International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878, Impact Factor (2015): 3.791

Distinguishing Identifiers Using Code Inspection

Sujata S. Deshmukh[1], S. Pratap Singh[2]

[1]Computer Engineering Dept., SP's Institute of Knowledge, Pimple Jagtap, Pune

[2]Computer Engineering Dept., SP's Institute Of Knowledge, College of Engineering College of Engineering, Pimple Jagtap, Pune

Abstract: ***Software quality is not defined in terms of quality attributes but instead must be inferred from characteristics that correlate to quality attributes and defect attributes. Readability of source code in one such attribute. Software characteristics have been identified by empirical research, which correlate well to source code. One of such software characteristic is choice of identifier name. Identifier names are primary means by which source code authors communicate their concepts to their readers. They are key mechanisms by which readers of source code such as maintenance programmers access and understands source code. There exists a relationship between flawed identifier names and software maintenance cost. Studying such a relationship is useful to assess whether naming conventions have impact on maintenance effort and to gain a deeper and finer-grained understanding of which program comprehension issues lead to code quality problems. We present in this paper the identifier analysis method that can identify flaws in Java identifier names. We found that percentage of using compound word identifier names is significantly more than simple word identifier names. Also percentage of compound word identifier names with no dictionary meaning is significant at variable, class and method level identifiers. It unnecessarily increase the splitting overhead of compound word identifier names before finding their dictionary meaning without any clear benefit in readability of source code.***

Keywords: Code Inspection, identifier analysis

1. Introduction

Modern programming languages permit the creation of clear, readable and meaningful identifiers. Programming conventions provide guidance on the typographical form of identifier names associated with particular language constructs and the parts of speech to be used in different types of identifiers. However, only limited advice is given on matters such as the length of identifiers. The poor identifier names are barriers to source code comprehension is sufficient reason to create good quality identifiers. However, there might be other consequences of poor quality identifier names such as poor understanding of code during maintenance and increased cost of maintenance. Given the importance of the natural language content and structure of identifer names to the readability of source code one can find relationship that exists between software quality and the quality of identifiers names used in the source code.[1]

Identifier name quality is multifactorial. The use of typography, as defined in programming conventions gives the reader clues to the role of each identifier. However, typography alone is insufficient. The good identifier name should clearly communicate the concept represented and its function through the use of natural language. Identifier names are crucial components of source code which have impact on program comprehension. As artifacts of the programmers thought processes, identifier names are mechanism by which source code may be accessed and understood. Similarly, they may reflect difficulties the programmer had understanding a problem and thus of potential defects in the finished software. Although there is work relating identifier naming and program comprehension, work has not been done that directly relates identifier naming and code quality. Studying such a relation-ship is useful help to assess whether naming conventions have impact on maintenance effort and to gain a deeper and fine-grained understanding of which program comprehension issues lead to code quality problems.[2][4]

As Eclipse is widely used platform for Java development, we have used Eclipse Java Development Tool (JDT), Eclipse Java Model and Eclipse Abstract Syntax Tree API (AST API) to develop the identifier analysis module that can be used as plug-in for Eclipse.

2. Literature Survey

A. Importance of Coding Conventions In Soft-

Ware Maintenance and Testing

Coding standards have become increasingly popular as a means to ensure software quality throughout the devel-opment process. They typically ensure a common style of programming which increases maintainability and prevent the use of potentially problematic constructs, thereby increasing reliability. The rules in such standards are usually based on expert opinion gained by years of experience with a certain language in various contexts. Over the years various tools have become available that automate the checking of rules in a standard helping developers in locating potentially difficult or problematic areas in the code. These also include commercial offerings. Such tools generally come with their own sets of rules against which they check for violations of coding standards. However, in spite of the availability of appropriate standards and tools, there are several issues hindering adop-tion. In spite of the widespread use of coding standards and tools enforcing their rules, there is little empirical evidence supporting the intuition that they prevent the introduction of faults in software.[2][10][14][16]

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it is a constant, package or class which can be helpful in understanding the code. Code conventions are important to programmers for a number of reasons[5][6]:

1) 80 percent of the lifetime cost of a piece of software goes to maintenance.
2) Hardly any software is maintained for its whole life by the original author.
3) Code conventions improve the readability of the soft-ware allowing engineers to understand new code more quickly and thoroughly.
4) If developer ship source code as a product, developer need to make sure it is as well packaged and clean as any other product.

