - `fair` which indicates that the functionality of the code being commented is adequately described, and
  - `poor` where only a few or none of the functionalities of the code are described.

Table IV illustrates a few comments and their classification using the proposed taxonomy. For instance, the first example has been labeled as an explanation comment of an explicit type and which adequately comments the instruction following it. The fourth example is a working comment whose object is the code construct preceding it. Finally, the last example shows a case of commented code.

Table V
DISTRIBUTION OF THE CATEGORIES OF OUR TAXONOMY OVER THE PROCESSED COMMENTS (%)

object	follow	22
	block	51
	nocode	17
	other	10
type	explanation	71
	working	17.9
	code	9.3
	other	8.8
style	explicit	80
	implicit	20
quality	fair+	15
	fair	56
	poor	29

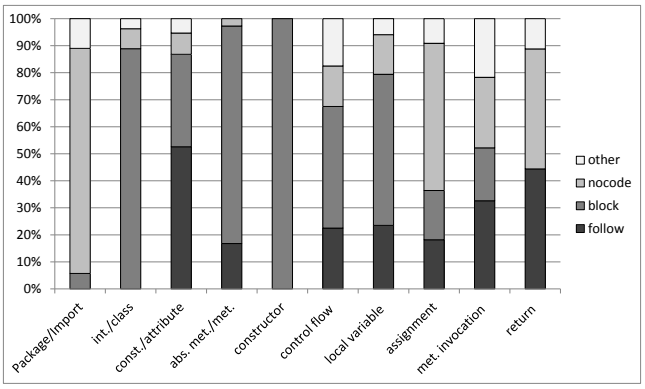

Figure 2. Results for Comment Object.

C. Analysis

Table V reports the frequency of each category of our taxonomy for the processed comments[7]. We observe that 73% (22%+51%) of the comments are dedicated to the following constructs (either a single instruction or a block of instructions). This confirms the intuition we discussed earlier. Also, we observe that the majority of comments are explicative (71%). Slightly less than 20% of them are labeled as working comments. It is also noticeable that most of the time, explicative comments are explicit (80%). Finally, more than two thirds (56%+15%) have a good quality, although 15% of them are too descriptive. All this concurs to indicate that the comments in the projects we analyzed are indeed useful comments.

We further analyzed those trends by distinguishing the nature of the constructs being commented. The distribution of each category in our taxonomy for all the constructs we considered are reported in Table VI. Globally, we can observe that commenting practices varies a lot with the type of constructs being commented.

1) Objects of Comments: Figure 2 shows a chart that summarizes the results obtained for the object of comments for each type of constructs. Comments before classes/interfaces and member declarations concern in general the constructs they precede. This is particularly the case of constructors with 100% of the comment having as object the following block of instructions. In many cases, a single comment is written for a set of attributes and constant declarations rather than one comment per declaration. This explains the large proportion of following block for these constructs (34.2%). A good illustration of this phenomenon is reported in Comment 1 .

Comment 1 (`block+explanation+explicit+fair`)

```
 /** Color for highlighting find results. */
public static final ColorOption
FIND_RESULTS_COLOR1 =
new ColorOption("find.results.color1", new
Color(0xFF, 0x99, 0x33));
public static final ColorOption
FIND_RESULTS_COLOR2 =
new ColorOption("find.results.color2", new
Color(0x30, 0xC9, 0x96));
```

Conversely, comments that precede assignments, method invocations, and return statements are less related with those constructs. Many of them are old code transformed into comments or working comments such as in Comment 2.

Comment 2 (`nocode+working`)

```
 // TODO, maybe: remove playlist config file
// from file system as well. Maybe provide
// this as a user option (pop-up asking for
// confirmation or something). But for now,
// not.
nonFatalConfigError = true;
```

2) Types of Comments: The second important aspect that we studied is the type of comments. The data we collected from the subjects' tasks is presented in Figure 3. Unlike the chart of Figure 2, the cumulative percentages exceed 100% because of possible multiple choices. Explanation comments are more frequent than working ones for all but package declarations, imports and assignments. Most comments before packages are copyright, authorship, and credit information. Import statements are generally preceded by working comments, such as in Comment 3.

Comment 3 (`nocode+working`)

```
 import java.awt.*;
// TODO: Check synchronization.
import java.util.Vector;
```

[7]For the type category, the sum of values exceeds 100% because of the possibility of selecting multiple choices.

