Komentarze w kodach wybranych programów

Comments in the codes of selected programs

Sviatoslav Skhut[1], Kateryna Iholkina[1],

ABSTRACT: Writing comments is as important as writing code. The main purpose of using comments is to improve readability of our code but frequently thoughtless comment writing decrease understandability of source code. Comments must be concise and precise simultaneously. Also, when our code is changed, comments for this code must be changed too. While using comments in our code we must realize that if expressiveness of our programming language allows us to express clearly what we want in code, there is no need to write comments at all. And if we decide to use comments, they must be extremely accurate and understandable, because another person must understand, what we do and most importantly, why we do it.
Frequently comments can be replaced with good clear names of variables, functions or classes. Also, we can replace our comments with assertions. Comments should clarify and explain our intentions. Copyrights and an authorship can be implemented using comments too. But our IDE can do these things automatically.
Classification of comments depends on their place in code, for which type of code they are attached and format.

Keywords: source code, software and its engineering, documentation, software management, code comments

INTRODUCTION: Improving code commenting techniques is important for programmers for several reasons: code maintenance takes about 40-80% of the lifetime cost of a piece of software [1], besides software is rarely maintained by its original author for its whole life. It's obvious that, the author`s perception of the code is very different from this of the following developers. Some moments and solutions in the code, self-explanatory for the author, can create a lot of problems for programmers supporting or refactoring the program. Due to poor documentation and poor-quality comments it is often even easier to write a new program than to change an old one. Thus, writing a code with useless, unreliable and inaccurate comments provides to large increase in cost.
Unfortunately, there is no general standards and conventions of writing and formatting code comments. In last twenty years there was published a lot of books, thesis and blog posts on the topic of programming style, clean code writing and code documentation. Namely, "The elements of programming style" by Brian W. Kernighan and P. J. Plauger, a study supporting point of view that source code should be, first of all, human readable. We should also note Robert C. Martin`s book "Clean Code. A Handbook of Agile Software Craftsmanship". A huge part of this study describes patterns and practices of writing maintainable code and efficient comments. In addition to these books our essay is based on "Java code conventions" published by Sun Microsystems Inc. in 1997 and "C++ style guide" powered by Google Inc.
The aim of our paper is to treat a problem of code commenting in the source code of selected programs. In the first two chapters we pay attention on the different code comment`s classification based on their function, placement and format. In the third chapter we provide an analysis of costs and benefits of using comments. On the examples of different programs, we consider the best practices of code commenting that should be in each program such as legal comments, clarifications and TODO comments and explanations of intent. Also, we study examples of useless and sometimes even dangerous comments, that should never appear in the source code. The last part of this paper is devoted to techniques that make the code more clear and readable without using comments.

1. COMMENT TYPES

Comments can be used for different purposes in different parts of the program. They fall into one of three categories: header comments, block comments and trailing comments, which describe successively smaller areas of code.

1.1. Header comments.

This class of comments serves to help reader navigate, understand a general purpose of the code and use the code itself. This category of comments makes a maintenance

1.Wrocławska Wyższa Szkoła Informatyki Stosowanej „Horyzont”, ul. ks. Marcina Lutra 4, 54 - 239 Wrocław Email: K_Iholkina@gmx.com, sgshkut@gmail.com

of code easier. Thus, they should be included in any code planned to be in use more than a few weeks. The header comments usually occupy a number of lines (typically between 10 and 50) [2] and contains following elements:

- Filename
- Source control version history
- Creation date
- Revision history
- Author's name
- Copyright notice
- License summary
- Purpose
- Change history
- Restrictions
- Special hardware requirements (e.g. Analog/Digital signal converters)

Header comments are placed at the beginning of the program which makes them stand out and easier to remove or copy. The recommended practice is dividing a comment block onto the section to ameliorate its readability. Using capital letters for the section headings and tabbing information out allows to navigate and read them more quickly:

```
/*
 ...
 * GLOBAL DATA:
 *      int     DB_ErrStatus          Contains most recent database error
```

1.2. Class comments

If we have a non-obvious class, the comments are required. These types of comments should describe what this class serves for and how it should be used. The class comments should provide such information as: interface of the class, multiple threads (if any) and if it is possible a few small examples of code demonstrating its usage. When we separate implementation and declaration (e.g. .cpp and .h files), comments that describe use of the class should go together with declaration, comments that describe class operation and implementation should be insert into implementation file.

1.3. Function comments

As with class function, comments should appear when usage of the function is non-obvious. Comments attached to function declaration should describe the usage of the function. They shouldn't describe how the function performs its tasks, just tell reader in descriptive way what the function must do, what inputs and outputs are, in which way user must free memory (if the function allocates memory) also function override should be described if it is not trivial. Comments in function definition should describe operations. These types of comments describe how function works.

