A Naming Convention for Self-Describing Code

Luke Plewa, *Student, Cal Poly San Luis Obispo*

Abstract—Coding styles help collaborators communicate with code effectively. This is promoted through uniformity and ease of reading. Naming conventions are a subset of coding styles that define rules for naming identifiers within code. This paper investigates specific naming conventions in open source C code collected across a variety of authors and applications. That investigation defines a publicly available corpus specifically for C naming conventions and reports upon the features of that corpus. The other contribution of this paper is a naming convention whose rules are defined by the populus.

Keywords—Naming Convention, Coding Style, Self-Describing Code, Software Engineering, Machine Learning

1 Introduction

THIS paper serves the public by describing a newly defined corpus for C-language naming conventions and a naming convention for self-describing code defined by the populus. At the time of this paper, there is no existing corpus for C-language naming conventions. The one described in this paper is the first of its type. The naming convention defined in this paper is supported through the data collected on that corpus.

December 11, 2014

2 Background

2.1 Naming Conventions

Naming conventions are a set of rules used for identifiers in software engineering [10]. Variables, classes, functions, methods, etc. are all examples of identifiers. Some naming convention rules include restricting the length of the identifier, case (snake_case or camelCase), and use of abbreviations or dictionary words. Good naming conventions are useful because they make code easier to read and convey more information about the identifier.

• *L. Plewa is with the Department of Computer Science, Cal Poly San Luis Obispo, San Luis Obispo, CA, 93405.*
E-mail: lplewa@calpoly.edu

Manuscript received December 11, 2014.

2.2 Code Maintenance

Maintenance of code is determined to be about 60% of work involved in the project lifecycle [5]. One of the biggest contributors to reducing code maintenance is coding styles and more specifically naming conventions for identifiers. Bad naming conventions can make code maintenance costly [12]. By creating a better understanding of naming conventions, there is an opportunity to improve software engineering as a whole.

2.3 Self-Describing Code

Self-describing or self-documenting code is a coding style that prefers the use of programming style as a form of documentation [9]. While the self-describing code principle does not promote the use of comments or external documentation, there is room for programmers to use both. The end-goal is to help others understand the code. One of the key features of self-describing code is good idenfitier names. More specifically, the name of the identifier should be able to describe the purpose of the identifier.

3 Related Work

3.1 Popular Coding Convention on GitHub

”Popular Coding Convention on GitHub” [3] is a project done through ”The GitHub Data Challenge II” [4]. This project breaks down various coding styles across various languages.

The languages supported are Javascript, Java, Python, Scala, Ruby, C#, PHP. Some coding styles that are examined include the use of spaces versus tabs, curly brace placement in block statements, the use of capital letters in constants, line length, use of whitespace, and argument definition spacing. Other language specific coding styles are also examined. The project looks at a specific set of projects on GitHub, presented by the data challenge, and provides a percentage breakdown of those projects in terms of each coding style.

This project notably leaves out the C programming language, which is the focus of this paper. There is also no mention of naming coventions.

3.2 Mining Java Class Naming Conventions

Another notable project examines the naming conventions of Java classes [2]. The authors looked at 60 open source Java projects. They identified 120,000 unique class names across those projects. They found that the class identifiers often share a substring with their parent or children classes. They also do identification on poor naming patterns. Unfortunately, the authors do not examine all identifier types (notably variables and methods).

3.3 Restructuring Source Code Identifiers

An extremely relevant paper [5] attempted to discover what identifier features correlate with high confusion in readers. They found that fault proneness has a high relation to high term entropy and high context-coverage.

Fault proneness is the potential for a segment of code to be confusing, and is measured by third-party applications Rhino and ArgoUML. Term entropy increases with probability distribution, so if an identifier is used often then it has high term entropy. Context-coverage increases as an identifier is used in different contexts. If a term is only used in loops (a specific type of context) then it has low context-coverage. The authors determined that an identifier must be rarely used and only within a few number of contexts in order for it to produce minimal confusion in readers. The authors do not look into other features about identifiers that are specific to naming conventions.

4 Corpus

4.1 Repositories

Seven repositories are pruned and parsed using regular expressions on December 2014. These seven repositories include the implementations for git (a version control system), sci-kit learn (a machine learning library for python), PHP (the language interpreter), redis (an in-memory database), Arduino (an application for electronics prototyping), macvim (a text editor specifically for mac), and dynomite (a database implementation based on the Dynamo whitepaper) [4]. These repositories feature a wide range of applications. The corpus utilizes the code written by over 1,603 authors and 17,152 uniquely named identifiers.