B. Naming Conventions for Java Identifiers

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it is a constant, package or class which can be helpful in understanding the code. Following table enlists the Java identifier naming guide-lines.[5][6]

Identifier Type	Rules for Naming	Examples
Classes	Class names should be nouns. In mixed case with the first letter of each internal word capitalized.	class Raster; class Image-Sprite;
Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; interface Storing;
Methods	Methods should be verbs. In mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run(); runFast();
Variables	Variable names should be short yet meaningful. The choice of a variable name should be mnemonic.	int i; char c; float myWidth;
Constants	class constants should be uppercase with words separated by underscores ("_").	int MIN_WIDTH = 4; int MAX_WIDTH = 999; int GET_THE_CPU = 1;

Figure 1: Naming conventions for JAVA identifiers

C. Role of Intermediate Program Representation in Software Maintenance

A fundamental goal of software engineering is to make program development and maintenance easier, faster and less error prone. This includes addressing problems like
1) Understanding what an existing program does and how it works.
2) Understanding the differences between several versions of a program.
3) Understanding whether design rules have been followed in coding phase.

Tools that assist programmers with such problems are most useful if they are language based that is, if they incorporate knowledge about the programming language in use. On the other hand, it is desirable to base these tools on language-independent algorithms and data structures so as to avoid the need to re-design and redevelop a set of tools for every different programming language. One of such intermediate representation which is widely used in software maintenance is Abstract Syntax Tree(AST). Abstract syntax trees (ASTs)[7] are known from compiler construction where they build the intermediate data format which is passed from the analytic front-end to the synthetic back-end. In software development, ASTs are used as a model of the source code. They represent a program on the level of the abstract syntax that means that they are independent from the concrete syntax for identifiers, operators, conditions or statements of the underlying programming language. The striking advantage in the use of ASTs in contrast to source programs is the higher level of abstraction. Hence, the algorithms have to be developed only once and can then be used for programs written in various different languages. The Abstract Syntax Tree is the base framework for many powerful tools of the Eclipse IDE including re-factoring, quick fix and quick assist. The Abstract Syntax Tree maps plain Java source code in a tree form. This tree is more convenient and reliable to analyze and modify programmatically than text-based source.[7][14]

Achieving Code Quality through Identifier Names

The development of software, like any other manufacturing processes can introduce defects in the resultant product. However, unlike other manufacturing processes, software is manufactured extensively once and then modified until sufficient defects have been removed such that the customer will accept the product. The customer may only be concerned with capability and will have no further interest in the internal software structure. However, some customers are becoming increasingly interested in the internal structure of software delivered to them as this has a flow-on effect to the cost of maintaining the product (i.e., the cost of modifying and adding new capability which is distinct from the activity of software bug fixes). Customers may hence contractually impose soft-ware quality standards on their contractors. These standards are interpreted by the contractor and typically further standards are derived such as the project coding standards.[2][8][9] However, the software engineer suffers from various cognitive limitations that make the production of readable software a difficult task. In addition, the software engineer is not prepared well by their education to address quality in their software development practices. Often, they are given very general instruction regarding what constitutes software quality and rarely they are instructed in how to achieve the quality production of

software. Their education is heavily waited towards discussing the production phase of the software life cycle with the maintenance phase given a rudimentary treatment. Similarly, their peer culture and the culture of their management places little emphasis on future maintenance of the software during the development phase. There are no industry wide adoption of software quality tools that actively assist the software engineer to produce a quality product.[9][15]

A software quality characteristic that has the potential to im- prove software quality is the choice of identifier name and this is particularly so in large software systems. Identifier naming style guidelines supported by empirical evidence and generally accepted by software professionals to direct towards improved source code readability are candidates for automation by a static analysis tool. Such an automated tool could make visible aspects of software quality that are less keenly perceived by the novice programmer and could assist in their education along the path to expert status. Hence is the motivation to develop a plugin that can do identifier analysis in Eclipse IDE[19] which is widely used platform for Java development.[1][8][10] Table I shows identifier naming style guidelines in gen- eral to achieve code quality through identifier names.[2]

3. Implementation Details

Todays large, complex software systems require automatic software analysis and recommendation systems to help the software engineer in completing maintenance tasks effectively and efficiently. The software maintainer must gain at least partial understanding of the concepts represented by existing source code before making modifications. A programmer codes the concepts and actions in terms of program structure and helps to convey the intent and application domain concepts to human readers through identifier names and comments. Thus, many of the program search, concern location, code reuse and quality assessment tools for software engineers are based on analyzing the words that programmers use in comments and identifiers. Hence identifier analysis in source code becomes important.