1.4. Variable comments

The actual name of variable should be enough for description of what it is used for. Comment can be attached if this variable need additional clarification. In classes we have data members. Their names must be descriptive enough too, and the comments are required if there are some non-obvious instances. Global variables should have accompanying comments that describe why they need to be global.

1.5. Explanatory comments

Well written and concise code will contain a lot of explanatory comments, which highly increase code readability and clearness. Even though explanatory comments are not necessary for each line of code, there are some items which definitely should have them.
For example, startup code, exit code, weird logic, regular expressions, sub routines and functions, long and complicated loops [3].
In startup code explanatory comments shoulo to mention how the program is initialized, what #defines do, what arguments are expected etc.
While writing the exit code explanator comments, we should properly treat normal and abnormal exit situations, error codes etc.
Subrouting and function explanatory comment should, first of all, clearly explain its tasks and purpose of its using. It is important to comment functions arguments passed and returned with mentioning values format and limits on values expected, as this is one of the biggest sources of bugs [4]. Thus, this kind of comments written before sub, routines greatly helps reader to gain a deep understanding of the following code.

2. COMMENT FORMATS

2.1. Block comments

Block comments are generally found within functions, methods, data structures and algorithms. Block comments have two main purposes:

- Commenting out code
- Writing long comments

In C, C++ and Java languages block comments begin with special separator "/*" and are terminated with "*/", as shown in the example below. A common used practice is to start block comments by a blank line to separate them from the rest of the code.

```
/* * Here is a block comment. */
```

The other technic of highlighting block comments is enclosing them into a rectangle box of stars and dashes.
The example of boxed comment is:

```
/***********************
 Comment in a box!!
***********************/
```

The initial "/*" could be followed by other characters such as "=", "_" or "-".

2.2 Single-line comments.
Single-line comment is a short comment which appears on a single line intented of the code that follows. As well as in the case of block comments, there is highly recommended to separate them from the rest of code by at least one blank line.
The example of single-line comment in Java code is:

```
if (condition) {
   /* Handle the condition. */
   ...
}
```

2.3. Trailing comments
Trailing comments are very short comments, which describe the action or use of a single line of code. They usually begin (and end) on the same line as the code they describe. For separating trailing comment from code it is common practice to tab it out. The comment should be far away from the code.
For example:

```
SelectSides( Players );          /* choose partners and positions    */
```

3. PROS AND CONS OF COMMENTING CODE

3.1. Why comments are not always good.
Comment can be very helpful if they are placed in correct place in right time. But frequently they just do mess or clarify the code, which we can understand without them. We need comments for clarification our motives (why we write our code that way) or even for warning about something. Nature of comments arises from low expressiveness of our programming languages. The best comment is the one that we don't need at all. That means that we can chose a good name for variable or function, decide to write an extra line that make our code more readable and understandable.
Another case is evolving our code. Chunks of it can be moved in another place or deleted or even rewrited. In this case we can face outdated misleading comments. Of course, programmers can maintain these comments, but it takes a lot of time and it`s better to change a piece of code and deal with comment only if we really need it in this place.
When we decide to write a comment, at first we need to think about people who will read it. Secondly we should do our best with this commenting. Time spending for writing a good comment will save a lot of time of our code readers.
Sometimes we can see redundant comments. They do nothing except amassing lines in our code. It means that there is no need to comment every single function or variable we declared. When we see a + b we already know what it does and there is no need to comment it. Usually this type of comments just makes it difficult to read the code.
Now we can see that comments are always helpful. There are a lot of cases when comments are not needed at all or when they just make code less readable.

3.1.1. Journal comments
Some programmers add a kind of "historical" comments containing their names, time or date and changes made every time they edit the code. For example:

```
// method name: pityTheFoo (included for the sake of redundancy)
// created: Feb 18, 2009 11:33PM
// Author: Bob
// Revisions: Sue (2/19/2009) - Lengthened monkey's arms
//            Bob (2/20/2009) - Solved drooling issue
 void pityTheFoo() {
    ...
}
```

There were some reasons to make log comments long, long ago, when there wasn`t a source control system making it automatically for us. Nowadays it is more likely to use one of the source control system and just fill the check-in comment boxes on our commits [5].

3.1.2. Noise comments
Sometimes comments are obvious and provide no new information. For example, each string of comments from code below can be discarded without loss of understandability:

```
/** The name. */
private String name;
/** The version. */
private String version;
/** The licenceName. */
private String licenceName;
/** The version. */
private String info;
```

Such code out commenting normally should be used in two cases: in code examples which serves teaching the concept of programming language, or in the case when programming language isn`t easily human readable (LikeAssembly).

3.2. Good comments
There are cases when we can't avoid using comments such as corporal rules or copyright. But we will not consider them now. A good example of good comment is to do comment. This type of comments can appear near the functions which will implement in the future (or not). It`s just a list of tasks that programmers want to do in future. Another example of good comments are warning comments. This type of comments warn other programmers about consequences of using code such as vulnerability or time of exe-

cution. Comments that describes our intents (why we decided to solve this problem this way or choose this data type etc.) or clarify our cod (when we really need it) are examples of the good comments too.

