An Empirical Comparison of Java and C# Programs in Following Naming Conventions

Shouki A. Ebad, Faculty of Computing and IT, Northern Border University, Arar, Saudi Arabia
Danish Manzoor, Northern Border University, Arar, Saudi Arabia

ABSTRACT

An important indicator of source code quality is compliance with naming conventions. It is believed that such practices improve program comprehension, which directly affects maintainability and reusability. In this paper, the authors conduct an experiment to determine how well Java and C# programs follow a set of well-publicized naming practices. The experiment evaluated 120 arbitrarily selected open-source Java and C# classes from different programmers with respect to four naming conventions. The results indicate that Java and C# programs do not always follow naming conventions. However, Java developers are more attentive than C# developers in terms of following naming practices. A disturbing trend was found in variable and constant naming conventions, which were violated in most C# subjects. Moreover, there is a positive correlation between the number of violations found in a C# class and its size but a negative correlation in case of Java class. The findings are expected to contribute to the existing knowledge of the use of coding standards and source code quality. The paper also discusses the threats to the validity of the study and suggests open issues for future research.

KEYWORDS
C#, Code Style, Coding Standards, Java, Naming Conventions

1. INTRODUCTION

An important indicator of code quality is compliance with naming conventions (interchangeably referred to as naming style or naming guidelines). Usually, these conventions consist of a set of programming practices or rules such as “class name should start with a capital C” or “do not use underscore with the method/function name”. With modern programming languages, developers’ compliance with these conventions is optional in the sense that programs that violate the conventions can still be functionally and syntactically correct. However, compliance with naming conventions is essential to ensure that software is understandable and readable and thus more likely to be reusable and maintainable (Elish and Offutt 2001; Galin, 2004). From the perspective of the software industry, developers are encouraged to apply such conventions because software is rarely maintained or reviewed for its whole life by the original author, especially within the open source model. In fact, names in the programs represent defined concepts because they connect the source code to the problem domain.
This is important because every code reader may infer different meanings from the code depending on naming and other conventions, despite the architecture and design of the code (Green and Ledgard 2011).

The consequences of violating these conventions in one place would lead to damaged program readability in numerous other places; thus, there is a real comprehension cost when names are not chosen carefully (Butler et al. 2009; Butler et al. 2010). As an example, the meaning of a method that returns the variable `detect` may take some time to understand. Using more words and underscores (e.g., `detect_bad_sector`) would remove the violation and clarify the method’s objective. Another example that highlights the importance of naming conventions comes from the variable `lcf_name`. This may also take considerable time to identify unless we discover that the developer prefixes the variable names with a tag that indicates the type of data and scope they hold. For example, `lcf_name` indicates that the `f_name` variable has local scope (l) and character data type (c). With such a notation (called Hungarian (Wang et al. 2014)), it becomes much worse once we know that any programming language has several data types and different scopes. Compared with the past, making the program names easily understandable has become a necessity because programming currently is a team-based activity rather than an individual experience. Several studies have noted the importance of careful naming (Caprile and Tonella 2000; Butler et al. 2009; Lawrie et al. 2006; Rilling and Klemola 2003).

To the best of our knowledge, there is no study available to determine how well code complies with the language naming conventions. In this study, we conducted an experiment to fill this gap empirically for the Java and C# programming languages. Our choice of these languages resulted not only from their widespread use but also from their early attention to the naming convention issue; the founders of the languages (Microsoft for C# and Sun Microsystems/Oracle for Java) defined a single naming guideline for the entire programming language (Sun Microsystems 1999; Pradeep 2008; Jasonall 2008; DoFactory website). Herein, 120 arbitrarily selected open source Java and C# classes were evaluated with respect to naming conventions. The next section describes the naming conventions of Java and C# that were studied. Section 3 describes the design and planning of the experiment. The experimental results are analysed and discussed in section 4. Section 5 describes the study’s limitations and the threats to its validity. In section 6, we present an overview of the related work in this area. A conclusion and plans for future work are presented in section 7.

### 2. NAMING CONVENTION

One of the challenges of measuring compliance with naming conventions is that there are different styles in use. Most software engineers would agree that following some style is important; the specific style followed is secondary (Elish and Offutt 2001). This makes our task a costly one. Fortunately, the naming guidelines for the modern programming languages have been developed and are used by the software development community (Wang et al. 2014; Wiltamuth and Hejlsberg 2003). The experiment in this study uses four naming conventions that can be measured by automated tools. In particular, we were able to automatically collect data using the PMD and StyleCop tools. The naming conventions are class naming (CN1), method naming (MN), variable naming (VN), and constant naming (CN2). Table 1 defines them, in some cases with rationales. However, we know of no automated tool that distinguishes nouns from verbs for both Java and C# code; therefore, this practice is excluded from our study.

From Table 1, it is clear that both Java and C# discourage the use of underscores; Java allows underscores only with constant naming. This recommendation comes with the modern popularization of case-sensitive languages such as Java, C++, C#, and Python, so the trend has been towards the use of camel-case rather than underscores¹. With some old programming languages, it was encouraged to use underscores to separate names because those languages are case insensitive (Binkley 2013). Another observation from Table 1 relates to the method and class naming of Java and C#. The programmers are encouraged to differentiate the method name from the class name in which it is
Computational Engineering in the Cloud: Benefits and Challenges
www.igi-global.com/article/computational-engineering-cloud/58546?camid=4v1a

Framework to Enhance the Mobile User Experience of Goal Orientated Interactions
Adéle Botha (2014). Research and Design Innovations for Mobile User Experience (pp. 47-75).
www.igi-global.com/chapter/framework-to-enhance-the-mobile-user-experience-of-goal-orientated-interactions/80363?camid=4v1a