In previous section, it has been seen that there exist a relation between identifier naming conventions and quality of software developed as well as readability and understanding of the same during maintenance. There is a need of tool that can find identifiers automatically in Java source code and validate them against identifier naming conventions of Java. As Eclipse is widely used platform for Java development, we have developed a module that can do identifier analysis automatically of entire Java project in Eclipse workspace. By extending the

Table 1: Identifier Naming Style Guidelines

Name	Description
Capitalisation Anomaly	Identifiers should be appropriately capitalised.
Excessive Words	Identfier names should be composed of no more than four words or abbreviations.
External Underscores	Identfiers should not have either leading or trailing underscores.
Long Identifier Name	Identifier names of more than twenty above characters should be avoided where possible.
Naming Convention Anomaly	Identiers should not consist of non-standard mixes of upper and lower case characters.
Non-Dictionary Words	Identier names should be composed of words found in the dictionary and abbreviations and acronyms that are more commonly used than the unabbreviated form.
Number of Words	Identiers should be composed of between two and four words.
Numeric Identfiier Name	Identiers should not be composed entirely of numeric words and numbers.
Short Identifier Name	Identfiiers should not consist of fewer than eight characters, with the exception of c, d, e, g, i, in, inOut, j, k, m, n, o, out, t, x, y, z
Type Encoding	Type information should not be encoded in identifier names using Hungarian notation or similar

Eclipse functionality with the addition of identifier analysis plugin developers can get identifier analysis of all Java projects in workspace in just single click of mouse. With identifier analysis it doesnt only mean finding the identifiers and validate them against naming conventions but also there is need to find other fields related to identifiers which contribute to readability of code being analyzed during its maintenance.

For each identifier found there is need to find following fields.

1) Type of identifier
It can be of type class, method, variable, interface or constants.

2) Simple or compound word
This field find out whether identifier found is sin- gle word identifier i.e. simple or it is combination of multiple words i.e. compound word

3) Dictionary word or non dictionary word
This field find out whether name used for identifier is belonging to English dictionary or not. This field has impact on readability of code during maintenance of the same.

4) Grammatical sense of identifier
In this field if identifier found is dictionary word then there is need to find grammatical sense of the same i.e. noun, verb. According to Java coding conventions class identifiers should be nouns and method identifiers must be verb.

To develop a plugin in Eclipse that can analyze identifiers in Java source code and validate them against identifier naming conventions, we have used Eclipse Java Development Tool

(JDT), Eclipse Java Model and Eclipse Abstract Syntax Tree API (AST API). Following are the steps of implementation.

1) Create parser plugin project in Eclipse with appropriate name and also choose appropriate Eclipse platform version.
2) Select one of the available templates to generate a fully functioning plugin. This work adds Id-Analyzer command plugin in Eclipse platform, so hello world command template is used.
3) Add packages on which this plugin depends without explicitly identifying their originating plugin.
4) Modify plugin.xml file according to requirements to give name to command and menu under the command.
5) After writing the handler program create Java archive of the plugin project and copy it to appropriate destination directory. Now archive is ready for deployment.
6) Copy created Java archive and dictionary into Eclipse plugin directory (default directory is /usr/lib/eclipse) and make it executable.
7) Restart eclipse. This plugin finds all indentifiers including classes and methods in current Java project. It displays its type, whether it is complex word, dictionary word etc.

