The Effect of Identifier Naming on Source Code Readability and Quality

Simon Butler
Centre for Research in Computing, The Open University, Walton Hall, Milton Keynes MK7 6AA, UK

ABSTRACT

Identifier names are natural language representations of program concepts in source code, which play an important role in program comprehension. Ill-considered, or poor quality, identifier names may hinder program comprehension, but is it possible they may also indicate problems in the source code, including bugs? We outline methods used to evaluate identifier quality, and propose the development of a more sophisticated understanding of identifier name quality, which will facilitate further exploration of the possible implications of poor quality identifier names.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*Standards*; D.2.8 [**Software Engineering**]: Metrics

General Terms

Measurement

1. INTRODUCTION

Identifier names communicate program concepts [5] to the reader, and are important elements in the process of program comprehension [10], and, thereby, software maintenance. Identifier names are also artefacts of the cognitive processes of the programmer, and may be seen as a reflection of the programmer's understanding of the concepts involved in the problem solution being described in the source code.

That poor quality identifier names can be a barrier to program comprehension [5, 10] is sufficient motivation to create good quality identifiers. However, could there be wider implications of poor quality identifier names? Recent research on source code readability [1] demonstrated a link between less-readable source code and software defects. However, the research ignored the quality of identifier names. Given the importance of identifier names to source code readability we hypothesise that the quality of identifier names is related to the quality of software.

2. RELATED WORK

Research into the quality of identifier names, generally, focuses on their role in program comprehension, i.e. the utility of identifier names to the programmer and the readers of source code [5, 7, 8, 9]. Deissenboeck and Pizka [5] developed a formal model to describe the relationship between identifier names and program concepts, which was applied to create clear identifier names. Lawrie et. al. found readers of source code understood identifiers composed of dictionary words more readily than those composed of abbreviations or single letters [8], and that the trend over the last 30 years has been towards identifiers constructed from dictionary words as opposed to abbreviations [7]. A study of identifier names by Liblit et. al. [9] showed sophisticated use of grammar in differing types of identifier name, e.g. the use of valence in method names to imply when the method accepts one or more arguments.

More recently, the influence of source code readability on software quality was investigated through the development of a readability metric for Java by Buse and Weimer [1]. The readability metric is derived from textual and typographical features of source code, such as the use of whitespace, the number of brackets and parentheses, and line length. The readability metric does not assess the quality of identifier names; it records only the length and frequency of identifiers, without reference to typography or natural language content. Buse and Weimer found Java methods identified as containing defects by FindBugs [6] were largely judged to be less-readable by the readability metric.

3. APPROACH

In recent studies [2, 3] we applied an adapted set of identifier naming style guidelines [11] as a ruleset to evaluate the quality of identifiers in eight open source Java applications. The identifier naming style guidelines embody much of the advice given in the programming literature and programming conventions [12] on the typography and natural language content of identifier names. Unlike programming conventions, the identifier naming style guidelines have been evaluated empirically.

To investigate any possible effect of poor quality identifier names on software quality we examined two measures of internal software quality: readability, as measured by the readability metric [1]; and the presence of defects determined by FindBugs [6]. We found statistically significant associations between poor quality identifiers and both software defects and less-readable source code.

In more recent work we applied a statistical technique commonly used in medicine to evaluate diagnostic tests. We found that it may be possible to apply the more narrowly defined identifier flaws as screening tests at the method level. In other words, the absence of poor quality identifiers in a

Copyright is held by the author/owner(s).
ESEC/FSE Doctoral Symposium'09, Aug. 25, 2009, Amsterdam, The Netherlands.
ACM 978-1-60558-731-8/09/08.

method is largely indicative that the source code is more readable and that there are no defects associated with the method.

4. CONCLUSIONS AND FUTURE WORK

So far, our studies have found statistical associations between poor quality identifier names and poor quality code at both the class and method level. We also found statistical associations between poor quality identifiers and less-readable source code. However, these findings are based on relatively narrow views of identifier quality and software quality, and have been constrained to open source Java projects. To investigate the possible effects of poor quality identifier names on source code readability and software quality further we need to refine our understanding of identifier name quality in terms of typography, natural language content and grammar. We also need to refine our perspective of software quality at the source code level, and expand the range of source code studied in terms of development methods and, eventually, programming languages.