4.2 Type Breakdown

The breakdown of types of identifiers is displayed in both Fig. 1 and Fig. 2.

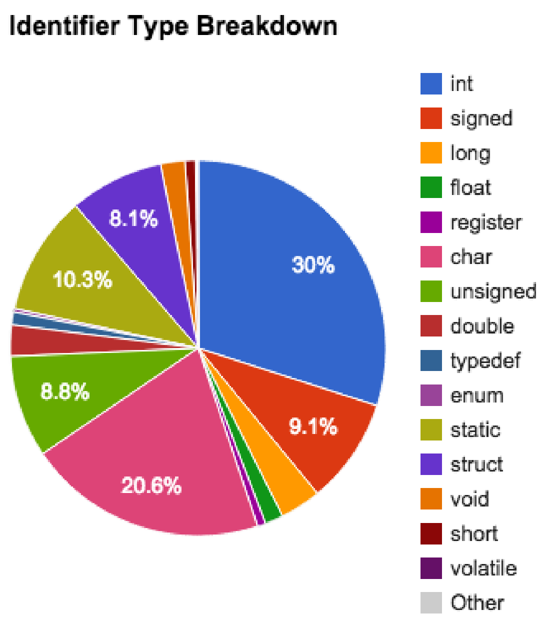
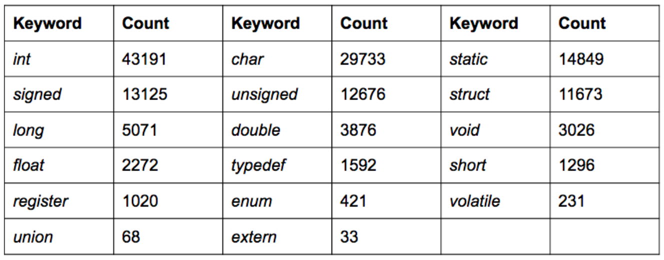


Fig. 1. Identifier Type Breakdown Pie Chart.

This breakdown gives some information about the corpus. The most notable type is the int datatype, as it is highly prevalent with 43,191 cases (non-unique). The next most common identifier type is char with 29,733 cases.



Keyword	Count	Keyword	Count	Keyword	Count
int	43191	*char*	29733	*static*	14849
signed	13125	*unsigned*	12676	*struct*	11673
long	5071	*double*	3876	*void*	3026
float	2272	*typedef*	1592	*short*	1296
register	1020	*enum*	421	*volatile*	231
union	68	*extern*	33		

Fig. 2. Identifier Type Breakdown Table.

Static identifiers are also very common. Interestingly, there is an almost even split between signed and unsigned types.

4.3 Parts of Speech

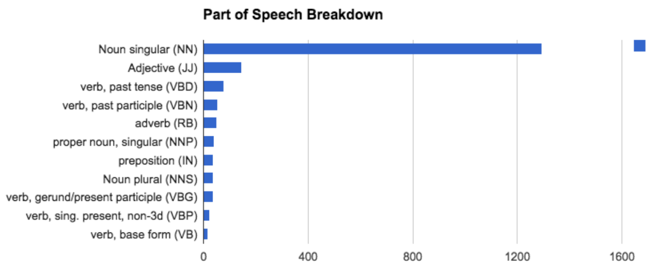
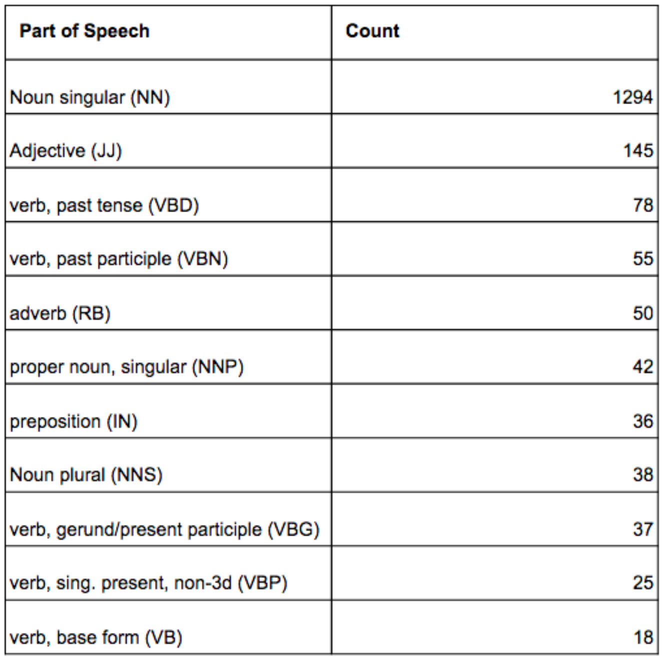


Fig. 3. Parts of Speech Breakdown Chart.



