Relating Identifier Naming Flaws and Code Quality: an empirical study

Simon Butler, Michel Wermelinger, Yijun Yu and Helen Sharp
Centre for Research in Computing, The Open University, UK

***Abstract*—Studies have demonstrated the importance of good identifier names to program comprehension. It is unclear, however, whether poor naming has other effects that might impact maintenance effort, e.g. on code quality. We evaluated the quality of identifier names in 8 established open source Java applications libraries, using a set of 12 identifier naming guidelines. We found statistically significant associations between flawed identifiers (i.e. violating at least one guideline) and code quality issues reported by FindBugs, a static analysis tool.**

***Keywords*-programming; software metrics; software quality;**

I. Introduction

Identifiers are the primary means by which source code authors communicate concepts to their readers, and are a key mechanism by which the readers of source code, such as maintenance programmers, access and understand source code [1], [2].

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 identifier. However, only limited advice is given on matters such as the length of identifiers [3], [4].

That poor identifier names are barriers to source code comprehension, is sufficient reason to create good quality identifiers. However, might there be other consequences of poor quality identifier names? Given the importance of the natural language content and structure of identifier names to the readability of source code [1], [2], we hypothesise that a relationship exists between software quality and the quality of identifier names used in the source code.

We evaluated the quality of identifiers extracted from established open source Java projects using a set of identifier naming style guidelines. The distribution of identifiers not conforming to the guidelines was then related to the distribution of code quality warnings reported by a static analysis tool, and the nature of any associations between the two subjected to statistical analysis. We chose to work with Java in order to build upon existing tools and related research on naming quality and source code readability, undertaken with Java source code [5], [6].

II. Related Work

Previous work focuses on the contribtion to readability and program comprehension made by the semantic content and typographical structure of identifiers [1], [7], [2], [6]. Identifiers are a significant source of domain concepts in program comprehension [2]. Lawrie et al. found identifier names composed of dictionary words are more easily recognised and understood than those composed of abbreviations, or single letters [7]. And Deissenboeck and Pizka developed a formal model of identifier name semantics with the intention of increasing source code clarity [1].

Relf derived a set of cross-language identifier naming style guidelines from the programming literature, linking the use of typography and natural language content, and investigated their acceptance by programmers in an empirical study [6].

There is little work exploring the possible connections between identifier naming and software quality. Boogerd and Moonen found compliance with some coding standards may reduce software quality, but that specific standards, including those related to naming, had inconsistent effects in both applications investigated [8].

Buse and Weimer [5] developed a readability metric for Java and found correlation between the readability of methods in open source programs and the presence of defects found by FindBugs [9]. Although they demonstrated a link between readability and software quality, their notion of readability excludes the quality of identifier names.

In summary, a review of the literature has shown that although the need for good identifier names has been argued and their impact on program comprehension has been studied in various ways, their effect on code quality is yet largely unknown. This paper thus aims to provide a step in that direction.

III. Identifier Quality

Identifier name quality is multifactorial. The use of typography, as defined in programming conventions [3], gives the reader cues to the role of each identifier. However, typography alone is insufficient; a good identifier name should clearly communicate the concept represented [1], and its function through the use of natural language.

Relf developed twenty-one naming style guidelines for both Ada and Java [6]. The guidelines focus on the typography and length of identifiers, and include the distillation of practical advice from the programming literature, which is often absent from programming conventions. Relf's guidelines do not deviate significantly from Java identifier naming conventions [3], [4]. However, in contrast to programming

Table I
THE IDENTIFIER NAMING STYLE GUIDELINES APPLIED

Name	Description	Example of flawed identifier(s)
Capitalisation Anomaly	Identifiers should be appropriately capitalised.	`HTMLEditorKit`, `pagecounter`
Consecutive Underscores	Consecutive underscores should not be used in identifier names.	`foo__bar`
Dictionary Words	Identifier names should be composed of words found in the dictionary and abbreviations, and acronyms, that are more commonly used than the unabbreviated form.	`strlen`
Excessive Words	Identifier names should be composed of no more than four words or abbreviations.	`floatToRawIntBits()`
Enumeration Identifier Declaration Order	Unless there are compelling and obvious reasons otherwise, enumeration constants should be declared in alphabetical order.	`enum Card {ACE, EIGHT, FIVE, FOUR, JACK, KING ...}`
External Underscores	Identifiers should not have either leading or trailing underscores.	`_foo_`
Identifier Encoding	Type information should not be encoded in identifier names using Hungarian notation or similar	`int iCount;`
Long Identifier Name	Long identifier names should be avoided where possible.	`getPolicyQualifiersRejected`
Naming Convention Anomaly	Identifiers should not consist of non-standard mixes of upper and lower case characters.	`FOO_bar`
Number of Words	Identifiers should be composed of between two and four words.	`ArrayOutOfBoundsException`, `name`
Numeric Identifier Name	Identifiers should not be composed entirely of numeric words or numbers.	`FORTY_TWO`
Short Identifier Name	Identifiers 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`	`name`

conventions, Relf's naming style guidelines have been evaluated empirically.