The work of Liblit et. al. [9] can be applied to create a taxonomy of the grammatical structure of identifier names, from which we expect to be able to derive a grammar of identifier names similar to that developed for C identifier names by Caprile and Tonella [4]. We intend to apply the grammar as a means of evaluating the quality of identifier names. However, identifiers do not exist in isolation. Necessarily the quality of an identifier name is related to the context in which it is found, particularly where identifier names are chained as in object oriented languages. Thus, we propose to develop a metalanguage to describe identifiers *in situ* and their relationships with other adjacent identifiers and programming language elements, such as operators. Such a metalanguage would describe source code as the mixture of programming language and natural language identifiers that is perceived by the human reader, and may offer another perspective on the readability of source code, not just identifier name quality.

Our studies have focused on a single programming language, Java. Given the linguistic similarities of programming languages, we will apply the lessons learnt from developing a detailed model of identifier quality for Java to the development of a more general model that can be applied to other languages. It may be possible to generalise the resulting model to other C syntax based languages, such as C, C++ and C#. However, the process of extending the model to other languages will require an appreciation of the subtleties of the grammatical relationships of identifiers in different syntaxes. For example the use of valence is pronounced in SmallTalk because of the method naming syntax.

We are already investigating the use of static analysis tools other than FindBugs to provide a broader perspective of code quality, with the intention of synthesising a methodology that incorporates software metrics, such as McCabe's cyclometric complexity, and defect reports. It is imperative to develop a more generalised methodology to evaluate the quality of source code based on a toolbox of techniques, because of the inconsistent availability of static analysis tools for different programming languages.

Confounding factors are another concern. The subject applications for our initial studies were selected to avoid domain and project bias. We will investigate domain influences by expanding the set of subject applications to include subsets of applications from particular domains. The influence of project specific factors is to be investigated in longitudinal study of Eclipse.

The practical contribution of this research is expected to be 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 improving the development process, rather than overloading the programmer with new rules to remember. Consequently, we intend to implement our future findings as an IDE plugin that would draw the programmer's attention, in real time, to identifier flaws, ranging from failure to follow typographical conventions to more complex issues of grammatical structure. Any resulting plugin 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.

5. REFERENCES

[1] R. P. Buse and W. R. Weimer. A metric for software readability. In *Proc. Int'l Symp. on Software Testing and Analysis*, pages 121–130. ACM, 2008.

[2] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp. Is identifier readability related to software quality? In *Proc. of the 5th Work in Progress Meeting of the Psychology of Programming Interest Group*, 2009.

[3] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp. Relating identifier naming flaws and code quality: an empirical study. Submitted to WCRE'09, 2009.

[4] B. Caprile and P. Tonella. Restructuring program identifier names. In *Proc. Int'l Conf. on Software Maintenance*, pages 97–107, 2000.

[5] F. Deissenboeck and M. Pizka. Concise and consistent naming. In *Proc. 13th Int'l Workshop on Program Comprehension*, pages 97–106, 2005.

[6] FindBugs. Find Bugs in Java programs. `http://findbugs.sourceforge.net/`, 2008.

[7] D. Lawrie, H. Feild, and D. Binkley. Quantifying identifier quality: an analysis of trends. *Empirical Software Engineering*, 12(4):359–388, 2007.

[8] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What's in a name? A study of identifiers. In *14th IEEE Int'l Conf. on Program Comprehension*, pages 3–12, 2006.

[9] B. Liblit, A. Begel, and E. Sweetser. Cognitive perspectives on the role of naming in computer programs. In *Proc. 18th Annual Psychology of Programming Workshop*. Psychology of Programming Interest Group, 2006.

[10] V. Rajlich and N. Wilde. The role of concepts in program comprehension. In *Proc. 10th Int'l Workshop on Program Comprehension*, pages 271–278, 2002.

[11] P. A. Relf. Achieving software quality through identifier names. 2004. Presented at Qualcon 2004 `http://www.aoq.asn.au/conference2004/conference.html`.

[12] Sun Microsystems. Code conventions for the Java programming language. `http://java.sun.com/docs/codeconv`, 1999.