Part of Speech	Count
Noun singular (NN)	1294
Adjective (JJ)	145
verb, past tense (VBD)	78
verb, past participle (VBN)	55
adverb (RB)	50
proper noun, singular (NNP)	42
preposition (IN)	36
Noun plural (NNS)	38
verb, gerund/present participle (VBG)	37
verb, sing. present, non-3d (VBP)	25
verb, base form (VB)	18

Fig. 4. Parts of Speech Breakdown Table.

The NLTK corpus of words [1] is used to determine the existence of unique dictionary words combined with the use of regular expressions during parsing. Fig. 3 and Fig. 4 displays the breakdown of parts of speech (PoS) tags within the corpus. This breakdown is determined through NLTK's PoS tagger. These figures omit tags that were featured less than 10 times.

Singular nouns are clearly the most common, with adjectives being a significantly less common second. The most noticeable breakdown here is the different between the types of verbs. Of all unique verbs, the past tense and past-participle forms are much more common (about twice) as the present participle, base, and present tense forms.

4.4 Case

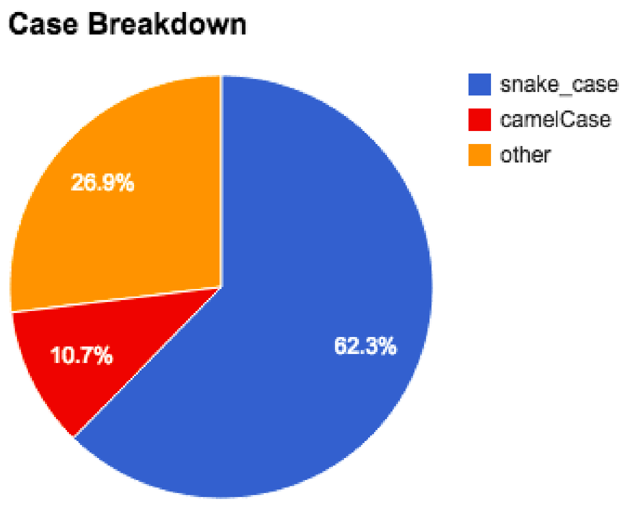


Fig. 5. Case Breakdown Table.

The corpus features 17,152 unique identifiers. Of these identifiers, 1,843 are camelCase and 10,691 are snake_case. This shows a heavy favoritism across authors towards snake_case. The remaining identifiers that are not counted did not show features of either style. This breakdown is displayed in Fig. **??**

4.5 Dictionary Words

The total number of dictionary words in this corpus amounts to 10,886. The average length of each dictionary word is 5.79. The longest word in this corpus, authentication, is 14 characters. Of all identifier, 65.08

5 Implementation

5.1 Naive Bayes Classifier

The rest of this paper is focused on the use of this corpus to show trends in identifiers based on the repository. It is proposed that naming conventions may be identifiable based on the corpus. The NLTK implementation of a Naive Bayes classifier is used [1]. This is based on the Bayes' Theorem which assumes that specific events (the classes) are statistically independent.

5.2 Features

Numerous features are used in this implementation. For each identifier, a list is created of binary and numeric values for each feature. The binary existence of a keyword type used on each identifier (int, char, etc.) composes part of this feature list. The binary existence of a specific case (snake_case, camelCase, and CONSTANT) is also part of this feature list. The total count of dictionary words found within the identifier is included. Also the specific count of each part of speech tag (NN, JJ, etc.) is included for each tag.

6 Results

6.1 Distribution

The Naive Bayes classifier [1] is used to determine how many repositories an identifier is part of. This shows that the previously described features have some way of representing the distribution of an identifier. There are seven possible classes in this case, as there are seven repositories. This creates a 17% baseline of randomly guessing the number of repositories.

Two experiments are ran, one with a 50/50% training and testing dataset split and another with a 66-33% training and testing dataset split. Each experiment is ran five times and the reported accuracies are an average of those five runs. The identifiers included in each dataset are created randomly. For a 50/50% split, the average accuracy is 88.39%. For a 66/33% dataset split, the average accuracy is 91.16%. These high accuracy (compared to the baseline) determine that there is a strong relationship between the defined features and the commonality of these identifiers.