Table VI
DISTRIBUTION OF THE CATEGORIES OF COMMENTS OVER THE CONSTRUCTS BEING COMMENTED

		object				*type*				*style*		*quality*		
		follow	block	nocode	other	explanation	working	code	other	explicit	implicit	fair+	fair	poor
packages/import decl.	(freq)	0	5.7	83.3	11	0	27.8	5.5	66.7					
interfaces/classes decl.	(freq)	0	88.9	7.4	3.7	81.5	25.9	3.7	14.8	63.6	36.4	27.3	50	22.7
constants/attributes decl.	(freq)	52.6	34.2	7.9	5.3	76.3	21.1	0	5.3	62.1	37.9	6.9	51.7	41.4
abstract/class methods	(freq)	16.8	80.5	2.7	0	93.8	8	0	4.4	87.7	12.3	17.9	62.3	19.8
constructors	(freq)	0	100	0	0	100	0	0	0	83.3	16.7	16.7	75	8.3
control flow	(freq)	22.5	45	15	17.5	67.5	20	15	0	74.1	25.9	11.2	44.4	44.4
local variables	(freq)	23.5	55.9	14.7	5.9	76.5	20.6	14.7	2.9	76.9	23.1	3.9	61.5	34.6
assignments	(freq)	18.2	18.2	54.5	9.1	18.2	45.4	36.4	0	100	0	0	50	50
method invocation	(freq)	32.6	19.6	26.1	21.7	52.2	28.3	21.7	6.5	83.3	16.7	12.5	50	37.5
return	(freq)	44.4	0	44.4	11.2	44.4	11.2	33.3	11.1	0.5	0.5	0.25	0.5	0.25

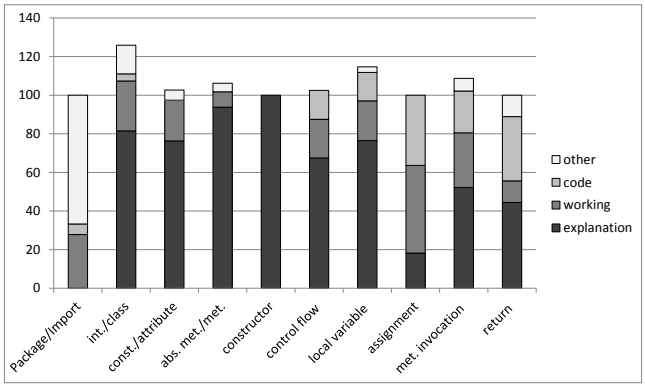

Figure 3. Results for Comment Type

There are surprisingly very few explanation comments before assignments. We found that comments located before assignments discuss other regions of the code concerned directly or indirectly by the variable. Conversely, all comments before constructors are explanation ones. This proportion should be put into the perspective that our sample includes only 3.7% of comments linked to constructors. Still, the dozen of comments considered all in a way or another the constructors.

Finally, as might be expected, comments containing old code are essentially in method bodies (control structures 15%, assignments 36%, invocations 22%, etc.). Old class (3%) and member (0%) declarations that are not used are usually removed rather than commented when changes occur.

3) Style of Comments: As mentioned earlier, the subject analyzed the style (`explicit` vs. `implicit`) only for explanation comments. The results of this analysis are presented in the chart of Figure 4.

An interesting finding is that method declarations, including abstract methods and constructors are generally explicitly explained. When a method returns a value, the associated comment often describes this returned value as in Comment 4.

Comment 4 (`block+explanation+explicit+fair`)

```
/** Returns the current build directory in
the project profile. */
private File _getBuildDir() {...}
```

For void methods, the comment typically explains the task performed by the method, as in Comment 5.

Comment 5 (`follow+explanation+explicit+fair`)

```
/** Displays this panel in a dialog box. */
public void displayView(View parentView)
{...}
```

There is a large portion of comments before class, attributes, and constant declarations that are implicit (more than a third). Examples of implicit comments are those that explain where the attributes are used, but not how, as in Comment 6.

Comment 6 (`follow+explanation+implicit+poor`)

```
// used for playlist searches
ConditionalPlaylist ownerPlaylist = null;
```

In the other cases however, classes, attributes, and constants are explicitly commented, as for the boolean attribute in Comment 7.