4. Performance Evaluation and Result Analysis

For performance evaluation, comparison of the identifier flaws found by FindBugs[12] and identifier analysis plug-in of Eclipse has been made. we have used a total of 4 established Java open source applications and libraries for investigation from a variety of domains and uses including GUI applications, programmers tools and charting applications. The variety of projects chosen reduces the possbility of any unanticipated project or domain specific influence on identifier names. Table II shows the version and size of each code base analyzed in terms of number of classes and thousands of non-commenting source statements (KNCSS), as measured by FindBugs[12] which is static analysis tool for finding flaws in Java source code.

FindBugs is a static analysis tool used for Java source

Table 2: Source Code Analyzed

Source	KNCSS	classes
Ant 1.71	72	1639
Tomcat 6.08	114	2128
jEdit 4.3	58	2069
jFreeChart 1.01	61	1031

code which generate priority warnings for identifier naming flaws present in Java source code. In each Java open source applications considered for performance evaluation with respect to same naming conventions of Java identifers, e.g. capitalization anomaly of class and method level identifiers, grammatical sense of class and method level identifiers. Figure 1 shows comparison of identifier naming flaws found by FindBugs and Id-Analyzer in each Java open source applications considered for performance evaluation with respect to same naming conventions of Java identifers After analyzing the warnings generated by FindBugs, it has been found that warnings have been generated for minority of classes. Similarly, many of the identifier flaws were found in minority of classes. Also, it shows that associations between priority one warnings and identifier naming flaws are less common than the more consistent associations for priority two warnings. All the identifier naming flaws of Java are associated with priority two warnings in all open source projects.

While analyzing the results of plug-in, it has been observed

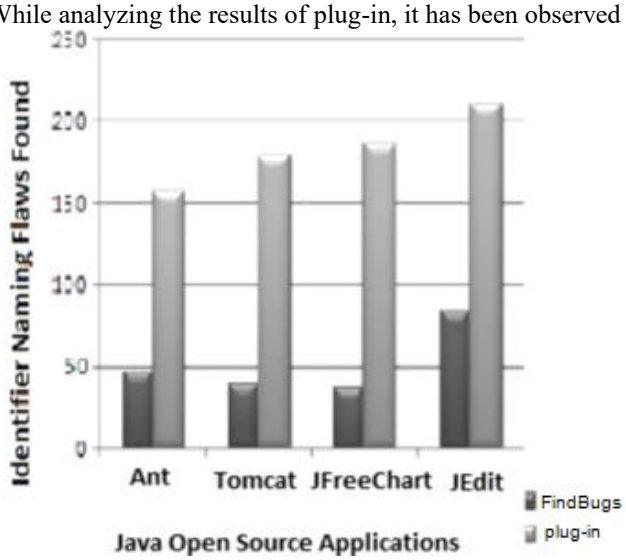

Figure 2: Comparison of identifier flaws found by FindBugs and plug-in

That percentage of using compound word identifier names is more as compare to simple word identifier names in all open source projects considered during result analysis. The reason behind is single word may not be as meaningful as compound word considering its English dictionary meaning. The compound word identifier name also convey the purpose of using that name and function actually it performs. In short compound word identifier names increase readability of code during maintenance and testing if constructed properly.

It has been also found that majority of the compound word identifier names used in all open source Java projects considered are non-dictionary words. If compound word identifier names are used and they are not carrying dictionary meaning then they are not useful in increasing readability and understanding of code during maintenance and testing. Also, compound word identifier names poses a challenge of splitting them before finding out their dictionary meaning. So, use of compound word identifier names that doesnt have dictionary meaning should be avoided because unnecessarily it increase splitting overhead before finding out their dictionary meaning without any clear benefit in enhancing readability and understanding of code. Figure 2 shows percentage of compound word identifier names with dictionary meaning. It includes identifier names at variable, class and method level in all open source Java projects considered. It also shows overall percentage of compound word identifier names that are dictionary words.