The combination of typography and a simple approach to language content enable Relf's naming guidelines to be applied as rules to evaluate identifier name quality. Table I provides descriptions of the guidelines applied in this study, which were selected because they concern the structure of identifier names, and can be measured objectively.

Where necessary we refined some guidelines to increase their clarity.

Short Identifier Name: We updated Relf's guideline to include more single letter and short identifiers commonly used in Java [3], [4], see Table I.

Dictionary Words: We defined a dictionary word as belonging to the English language because all the projects investigated are developed in English. We constructed a dictionary consisting of some 117,000 words, including inflections and American and Canadian spelling variations, using word lists from the SCOWL package up to size 70 [10], and added ca. 90 common computing and Java terms, e.g. 'arity', 'hostname', 'symlink', and 'throwable'. A separate dictionary of abbreviations was constructed, using the criterion that "the abbreviation is much more widely used than the long form, such as URL or HTML" [3].

Capitalisation Anomaly: We test for capitalisation of the initial letter of abbreviations and acronyms only, where appropriate [3], [4]

Number of Words: The Number of Words guideline has a wide scope (see Table I). We created a new guideline, named Excessive Words, which defines identifiers composed of more than four words as flawed, to provide a sharper focus on linguistically complex identifiers.

IV. CODE QUALITY

Though Relf's guidelines were developed for Java and Ada source code, we decided to conduct our study on Java programs to build on related work [5], [6].

Our interest is to measure the quality of source code in a way that reflects programmers' errors. We decided to evaluate code quality with FindBugs [9], a static analysis tool for Java. FindBugs examines Java bytecode for 'bug patterns'. Some defects found by FindBugs can result in observable, aberrant runtime behaviour, while others are related to the maintainabilty of source code [11]. Many of the defects can be attributed to programmer error and misunderstanding of Java language concepts. We are also able to investigate whether the relationship between readability and FindBugs warnings, established by Buse and Weimer [5], holds with a measure of readability based on identifier quality instead of code layout.

FindBugs classifies defects by their potential impact, and in categories reflecting, for example, poor practice, correctness and security concerns. We investigated the relationships between flawed identifiers and the two highest priorities of FindBugs warnings.

Like other static analysis tools, FindBugs has a non-negligible rate of false positives [11]. In May 2009, Google engineers reviewed 4,000 quality issues reported by FindBugs on Google code [9]. Over three quarters of the reviews stated that the reported issue should be fixed. Given this finding and the reasons mentioned above, we believe that while not perfect, FindBugs provides an adequate assessment of code quality for our purposes.

V. Methodology

We selected a total of 8 established Java open source applications and libraries for investigation from a variety of domains and uses including GUI applications, programmers' tools, and charting and persistence libraries. The variety of projects chosen reduces the possibility of any unanticipated project or domain specific influence on identifier names. Table II shows the version and size of each code base analysed, in terms of number of classes and thousands of non-commenting source statements (KNCSS), as measured by FindBugs.

Source	KNCSS	Classes
Ant 1.71	72	796
Cactus 1.8.0	7	128
Freemind 0.9.0 Beta 20	36	404
Hibernate Core 3.3.1	67	1145
JasperReports 3.1.2	76	1140
jEdit 4.3 pre16	58	483
JFreeChart 1.0.11	61	582
Tomcat 6.0.18	114	1019

Table II
Source Code Analysed

We developed a tool to automate the extraction and analysis of identifiers from Java source code. Java files were parsed and identifiers analysed on the parse tree to establish adherence to the typographical rules for their context, e.g. method names starting with a lowercase character. Then, identifiers were extracted and added to a central store, with information about their location, and divided into their component words, and abbreviations, relying on the conventional Java word boundaries of internal capitalisation and underscores. Identifiers were then analysed by the tool for conformance to Relf's guidelines in Table I and our own Excessive Words guideline.