Comment 7 (`follow+explanation+explicit+fair`)

```
/** Edit mode if true. */
protected boolean _editMode = false;
```

A final finding is that the quarter of control structure comments are implicit. When examining these comments,

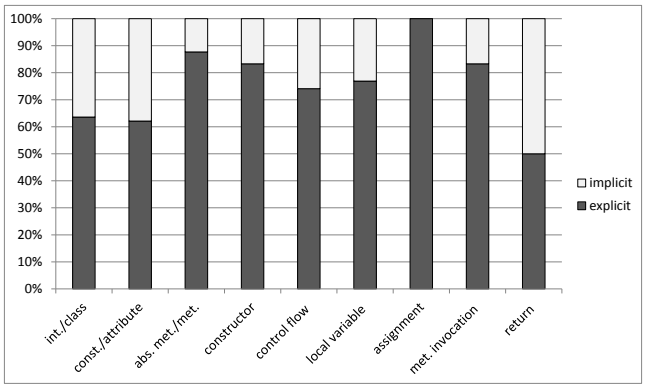

Figure 4. Results for Comment Style.

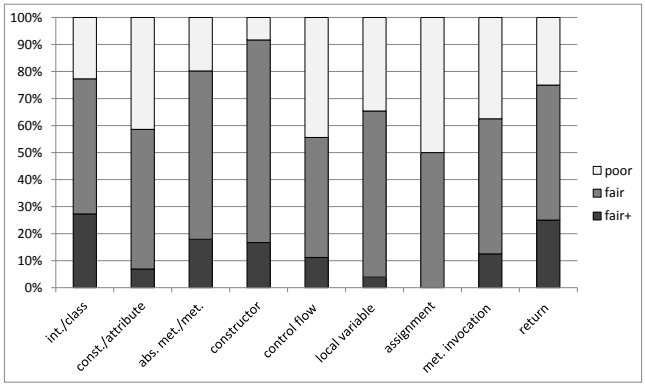

Figure 5. Results for Comment Quality

we noticed that the test condition is explained, but not the task performed in the loop or the conditional block. See for instance Comment 8.

Comment 8 (other+explanation+implicit+poor)

```
if (im.keys()!=null)//keys() may return null
{...}
```

4) Quality of Comments: The final aspect that we studied is the quality of explanation comments. The results are reported in Figure 5.

The first thing that caught our attention is the assignment paradox. We previously mentioned that 100% of the assignment comments were explicit. However, a closer inspection reveals that half of them provide poor explanations. Control structures have also many poor comments (44%). At the same time, method and constructor declarations are often very well explained.

More generally, there are more poor comments in method bodies than in class/member declarations with the exception of constants/attributes. In those cases, the comment is very short and does not bring additional information than the

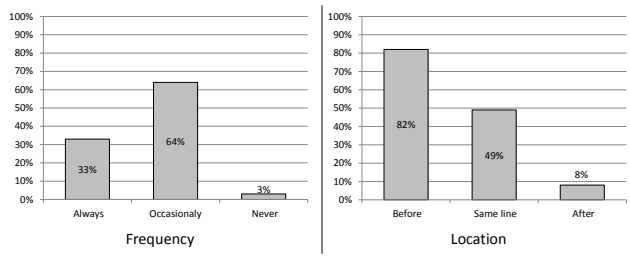

Figure 6. Frequency and Location of Comments

attribute/constant name. Comments 9 and 10 are interesting examples of these types of comments.

Comment 9 (follow+explanation+explicit+poor)

```
 /** Extra class path. */
public static final VectorOption<File>
EXTRA_CLASSPATH = new ClassPathOption()
                  .evaluate("extra.classpath");
```

Comment 10 (follow+explanation+implicit+poor)

```
 /** Frame state. */
protected FrameState _lastState = null;
```

The last observation worth mentioning is that a very few comments are too descriptive. Although it is difficult to put a clear separation between descriptive and too descriptive comments, long comments were generally considered by the subjects as too descriptive. An example of such a comment is shown in Comment 11.

Comment 11 (block+explanation+explicit+fair+)

```
 /**
* A Standard Playlist may have multiple
* instances of the same song, and the Wrapper
* class allows these to be placed in the
* model.
* This method locates all of these wrapped
* instances of a particular song
* and returns all of them in a Vector
* @param song
* @return a Vector containing all the wrapped
* instances of the supplied song, sorted in
* order from least to greates index in the
* Playlist
*/
private Vector<Wrapper>
getAllWrappedInstancesOfSong
(AbstractSongInfo song){...}
```

V. Subjective Evaluation

In addition to their classification tasks the subjects had to fill a questionnaire. This questionnaire contained three types

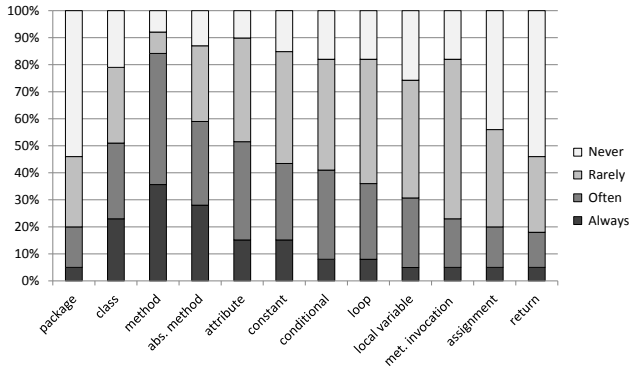

Figure 7. Commented Constructs

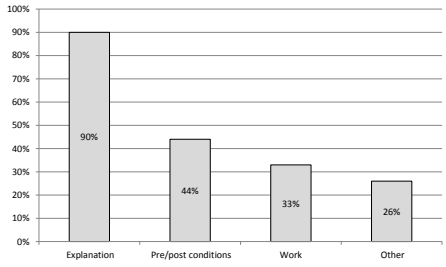

Figure 8. Type of Comments

of questions about the subject: (1) programming skills, (2) commenting habits, and (3) feedback on their participation. We will discuss the first type of questions in the study validity section (Section VI). The third type of questions was intended to collect comments to improve the design for replication studies. We will also discuss this issue in Section VI. In the remainder of this section, we discuss the data collected for the question of the second type. We study these data to compare and contrast some of the results obtained from our comment sample with the commenting habits as expressed by our subjects.

The two first questions concerned the frequency and the location of comments in their own code. As shown in Figure 6-left, only one third of the surveyed subjects claim that they comment systematically their code. The others (almost two thirds) do it occasionally. When they write comments, almost every subject locates them before the commented construct (82% as indicated in Figure 6-right). This result confirmed our choice to assign comments to the following constructs. It is consistent with the results found in our comment sample (see Table V).

When asking the subjects to give frequencies with which they comment the different types of constructs, some of the answers confirmed our finding of Table III, others were different. Indeed, as shown in Figure 7, we found that methods (including constructors) are very frequently commented (more than 80% for always+often). This outcome is consistent with what we found in the studied code. Similarly, constructs that are the least commented are the same in the subjective evaluation and the comment sample (assignments, return, etc.). However, we were surprised by the low commenting rate for packages and, to a lesser degree, classes. In our sample, almost all the packages were commented (99% in average). In the subject evaluation, only 20% of the subjects selected always or often. We conjecture that since most of the comments analyzed for packages were not explanation ones, the 20% corresponds to the cases where the subjects comment the packages in term of functionalities and not for copyright or ownership.

The last question concerned the content of the comments. We asked the subjects to give the type of their comments by selecting one or more of the items: explanation, pre/post-conditions, work, and others. Here again, the results were very similar to those presented in Table V and Figure 3. As we can see from Figure 8, explanation is the most selected item (90%). Working comments were mentioned by the third of the subjects. This value is slightly bigger than the one of Table V, but closer to values obtained for many construct types in Figure 3.

More globally, the results of the subjective evaluation confirmed the observations made on the studied sample of comments.

VI. Study Validity

Internal Validity: We identified two possible threats to internal validity: maturation and diffusion of the treatments. In the case of maturation, we addressed the learning and fatigue effects by presenting the comments to subjects in random and different orders. We tried to reduce the fatigue effect also by limiting the number of code fragments per subject to 30. We are aware that this number is still high. Many subjects mentioned in their answers to the questionnaire that the task duration was a bit high. To prevent subjects from learning the treatments before hand, we gave instructions to subjects not to talk about the experiment before the end of data collection.

Construct Validity: To ensure the accuracy of comments classification by subjects, we used three opinions per comment and aggregated them by a voting mechanism. To avoid non-consensual data, we eliminated code fragments with three divergent opinions.

External Validity: For external validity threats, the selections of subjects and code fragments are possible threats. In the case of subjects, some of the subjects are students. Although they are not professional programmers, most of them have industrial experience and knowledge comparable to junior professionals. Indeed, the profile portion of the questionnaire confirmed that 59% were good programmers, *i.e.*, an experience of two to five years, and 41% were experts. Their experience was mostly on Java (69% good

and 31% expert). The selection of code fragments followed two sampling steps. For the projects selection, we considered various project sizes (20 KLOC to 145 KLOC), application domains, and team sizes (one to 55 developers). The second step concerned the fragment selection. We used a stratified rather than a pure random sampling to have a sample more representative of the comment distribution we observed. However, the fact that we eliminated non-consensual comments and those of subjects who did not finish their session might compromise the distribution representativity. We checked the final distribution and we did not find significant variations with the initial one.

VII. Conclusion

In this work, we empirically studied comment location, content, and relevance. To this end, we proposed a comment taxonomy to guide the study subjects classifying them. Our study involved 49 programmer subjects and three open source projects.

Our study covered three aspects: quantitative, qualitative, and subjective. In the quantitative step, we were interested in the distribution of comments over the program constructs and the frequency of construct commenting. The qualitative step concerned the comment object (commented constructs), type (explanation, work, etc.), style (implicit vs explicit), and quality. Finally, the subjective evaluation concerned the commenting habits of our subjects.

We found consistent results between quantitative and qualitative studies on the one hand and the subjective evaluation on the other hand. Our results showed that some constructs such as methods are regularly commented for explanation reasons. We also found that an important portion of comments is dedicated to the communication between programmers or to notes for future changes.

This study gives an interesting picture about the location and content of the comments in the code. Other replications with larger subject and comment samples are necessary to confirm and complete this picture. Additionally, our findings could help to learn how to (automatically) recognize poor comments in a program, that could be pointed out to the developer (for instance by coloring them differently). Our main motivation and future work is to use these findings to build a program documentation/summary system. Such a system will extract relevant comments using a statistical model to document or summarize some constructs. It will complete the documentation with comment generation techniques for important, but less documented constructs.

VIII. Acknowledgment

We are grateful to all the programmers who participated in this study. This work was partially funded by NSERC.

References

[1] A. B. Begel. Spoken language support for software development. Master's thesis, University of California, Berkeley, 2005.

[2] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira. Which documentation for software maintenance? *J. Braz. Comp. Soc.*, 12(3):31–44, 2006.

[3] B. Fluri, M. Würsch, E. Giger, and H. C. Gall. Analyzing the co-evolution of comments and source code. *Software Quality Control*, 17:367–394, December 2009.

[4] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus. On the use of automated text summarization techniques for summarizing source code. *Reverse Engineering, Working Conference on*, 0:35–44, 2010.

[5] H. Malik, I. Chowdhury, H.-M. Tsou, Z. M. Jiang, and A. E. Hassan. Understanding the rationale for updating a function's comment. In *ICSM*, pages 167–176, 2008.

[6] K. Nørmark. Requirements for an elucidative programming environment. In *IWPC*, pages 119–, 2000.

[7] Y. Padioleau, L. Tan, and Y. Zhou. Listening to programmers - taxonomies and characteristics of comments in operating system code. In *ICSE*, pages 331–341, 2009.

[8] S. L. Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall PTR, 2nd edition, 2001.

[9] T. M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, Inc., 1996.

[10] D. Schreck, V. Dallmeier, and T. Zimmermann. How documentation evolves over time. In *Proceedings of the 9th International Workshop on Principles of Software Evolution*, September 2007.

[11] R. C. seacord, D. Plakosh, and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[12] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 43–52, New York, NY, USA, 2010. ACM.

[13] L. Tan, D. Yuan, G. Krishna, and Y. Zhou. /*icomment: bugs or bad comments?*/. In *SOSP*, pages 145–158, 2007.

[14] S. S. Tehrani. Verbal source code descriptor. Master's thesis, The University of British Columbia, 2003.

[15] A. T. T. Ying, J. L. Wright, and S. Abrams. Source code that talks: an exploration of eclipse task comments and their implication to repository mining. In *Proceedings of the 2005 international workshop on Mining software repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.