3.2.1. Legal comments

These comments should not be duplicated of contract or legal tome. Legal comments can include copyrights, refers to standard licenses, authorship. They also may refer to external documents. This type of comments should be included inck at the beginning of source file.

3.2.2. Explanation of intent

This type of comments explains why we have decided to use this implementation. It allows a developer to understand what is purpose of ou code. Also, it reduces situations where our intents aren't clear at a glance.

3.2.3. Clarifications

Sometimes the best way to describe developer explain to our code is write about it in readable form. For this reason, clarification comments may be used. This type of comments helps us to describe our obscure functions, returning values, non-obvious behavior etc.

3.2.4. TO DO comments

It is not a bad idea to include some "to do" lists in our code. It can be done with this type of comments. They can be connected to functions or pieces of code that we want to implement in future. Todo comments show developers that this function does nothing except reminding.

3.3. Alternative comments

As we mentioned above the one of the worst practice is out commenting code. Obvious, annoying, trashy, redundant comments lead to incomprehensible hard to maintain code. Having considered what comments' strengths and weaknesses are, we will treat how they could be replaced by other tools.

3.3.1. Identifiers as comments

Consider the example demonstrating how a typical comment can be encoded in an identifier:

```
Before:
 ++i;/* record another match of this expression */
After:
 ++number_of_expression_matches;
```

Huge part of source codes comments could be replaced by carefully and thoughtfully named variables, functions, methods and classes names. If a comment is intended to explain a complex expression, the expression should be split into understandable subexpressions using extract variable. If a comment explains a section of code, this section can be turned into a separate method via extract method.
However, identifiers could be completely misleading, if the programmer isn`t attentive when modifying code. This is the same problem which appears also when comments aren`t updated respectively to code, modifications. So, when we refactor code, we should be vigilant to change both comments and identifiers.

3.3.2. Replacement comments with assertions

From time to time it is reasonable to refactor a comment into an assertion. For example:

```
Before:
// value must not be negative
 public double squareRootOf(int value) {
...square root algorithm...
}
After:
public double squareRootOf(int value) {
Assert.isTrue(value >= 0);
   ...square root algorithm...
}
```

The benefits of this solutions are: supporting better testing, making debugging easier, serving as understandable comment about preconditions [6]. However, assertions slow down our code and may make a program incorrect when they are used improperly. So, assertions have some advantages as they are enforced as code and form programmable safeguards, but then also have all the disadvantages of code: expression of abstraction can be verbose and nontrivial

4. DOCUMETING SYSTEMS

There are several documenting systems available for various programming languages. These systems deal with the „Explanatory" type of comment. They create documentation out of comments from the code. Let's demonstrate these systems via Javadoc. This is the Java API contained in the JDK. It uses comments with specific tag /** to generate HTML pages with descriptions of all classes, interfaces, constructors etc. It also generates a tree with class hierarchy. For more details we can use documentation provided with JDK. PHP and C# also have documentation system, but the latter uses XML instead of HTML. These systems require from developers to maintain not only the code, but comments too.

CONCLUSIONS

When reading tricky code, there is nothing more helpful than well-written comment. At the same time, there is often nothing harder than writing a well-placed, brief and clear comment.
On the one hand, plain English is always easier to read than code. Comments can explain things that couldn`t be easy expressed in programming language, besides, they don`t affect program execution speed. Writing good comments

discipline programmer`s mind. Comments are shorter than the code they document and much easier to skim-read.
On the other hand, they reduce the readability of well-written code, in addition they are less precise that the code they document. Sometimes using a lot of comments encourage bad code, take up screen space and time to read. By the way, programmers often refactor code, but don`t update comments, which provides to a high risk of spending hours tacking up a bug, because you trusted a non-reliable comment.
Thus, programmers should use comments carefully, for preference when it is impossible to make a code self-explanatory. Before writing a comment, it is recommended practice to try to increase code expressiveness by introduce an explaining variable, extract a method, use more descriptive identifier, or replace a comment with assertion.
The questions "To write or not to write?", "How many?", "How detailed comments to write?" is still hotly debated one.

SOURCES

[1] Sun Microsystems Inc, Java code conventions, 1997
[2] David Straker, C Style: Standards and Guidelines, 1991
[3] Bernhard Spuida, The fine Art of Commenting, 2002, Tech Notes, general Series
[4] Brian W. Kernighan, P. J. Plauger, The elements of programming style, 1978, McGraw-Hill Book Company,
[5] Robert C. Martin, Clean Code. A Handbook of Agile Software Craftsmanship, 2009, Prentice Hall,
[6] Dori Reuveni, Kevin Bourrillion, Code Health: to comment or not to comment, 2017, blog post