Where subject applications were found to contain source code files generated by parser generators, or to incorporate source code from third party libraries, those files were ignored to try to ensure only source code written by the applications' development team was analysed.

The Java archive (JAR) files resulting from the compilation of the source code were analysed with FindBugs, and counts of the 'priority one' and 'priority two' warnings were recorded for each class.

The identifier naming data collected for each Java class was stored in XML files and then collated with the XML output of FindBugs, using a tool we developed. Data extracted from the source code was matched with classes recorded by FindBugs to ensure that only identifiers from source code compiled into the JAR files were analysed. The collated data for each class and package was then written to R [12] dataframes for statistical analysis.

A. *Statistical Analysis*

Preliminary analysis of the results showed that FindBugs warnings were reported for a minority of classes. Similarly, many of the identifier flaws were found in a minority of classes. Given the absence of a normal distribution of warnings and flawed identifiers, we used the non-parametric chi-squared and Fisher Exact tests to determine the existence of any association between the presence of FindBugs warnings and identifier flaws in classes.

Contingency tables were generated using R for each relationship investigated. Expected frequencies were calculated from the contingency tables and the appropriate test selected: the Fisher Exact test where at least one expected frequency is less than 5, and the chi-squared test with Yates' continuity correction otherwise. For each test where the p-value was less than 0.05, the observed and expected frequencies were compared to establish the nature of the association.

VI. Results

The identifier flaws Consecutive Underscores and Enumeration Identifier Declaration Order were excluded from the statistical analysis because the former was not found in any of the code bases, and the latter only very rarely. Thus, we present the results of our statistical analysis in Tables III for 9 of Relf's guidelines and our own Excessive Words guideline.

Table III shows the statistically significant associations between identifier flaws and FindBugs warnings in dark grey for $p < 0.05$ and black for $p < 0.001$, the absence of any significant association (i.e. $p > 0.05$) in light grey, and blank areas show the absence of the identifier flaw[1]. For each statistically significant relationship, the trend was checked and the observed frequency of the occurrence of identifier flaws and FindBugs Warnings together in classes was always greater than expected by chance, with one exception (marked with '–'), which we discuss later.

[1]Full details of the χ^2 values can be found at http://www.facetus.org.uk/conferences/WCRE09/

Table III
Associations Between Naming Flaws and Priority One and Two Warnings

	Priority One Warnings								Priority Two Warnings							
	Ant	Cactus	Freemind	Hibernate	JasperReports	jEdit	JFreeChart	Tomcat	Ant	Cactus	Freemind	Hibernate	JasperReports	jEdit	JFreeChart	Tomcat
Capitalisation Anomaly	░	▩	░	░	░	░	■	■	■	■	■	■	▩	▩	▩	■
Dictionary Words	░	▩	▩	░	░	░	■	■	■	■	■	■	░	■	▩	■
Excessive Words	▩	▩	░	░	░	░	░	■	■	░	■	■	▩	■	■	■
External Underscores	░		▩			░		░	░		■			░		▩
Identifier Encoding	░		░		░	░	░	░	░		░		░	░	░	░
Long Identifier Name	▩	░	░	▩	░	░	░	■	■	░	▩	■	▩	▩	■	■
Naming Convention Anomaly	▩	░	░	░	░	░	░	▩	░	░	■	░	▩	▩	■	■
Number of Words	░	░	░	░	░	░	░	▩	▩	░	▩	▩	■	░	▩	▩
Numeric Identifier Name	░			░	░		░	░	░			▩	▩ –		░	░
Short Identifier Name	░	░	░	░	░	░	▩	▩	░	░	░	■	■	░	░	■

■	$p < 0.001$	▩	$p < 0.05$	░	$p > 0.05$		No flaw

Table III shows that associations between priority one warnings and identifier flaws are less common than the more consistent associations for priority two warnings. The Capitalisation Anomaly, Dictionary Words, Excessive Words and Long identifier Name flaws are each associated with priority two warnings in at least seven of the eight code bases.

Statistically significant relationships between identifier flaws and defects for Cactus in table III are constrained to the Capitalisation Anomaly, Dictionary Words and Excessive Words flaws. A possible explanation is that the Cactus development team use CheckStyle [13] to ensure committed code conforms to the project's detailed coding conventions [14]. Their development methodology reduces the number of flawed identifiers; however, our tool analyses capitalisation more rigorously than CheckStyle, and performs checks for the length of identifiers and natural language content, which CheckStyle does not.

