The Journal of Systems and Software 123 (2017) 160–172

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Understanding the syntactic rule usage in java

Dong Qiu[a], Bixin Li[a,*], Earl T. Barr[b], Zhendong Su[c]

[a] School of Computer Science and Engineering, Southeast University, China
[b] Department of Computer Science, University College London, UK
[c] Department of Computer Science, University of California Davis, USA

ARTICLE INFO

Article history:
Received 1 February 2016
Revised 26 September 2016
Accepted 19 October 2016
Available online 20 October 2016

Keywords:
Language syntax
Empirical study
Practical language usage

ABSTRACT

Context: Syntax is fundamental to any programming language: syntax defines valid programs. In the 1970s, computer scientists rigorously and empirically studied programming languages to guide and inform language design. Since then, language design has been artistic, driven by the aesthetic concerns and intuitions of language architects. Despite recent studies on small sets of selected language features, we lack a comprehensive, quantitative, empirical analysis of how modern, real-world source code exercises the syntax of its programming language.

Objective: This study aims to understand how programming language syntax is employed in actual development and explore their potential applications based on the results of syntax usage analysis.

Method: We present our results on the first such study on Java, a modern, mature, and widely-used programming language. Our corpus contains over 5000 open-source Java projects, totalling 150 million source lines of code (SLoC). We study both independent (*i.e.* applications of a single syntax rule) and dependent (*i.e.* applications of multiple syntax rules) rule usage, and quantify their impact over time and project size.

Results: Our study provides detailed quantitative information and yields insight, particularly (i) confirming the conventional wisdom that the usage of syntax rules is Zipfian; (ii) showing that the adoption of new rules and their impact on the usage of pre-existing rules vary significantly over time; and (iii) showing that rule usage is highly contextual.

Conclusions: Our findings suggest potential applications across language design, code suggestion and completion, automatic syntactic sugaring, and language restriction.

1. Introduction

Syntax and semantics define a programming language. Informally, a language has many features. A language's syntactic rules provide the most direct means to measure the use of a language's features. Thousands of programming languages exist; each embodies a different set of possible language features. Language designers usually have limited knowledge on how programmers actually use a language (Knuth, 1971). This leads to many unnatural and rarely used features being introduced, while expected ones not introduced (Strangest language feature, 2016; