The role of comments on Program Comprehension *

José Luís Freitas, Daniela da Cruz, and Pedro Rangel Henriques

Informatics Department
Universidade do Minho
Braga, Portugal
freitas.jlf@gmail.com, danieladacruz@di.uminho.pt, prh@di.uminho.pt

Abstract. This paper presents Darius, a tool that aims at helping the exploration of comments for Program Comprehension purposes. In particular, we present several experiments, using Darius, in order to study the relationship between comments and the Problem and Program Domain. The following questions established the basis of the study: (1) what is predominant in comments: Program Domain terms or Problem Domain terms?; (2) what is the relation between the type of domain terms used (problem or program) and the type of comment (inline, block and javadoc)?; (3) what is the relation between the type of domain terms used (problem or program) and the type of source code entity (class, method, statements, etc.) commented?
From the experiments conducted with Darius, we concluded that in the studied Java projects there is a significant amount of (inline, block and documentation) comments, and that terms of both Program and Problem Domains appear frequently inside the comments. This will allow to create the aimed bridge as a next step in this Program Comprehension project.

Keywords: program comprehension; source code analysis; comment analysis; problem domain; program domain; information retrieval

1 Introduction

Nowadays there are many methods/approaches that help software engineers and programmers to develop their product with more accuracy and with less costs. However, the maintenance of a system persists to spend too many resources. According to [3], before the early 1990s, Software Maintenance has spent almost half of the resources, and recently [4] showed that 80 to 95% of the budget given to Information Systems is for maintenance activities. Most of the problems in Software Maintenance, are due to the fact that half of the time is spent on comprehending the given system or program [5].

* This work is funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project ref. PTDC/EIA-CCO/108995/2008.

Nowadays there are areas of Software Engineering, such as Program Comprehension (PC) that are concerned with these problems; researchers on those areas work on theories, techniques and tools that help programmers and engineers to extract from a program the knowledge needed to understand it. However, most of PC tools only extract from a program its structural information but they lack the extraction of its meaning [7]. Although the structural information is important, the richness of semantic information included on source code comments, written in natural language, could give a bigger contribution for the understanding process.

However, the information included on comments should contain Problem Domain terms to attain a better understanding of the program. If comments only include terms from the Program Domain, it becomes more difficult to establish a relation between the code and its purpose.

In order to understand the nature of comments, we developed Darius[1], a tool that analyzes source code comments and provides information for Program Comprehension purposes. Taking advantage of Darius, we developed several experiments that investigated whether the terms used, when writing comments, are Problem Domain or Program Domain oriented. We address this problem by answering to the following research questions:

1. What is predominant in comments: Program Domain terms or Problem Domain terms?
2. What is the relation between the type of domain terms used (problem or program) and the type of comment (inline, block and javadoc)?
3. What is the relation between the type of domain terms used (problem or program) and the type of source code entity (class, method, statements, etc.) commented?

1.1 Structure of the paper

After a careful search and deep analysis, we provide in Section 2 some information regarding the work on comment analysis and its effect on Program Comprehension. In Section 3 we present our tool called Darius, describing in detail each one of its components. Then in Section 4, we describe two experiments, using Darius, in order to provide answers to the questions raised in this paper. We conclude this paper in Section 5, describing the lessons learned carrying out the development of Darius and the experiments; trends for future work are also pointed out.

2 Related Work

Considering Program Comprehension, different authors [10, 2] have defended the importance of Problem Domain knowledge on the understanding of a given program. The *Problem Domain* of a program is defined as a part of the world about

[1] Relative to King Darius I of Persia, the first known man to create the first bridge between Europe and Asia, on the Bosphorus strait.

which the program is concerned with solving problems [9], and as a *domain* is composed by a set of objects, the relations among them and the operators which manipulate them [2]. The Problem Domain of a store management software will include typical objects, such as *product* or *receipt*, and functions or operators that change or manipulate them such as *print the receipt* or *sell a product*.

In Program Comprehension, two major theories of understanding are accepted: *top-down* and *bottom-up*. When programmers are familiar with the Problem Domain, they tend to use a top-down comprehension process [10]. When using a top-down approach, the programmer tries to map Problem Domain concepts to their implementation on code (*Program Domain*), process which was called *concept assignment* by Biggerstaff et al. [1].

Some authors [8, 14] took advantage of Information Retrieval techniques for concept assignment, using the identifiers and comments, to establish links between code and documentation.

In [6], the authors studied whether comments and identifiers included Problem Domain terms. They created a list of terms from the Problem Domain underlying the program they were studying, and measure the frequency of terms occurrences in identifiers and comments. The results showed that half of the Problem Domain terms were included in the source code, mostly on comments. The study reflected in this paper, will assume a similar methodology, described on Section 4.

The effect of source code comments on the understanding of a program has always been a subject of study for many researchers. In his theory of comprehension, Brooks described in [2] the value of comments on the construction of a mental model of the program, specially if these include terms from the Problem Domain.

Taking this, several experiments [12, 13] were conducted to verify whether the comprehension of a program could be catalyzed by comments. To the subjects of these experiments were given two versions of the same program: one with comments and the other without. Then they had to answer a questionnaire about the program they just had analyzed. The results showed that the subjects which were given programs with comments were able to answer to more questions correctly.

3 Darius

In the last section we highlighted the work described in [6] that studied the existence of Problem Domain terms on comments and identifiers. However, the performed studies did not calculate in detail the positioning of Problem Domain information on different types of comments and on comments of different source code entities, and lack the study of the existance of Program Domain terms. As our goal is to use comment information to develop a Comment Analysis PC tool in the future, this information can be important to explore. In order to perform an initial step to fulfill this goal and to address the questions mentioned above, we developed a tool called Darius that analyzes source code comments, providing

quantitative information. The structure of Darius is shown in Figure 1. As we can see, Darius is composed by:

- a comment extractor that withdraws comments from source code files and a code associator that associates each comment with the piece of code (classified according to the source code entity type: class, method, statement, and so on) it is commenting;
- a statistics calculator that provides quantitative results regarding comments in a software project;
- a comment words analyzer which computes the frequency of words on comments, information that can be used to build a tag cloud or to identify the domain of each word;
- and a web interface, a graphical interface which can be used to visualize all the information provided by the other components.

Below, each one of these components is described in detail.

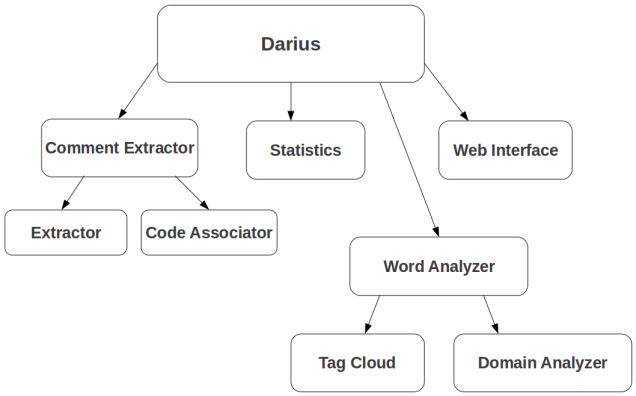

Fig. 1. Structure of Darius

3.1 Extracting and Locating Comments

The tool described in this section extracts comments from Java source code files. In the Java programming language there are three types of comments:

- InLine comments, **IC** for short, (//...)
- Block comments, **BC** for short, (/* ... */)
- JavaDoc comments, **JD** for short, (/** ... */)

These three types of comments are extracted from the source code using regular expressions. The use of regular expressions in spite of a parser, is explained by the fact that not all types of comments take part of the AST (javadoc only) extracted by a parser. Although there are approaches [11] that try to associate inline and block comments with syntactic nodes in the AST, by changing the Java grammar rules, the extraction of comments in Darius pretends to be as generic as possible for every Java source code file.

In order to discover and identigy what type of source code entity is associated with the comment, the next line after the comment is extracted too. Considering the Java programming language, Darius associates comments with *classes*, *interfaces*, *methods*, conditionals (*if*), loops (*while* and *for*) and *switches*.

One important detail of this procedure, is that the first comment of a given file is removed due to the fact that the first comment contains almost always a text related with the project's license. We came up with this conclusion after many observations.

3.2 Comment Statistics

With the information extracted as described above, Darius computes some comment statistics from a project. These statistics include:

- Number of comments of a project (global and per type);
- Percentage of comment lines per lines of code for each file, and the average from the projects;
- Average number of each type of source code entity which is commented;
- Type of comments most used (global and per source code entity).

3.3 Comment Words Analyzer

Darius has the ability of analyzing the comment individually, by analyzing the words that form it. Using regular expressions, Darius breaks the comment into words and associates them to the comment. All the words are analyzed by a stemmer[2], that reduces every word to its respective stem. Words are then stored in a table that maps its frequency (on all comments). With this information, Darius can create a tag cloud with the most frequent words, that can be visualized by the Web Interface, which will be described in the next subsection.

Apart from the tag cloud, Darius can also analyze a list of words, provided by the user and check the frequency in which the words appear in the comments, calculating the following numbers:

- Percentage and frequency of words in the list found in comments;
- Frequency of each type of comment that contains words from the list;
- Frequency of each type of source code entity commented that contains words from the list.

[2] Snowball Stemmer by Martin Porter, http://snowball.tartarus.org/

3.4 Web Interface

Darius offers a Web Interface (as shown in Figure 2) to make the user interaction easier and more appealing.

This interface allows to upload a Java software project, properly compressed. After submitting a project, the user is able to visualize all the information that the components, described above, provide.

The user can calculate and visualize information regarding the comment statistics. The user can also create a tag cloud, according to his preferences. Darius gives the possibility of creating a tag cloud for all comments or just comments associated with a particular type of source code entity; non-value words (English stop-words, words related with licenses and keywords from JavaDoc) can also be removed from the cloud. As each word is associated with the comments where it appears, we can navigate through the tag cloud and click on a word to visualize these comments (as shown in Figure 3), analyzing the context in which the word is inserted. These comments are shown in a list, and the chosen word is properly highlighted. Each listed comment contains also information about its type; the source entity associated and the source code file where it appears.

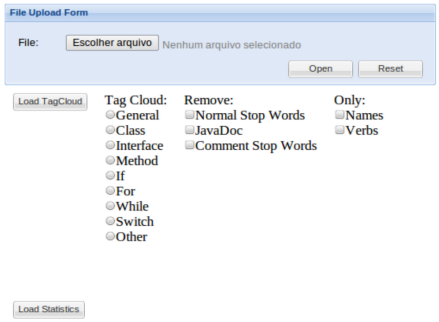

Fig. 2. Screenshot of the Web Interface

4 Experiments with Darius

In this section, we describe two experiments performed with Darius in order to compute actual information regarding the commenting practice.

The first experiment, will analyze the commenting practice in terms of quantity, providing information that could describe the frequency and positioning in which comments appear on source code.

The second experiment, will answer the questions raised in this paper, by analyzing the words included on comments, studying their domain and relat-

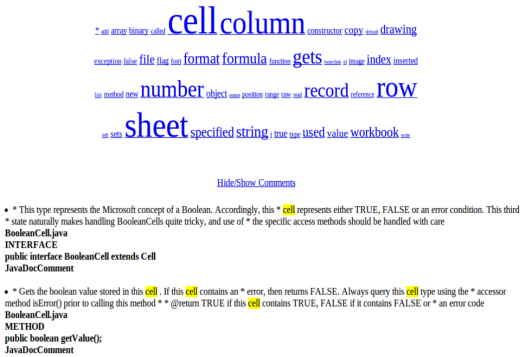

Fig. 3. Navigation over the Tag Cloud

ing that with the type of comment used and the type of source code entity commented.

4.1 Objects of Study

In order to perform the experiments, we have selected 10 software projects[3] written in the Java programming language. This selection has not followed any particular criteria, apart from the constraint that their Problem Domains should be different. Needless to say that the source code of all projects is fully available. The projects selected are described on Table 1.

Table 1. Description and size of each selected project

Project	Description	Files	LoC	Classes
iText	PDF Library	480	145666	403
ganttproject	Project Management Library	530	68945	394
gwt-dev	Google's Web Toolkit	987	192738	803
jEdit	Text Editor	531	176006	404
vuze	Peer-to-peer client	3284	785935	2463
junit	Tests Framework	154	10926	130
jfreechart	Chart Library	989	313231	876
antlr	Grammar Framework	221	85867	212
jexcelapi	Excel Library	461	98698	186
robocode	Programming Game of Robots	571	81519	485
Total		8176	1959531	6356

[3] These projects were retrieved from the SourceForge repository—`http://sourceforge.net`—in December 2010.

4.2 Comment Statistics

In the first experiment, we used Darius to compute statistics about the comments in the selected Java projects. This experiment had two main goals: to obtain a slight idea of what is the practice of commenting on real-world projects; and to check whether the quantity of comments is significant in order to perform more advanced studies, such as a qualitative study of the comments.

Methodology The procedure followed for performing this study was rather easy. We executed Darius in order to calculate statistics for each one of the projects, individually, and then gathered all the projects in a "big project" and repeated the analysis over the global project.

Table 2. Comments Frequency in the projects (detailed analysis)

					Type of Comments		
Project	Comments	Comment Lines	CommL / Comm	Comm / LoC	IC	BC	JD
iText	13343	35246	3.6	0.09	4930	3777	4636
ganttproject	4468	7544	2.99	0.06	2925	814	729
gwt-dev	12969	31648	4.25	0.07	7219	866	4884
jEdit	18986	37749	5.11	0.11	806	14421	3759
vuze	27723	64023	4.83	0.04	18245	2319	7159
junit	519	2281	4.99	0.05	2	77	440
jfreechart	22516	83934	4.86	0.07	6592	2530	13394
antlr	5292	11678	5.6	0.06	3903	1380	9
jexcelapi	8594	24735	3.58	0.09	2338	779	5477
robocode	5071	15657	6.39	0.06	3108	102	1861
Total	119481	314495	4.5	0.06	63758	13375	42348

Results Table 2 contains the basic results computed by Darius: the number of comments (global and distributed by the three comment types); the number of comment lines; the average of comment lines per comment; and the ratio between comments and lines of code (LoC). One important detail is the fact that the average number of comment lines per comment, is only relative to JavaDoc and Block comments, because Inline comments contains, by their nature, only one line. From Table 2 we can conclude that projects from real life incorporate comments in a rate that rounds the 6% of comments per line of code. Most of the comments, are block or Javadoc comments with an average size of 4,5 lines per comment.

To complete this first numeric evidence, Table 3 is useful to perceive the policy of commenting in the Java programming language. As it is observed in this table, the programmers of these projects usually comment classes, interfaces and methods and tend to comment less the others low level source code entities.

Table 3. Percentage of Source Code Entities (SC) commented (#SC commented / #SC) (average in all projects)

If	For	While	Switch	Class	Interface	Method
7	8	6	5	69	60	45

Table 4. Most used type of comment per type of source code entity (average in all projects)

If	For	While	Switch	Class	Interface	Method
IC	IC	IC	IC	JD	JD	JD

Table 4 shows in detail the comments frequency, separated by type of source code entity. JavaDoc comments is the most preferred type to comment classes, interfaces and methods. On the other hand, inline comments are most used to comment the others source code entities.

Roughly speaking, we can say that the number of comments embedded in the source code justify that we proceed with this preliminary study; now it is necessary to inspect the comments content, to decide whether they encapsulate valuable information on Problem or Program Domains.

4.3 Domain Analyzer

The second experiment had the main goal of studying and analyzing the domains to which the comment words belong. By performing this experiment, we might be able to get a slight idea of the relation between commenting and the Problem and Program Domains, and provide the answers to the questions raised in this paper.

Methodology In order to perform this experiment, we created a list of Problem Domain terms for each project and a list of Program Domain terms[4], the same for all projects (since all of them are written in the same programming language). To create the lists of Problem Domain terms, we analyzed the websites and manuals of every project, and selected the most relevant words, considering the requirements. All the lists of Problem Domain terms included in average 50 terms. The list of Program Domain terms (75 terms), included Java Language keywords and common words from the programming world, such as “class“, “array“ or “increment“.

To perform the analysis after opening each project, we have submitted to the Darius Domain Analyzer the respective list of Problem Domain terms and also the list of Program Domain terms. In this way, the Domain Analyzer computed and printed out the frequency in which terms occur in comments. It it also important to point out that non-valuable words (as described in 3.3) were removed from the table of comment words, so they did not enter in this analysis.

[4] All the list are available in `http://www.di.uminho.pt/~gepl/DARIUS`

Results As this second experiment, oriented towards the study of the commenting practices concerned with the occurrence of problem and program terms, mainly relies upon the two lists of terms chosen to characterize both domains, we started by assessing those lists. Table 5 shows the percentage of words in each list that actually occur anywhere in the comments of every project.

Table 5. Percentage of domain words (DW) found (#DW Found / #DW)

Project	Problem Domain	Program Domain
iText	91.23	87.67
ganttproject	77.97	69.86
gwt-dev	58.11	93.15
jEdit	88.89	89.04
vuze	88.1	93.15
junit	84.0	67.12
jfreechart	93.75	91.78
antlr	93.02	87.67
jexcelapi	86.67	89.04
robocode	86.05	82.19
Total	83.05	85.07

The average percentage found for both domains is around 84%. This value is actually bigger (near 90%) if we discard the 2 values clearly behind the average (ganttproject and gwt-dev).

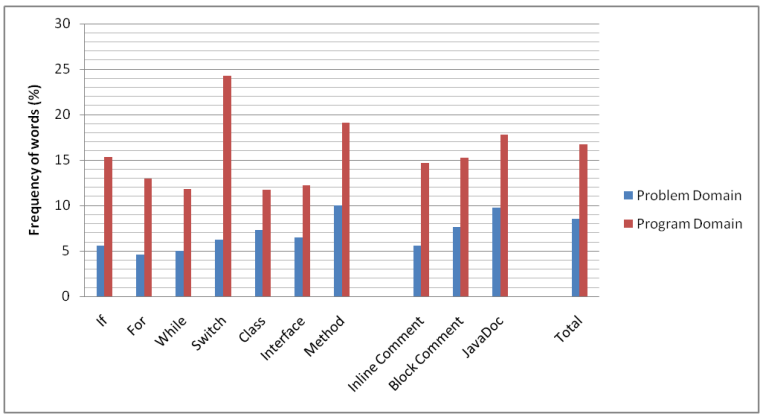

Fig. 4. Frequency of words for each domain

The chart displayed on Figure 4 is crucial for this study. It shows the frequency of Problem Domain terms and Program Domain terms per type of com-

ment, per type of source code entity and the total. There are several evident conclusions that can be drawn. Comments extracted from the several chosen projects include, indeed, Problem and Program Domain terms members of the lists created to characterize each domain. Program Domain terms are more frequent than Problem Domain terms, in a rate of 2:1. This shows that programmers write actually comments concerning both domains, but they tend to use more Program Domain concepts. Another interesting point, is the fact that 25% of the terms used in comments are Program or Program Domain oriented, which is a relevant result for Program Comprehension purposes. JavaDoc comment is the richest type of comment in terms of Program and Problem Domain terms, which can indicate that it is very useful for Program Comprehension purposes. JavaDoc and Block comments follow the rate of two Program Domain terms to one Problem Domain term, as mentioned above. However, as observed, Inline Comments do not follow this rule, as Program Domain terms are even more frequent. In general, the use of Problem Domain terms is more frequent in class and interface comments. On the other hand, Java statement and method comments include, in general, more Program Domain terms.

5 Conclusions and future work

In this paper we presented Darius, a tool that analyzes comments for Program Comprehension purposes. Using Darius, we conducted two experiments in order to study the role of comments on Program Comprehension, which answered a set of research questions, stated in the Introduction.

The first experiment, which focused on the quantitative results of comments in the studied Java software projects, showed that, in general, there is a higher tendency to comment classes, interfaces and methods. This experiment also showed a higher inclination for using JavaDoc comments to comment classes, interfaces and methods. On the other hand, InLine comments are most used to comment the other Java source code entities (statements such as if, while or for).

The second experiment was conducted in order to answer the questions enunciated in this paper. It was focused on the analysis of the content of comments, measuring the frequency of Problem or Program Domain terms for every selected software project. The results of this experiment showed that, in the studied projects, one in four comment words are Problem or Program Domain oriented. However, the frequency of Program Domain terms is bigger than the frequency of Problem Domain terms in a rate of 2:1. These results also showed that JavaDoc comments contain the biggest percentage of words from both domains and that InLine comments contain a bigger rate of Program Domain terms (3:1). As expected, the results showed also that class and interface comments contain the biggest percentage of Problem Domain terms and the other Java source code entities (methods and statements such as if, while or for) contain the biggest percentage of Program Domain terms.

Considering these results we now have established the foundation to proceed with our project aimed at the use of ontologies and comments to improve Pro-

gram Comprehension. The basic idea, that we intend to explore in the next step, consists in the construction of ontologies to model the Problem Domain and the Program Domain, and map their concepts into the program. We will take advantage of comments to implement these connections (from each ontology into the source code) precisely locating in the comments the ontology concepts (as we did in the experiment two); using the association between comments and source code entities, we create the bridge from each domain to the program.

References

1. Biggerstaff, T.J., Mitbander, B.G., Webster, D.: The concept assignment problem in program understanding. In: Proceedings of the 15th international conference on Software Engineering. pp. 482–498. ICSE '93, IEEE Computer Society Press, Los Alamitos, CA, USA (1993)
2. Brooks, R.: Using a behavioral theory of program comprehension in software engineering. In: Proceedings of the 3rd international conference on Software engineering. pp. 196–201. ICSE '78, IEEE Press, Piscataway, NJ, USA (1978)
3. Coleman, D., Ash, D., Lowther, B., Oman, P.: Using metrics to evaluate software system maintainability. Computer 27(8), 44–49 (Aug 1994)
4. Erlikh, L.: Leveraging legacy system dollars for e-business. IT Professional 2(3), 17–23 (2000)
5. Fjeldstad, R.K., Hamlen, W.T.: Application Program Maintenance Study: Report to Our Respondents. In: Proceedings GUIDE 48 (April 1983)
6. Haiduc, S., Marcus, A.: On the use of domain terms in source code. In: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension. pp. 113–122. ICPC '08, IEEE Computer Society, Washington, DC, USA (2008)
7. Maletic, J., Marcus, A.: Supporting program comprehension using semantic and structural information. In: Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on. pp. 103–112 (May 2001)
8. Marcus, A.: Semantic-driven program analysis. Ph.D. thesis, Kent, OH, USA (2003), aAI3100844
9. Rugaber, S.: The use of domain knowledge in program understanding. Ann. Softw. Eng. 9, 143–192 (January 2000)
10. Shaft, T.M., Vessey, I.: Research report–the relevance of application domain knowledge: The case of computer program comprehension. INFORMATION SYSTEMS RESEARCH 6(3), 286–299 (1995)
11. Sommerlad, P., Zgraggen, G., Corbat, T., Felber, L.: Retaining comments when refactoring code. In: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. pp. 653–662. OOPSLA Companion '08, ACM, New York, NY, USA (2008)
12. Tenny, T.: Program readability: procedures versus comments. IEEE Transactions on Software Engineering 14(9), 1271–1279 (Sep 1988)
13. Woodfield, S.N., Dunsmore, H.E., Shen, V.Y.: The effect of modularization and comments on program comprehension. In: Proceedings of the 5th international conference on Software engineering. pp. 215–223. ICSE '81, IEEE Press, Piscataway, NJ, USA (1981)
14. Zhao, W., Zhang, L., Liu, Y., Sun, J., Yang, F.: Sniafl: Towards a static noninteractive approach to feature location. ACM Trans. Softw. Eng. Methodol. 15, 195–226 (April 2006)