Also, as a part of result analysis, percentage of dictionary word identifiers found in all Java open source project has been calculated which will have impact on readability and understanding of code during maintenance. Also number of simple and compound word identifier names found in each Java project have been recorded. Figure 3 shows percentage of

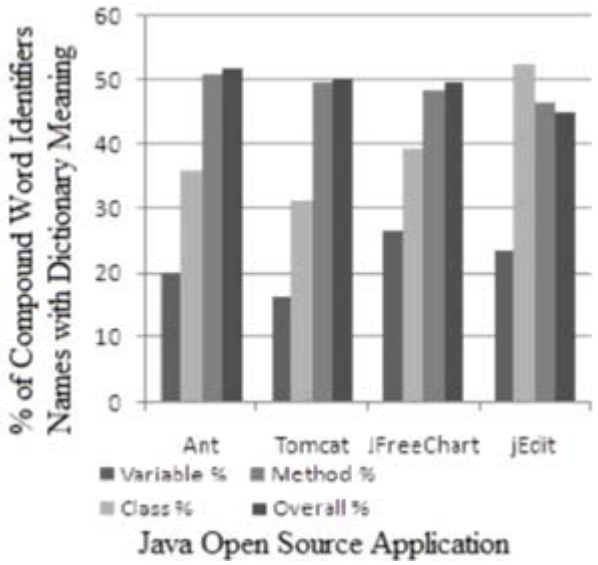

Figure 3: Percentage of compound word identifier names with dictionary meaning in Java open source applications found by plug-in

Dictionary word identifier names found during analysis of Java open source projects considered for performance evaluation. This percentage found during analysis will have impact on readability and understanding of code during maintenance of the same. If more is the percentage of dictionary word

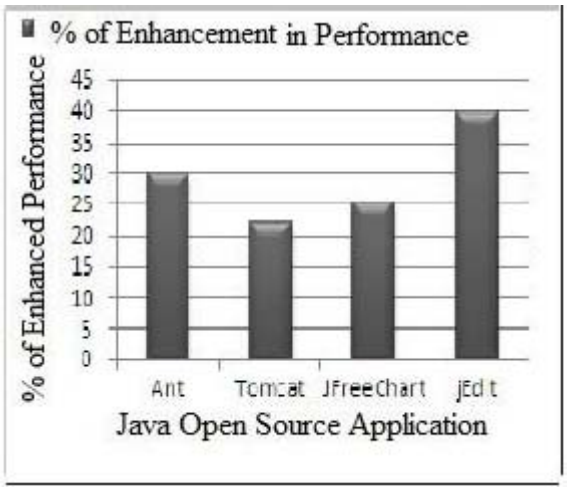

Figure 4: Percentage of dictionary word identifier names found in Java open source applications by plug-in module

identifier names found, more will be the readability and understanding of the code during maintenance. This can help in reducing maintenance effort and cost of source code. Figure 4 shows percentage of simple and compound word identifiers found in all Java open source projects considered for result analysis. It clearly shows that percentage of using compound word identifier names is more than than simple word identifier names because well constructed compound word identifier names with dictionary meaning can enhance readability of code during maintenance.

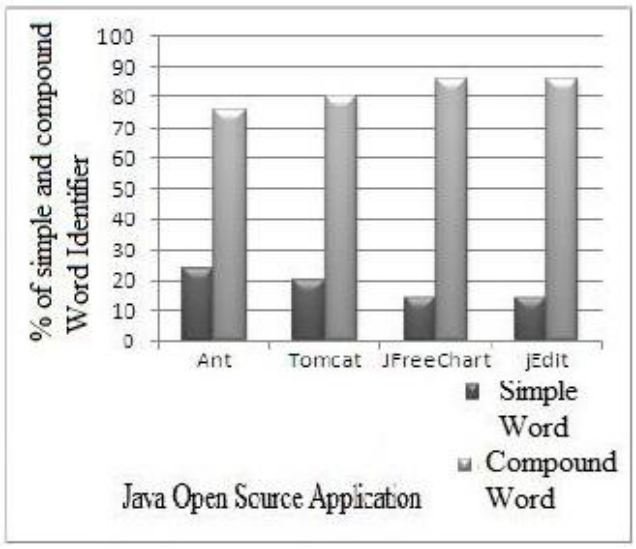

Figure 5: Percentage of simple and compound word identifier names found in Java open source applications by plug-in module

5. Conclusion

The contribution of this paper is a detailed understanding of the composition of a good quality identifier name and the relationship between identifier quality and source code quality. While this will allow the creation of detailed naming conventions, the emphasis must be on making a coherent contribution to improve the software development process rather than overloading the programmer with new rules to remember. With the same identifier naming guidelines of Java, comparison of identifier naming flaws of plug-in module and FindBugs has been made. Comparison shows clear enhancement in finding out identifier naming flaws with plug-in in all Java open source applications considered.

After analyzing the results it has been observed that percentage of using compound word identifier names is more as compare to simple word identifier names in all open source projects considered. The reason behind is single word may not be as meaningful as compound word considering its English dictionary meaning. The use of compound word identifier names with no dictionary meaning should be avoided during development because it increases the overhead of splitting them without any enhancement in code readability and understanding. The plug-in would require evaluation through empirical study; initially with a small group of programmers and subsequently through surveys of users and the collection of anonymised usage statistics.

References

[1] Simon Butler, Michel Wermelinger, Yijun Yu, Helen Sharp Exploring the Influence of Identifier Names on Code Quality: an empirical study 14th European Conference on Software Maintenance and Reengineering. IEEE 2010.
[2] Simon Butler, Michel Wermelinger, Yijun Yu, Helen Sharp Relating Identifier Naming Flaws and Code Quality: an empirical study 16th Working Conference on Reverse Engineering. IEEE 2009.

[3] F. Deissenboeck and M. Pizka, Concise and consistent naming Software Quality Journal, vol. 14, no. 3, pp. 261-282, Sep 2006.
[4] V. Rajlich and N. Wilde, The role of concepts in program comprehen-sion in Proc. 10th Intl Workshop on Program Comprehension. IEEE 2002, pp. 271-278.
[5] Sun Microsystems, Code conventions for the Java programming lan-guage http://java.sun.com/docs/codeconv, 1999.
[6] A. Vermeulen, S. W. Ambler, G. Bumgardner, E. Metz, T. Misfeldt, J. Shur, and P. Thompson, The Elements of Java Style Cambridge University Press, 2000.
[7] G. Fischer, J. Lusiardi, J.Wolff von Gudenberg Abstract Syntax Trees and their Role in Model Driven Software Development International Conference on Software Engineering Advances. IEEE 2007.
[8] R. P. Buse and W. R. Weimer, A metric for software readability in Proc. Intl Symp. on Software Testing and Analysis. ACM 2008, pp. 121-130.
[9] P. A. Relf, Achieving software quality through identifier names 2004, presented at Qualcon 2004 http://www.aoq.asn.au/conference2004/conference.html
[10] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, Whats in a name: A study of identifiers in 14th IEEE Intl Conf. on Program Comprehen- sion. IEEE 2006, pp. 3-12.
[11] C. Boogerd and L. Moonen, Evaluating the relation between coding standard violations and faults within and across software versions in Proc. 6th Intl Working Conf. on Mining Software Repositories. IEEE 2009, pp. 41-50.
[12] FindBugs, Find Bugs in Java programs http://fndbugs.sourceforge.net/, 2008.
[13] K. Atkinson, SCOWL readme http://wordlist.sourceforge.net/scowl-readme, 2004.
[14] Paul Anderson, Thomas Reps, Tim Teitelbaum and Mark Zarins, Design and Implementation of a Fine Grained Software Inspection Tool IEEE Transactions on Software Engineering, vol. 29, no. 8, August 2003, pp. 721-733
[15] Paul Anderson, Thomas Reps, Tim Teitelbaum and Mark Zarins Tool Support for Fine-Grained Software Inspection Published by the IEEE Computer Society,IEEE 2003.
[16] Cathal Boogerd Leon Moonen, Prioritizing Software Inspection Results using Static Profiling Proceedings of the Sixth IEEE International Workshop on source Code Analysis and Manipulation, IEEE 2006. f the First Workshop On Inspection in Software Engineering, PARIS, JULY, 2001
[17] Butler, S., Wermelinger, M. , Yijun Yu, Sharp, H." INVocD: Identifier name vocabulary dataset" Mining Software Repositories (MSR), 2013 10th IEEE Working Conference in May 2013
[18] Arnaoudova, V., Eshkevari, L.M., Di Penta, M., Oliveto, R. more authors "REPENT: Analyzing the Nature of Identifier Renamings" Software Engineering, IEEE Transactions on (Volume:40 , Issue: 5) March 2014
[19] http://www.eclipse.org