6.2 Repository Identification

The next set of experiments try to name the specific repositories that an idenfier is found in. This creates 5,040 unique classes based on the different combinations of these repositories. For example, on class may be just the git repository, and another may be the git repository and the sci-kit learn repository. This experiment is run five times, and a 50/50% training and testing dataset split is used. The averaged accuracy is 36.54%. This accuracy demonstrates that there is a strong relationship between these features and the specific distribution and authors of each repository. It is notable that the distribution may not be even, as some common identifier are included across all seven repositories and other identifiers are less common and are only included in one repository.

6.3 Proposed Naming Convention

This section defines the proposed naming convention based on the data acquired from open source C language code and a variety of authors. The ruleset dictates that snake_case should be used over other cases. In support of the self-describing style, abbreviations and short identifiers are not allowed. This is slightly supported by the corpus. The corpus strong supports the use of adjectives and singular, non-proper nouns. The corpus also supports the use of paste tense and paste participle verbs over the use of all other forms of verbs. Lastly, the ruleset for this naming convention dictates that identifiers should strive to use dictionary words around six characters in length, and that all identifier should be under 14 characters.

7 Future Work

There is significant room for growth in understanding naming conventions.

Due to the limitations of the available resources, the corpus proposed here is not as large as it could be. There are several linux distributions which should be included in this corpus, but are enormous and had to be left out.

There is also a need for individual authorship identification not just repository identification. The GitHub API [4] supports this, and there is a lot of previous work in this field [11] [6] [7].

The work done in fault tolerance [5] could be cross-referenced with the naming convention features described in this paper. It would be interesting to see how these features, such as lack of dictionary words and length, trend with fault tolerance.

Futher work could be done in parts of speech n-grams. What is highly likely is the bigram of adjective and singular, non-plural noun for identifiers. Other n-grams may be common and it would be beneficial to see what those are.

An obvious extension of this work would be to create more corpora based on other languages and determine a naming convention based on those corpora. These corpora could also be used to determine other features and do feature analysis to determine which ones are particularly relevant.

8 Conclusion

The corpus defined in this paper is the collective work of 1,603 unique authors. Together they have helped define a new naming convention for the C programming language that supports the self-describing style of coding. The most notable find of this corpus is the high use of past tense and past participle verbs over all other verb forms. A Naive Bayes classifier is used to determine that the features identified in this paper have a high correlation to specific repositories.

Acknowledgments

The author would like to thank Dr. Foaad Khosmood for teaching Natural Language Processing (CSC 582) at Cal Poly San Luis Obispo. Also thanks to the Computer Science Department at Cal Poly San Luis Obispo, whose facilities and administration made this possible. The author would also like to thank GitHub and the programmers who contributed to the corpus.

References

[1] Bird, Steven. ”NLTK: the natural language toolkit.” Proceedings of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics, 2006.

[2] Bulter, Simon, et. al. ”Mining Java class naming conventions.” Software Maintenance (ICSM), 2011 27th IEEE International Conference on. IEEE, 2011.

[3] Byun, JeongHoon, et. al. ”Popular Convention.” 2013. Available: http://sideeffect.kr/popularconvention/

[4] Doll, Brian. ”The GitHub Data Challenge II.” GitHub. 2013.

[5] Eshkevari, Laleh Mousavi. ”Restructuring Source Code Identifiers.” 2010.

[6] Kilgour, R. I., et al. ”A fuzzy logic approach to computer software source code authorship analysis.” (1998).

[7] Kothari, Jay, et al. ”A probabilistic approach to source code authorship identification.” Information Technology, 2007. ITNG’07. Fourth International Conference on. IEEE, 2007.

[8] Kuhn, Adrian, Stphane Ducasse, and Tudor Grba. ”Semantic clustering: Identifying topics in source code.” Information and Software Technology 49.3 (2007): 230-243.

[9] McConnell, Steven C. ”Code Complete.” 2003.

[10] Oracle. ”Naming Conventions.” Sun Microsystems, 1999.

[11] Spafford, Eugene H., and Stephen A. Weeber. ”Software forensics: Can we track code to its authors?.” Computers and Security 12.6 (1993): 585-595.

[12] Wilde, Norman, and Ross Huitt. ”Maintenance support for object oriented programs.” Software Maintenance, 1991., Proceedings. Conference on. IEEE, 1991.

Luke Plewa Luke Plewa is a student at Cal Poly pursuing a joint BSMS degree for Computer Science. His other works include progress made in computer architecture, OpenGL video games, Android applications, and machine learning.