The only statistically significant negative association we found (marked '–') is between Numeric Identifier Name flaws and priority two defects for JasperReports. Examination of the extracted identifiers found only the constants `ZERO` and `ONE` composed of numeric words alone, and that the former is used in 50 classes without FindBugs warnings.

A. Threats to validity

The Short Identifier Name guideline applied in this study differs in some regards from both the advice available to programmers and praxis. Consequently, the number of identifiers categorised as flawed under this guideline could be inflated, with implications for the reliability of observed statistical associations.

FindBugs, like other static analysis tools, reports false positives [11]. Without inspecting the source code for all warnings, we cannot know the false positive rate. Another concern is that FindBugs may have been applied to the subject code bases. In response to our enquiries, the developers of Freemind, jEdit and JFreeChart have indicated that they do not use FindBugs systematically, if at all.

VII. Discussion

Table III shows that priority one warnings often occur independently of identifier flaws. This suggests programmers are capable of making significant errors irrespective of the degree of adherence to naming conventions. However, that strong associations often exist between priority two warnings and particular identifier flaws indicates that connections exist between the use of low quality identifiers and less serious FindBugs defects.

It is unsurprising that the statistically significant relationships are not consistent across the subject code bases. Identifier names and software defects are artefacts of human behaviour, which have many influences, such as the constitution of development teams and working practices.

Boogerd and Moonen [8] attributed the differences they found between the influence of particular coding standard guidelines on the code quality of two projects to domain factors. We deliberately selected our subject projects from a variety of domains to avoid any possible domain bias in our results. Consequently we can not comment on the influence of domain factors on our results.

That the association between identifiers constructed of non-dictionary words and priority two defects is statistically significant in all but one of the subject code bases (JasperReports, see Table III) is notable because empirical studies [7] have shown that the use of dictionary words makes identifiers easier to read and understand.

VIII. Conclusions

Identifier names are crucial components of source code, which impact on program comprehension. As artefacts of the programmers' thought processes, identifier names are a 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, we are not aware of work that directly relates identifier naming and code quality. Studying such a relationship is useful to help 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. This paper provides a first step in that direction.

To assess identifier naming quality, we adopted (and slightly adapted) a set of 11 typographic and natural language naming guidelines for Java [6] because they had been empirically evaluated and are more detailed than other proposals in the literature. To assess code quality, we used FindBugs due to its coverage of quality issues that may lead to faulty application behaviour, as perceived by the user, and to greater maintenance efforts by the developers.

We developed an automated tool to check deviations from such guidelines against warnings reported by FindBugs, and applied it to 8 established Java applications and libraries. We found statistically significant associations between flawed identifier names and FindBugs warnings, even where the developers had used tools to detect programming convention violations, as in the case of Cactus. Associations for the potentially more serious priority one warnings are uncommon and appear to be largely application specific. It appears that, generally, programmers can make more significant mistakes regardless of the quality of identifiers. However, possibly attributable to the development methodologies or naming conventions used, particular types of identifier flaw may indicate the presence of more serious software defects. More widespread, statistically significant associations were found between priority two warnings and some identifier flaws. Some associations were common to a majority of the subject applications. Although the associations we found were not universal, a selection of identifier naming flaws could be used as a diagnostic toolkit, with an understanding of the development methodology and naming conventions used in a given project.

Acknowledgements

We thank Álvaro Faria, coordinator of the Statistics Advisory Service at The Open University, for his help in choosing the statistical method.

References

[1] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, Sep 2006.

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

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

[4] 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.

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

[6] P. A. Relf, "Achieving software quality through identifier names," 2004, presented at Qualcon 2004 http://www.aoq.asn.au/conference2004/conference.html.

[7] 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*, 2006, pp. 3–12.

[8] C. Boogerd and L. Moonen, "Evaluating the relation between coding standard violations and faults within and across software versions," in *Proc. 6th Int'l Working Conf. on Mining Software Repositories*. IEEE, 2009, pp. 41–50.

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

[10] K. Atkinson, "SCOWL readme," http://wordlist.sourceforge.net/scowl-readme, 2004.

[11] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, "Evaluating static analysis defect warnings on production software," in *Proc. ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, 2007, pp. 1–8.

[12] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, 2008, ISBN 3-900051-07-0.

[13] O. Burn, "Checkstyle," http://checkstyle.sourceforge.net/, 2007.

[14] Apache Software Foundation, "Jakarta Cactus - coding conventions," http://jakarta.apache.org/cactus/participating/coding_conventions.html, 2008.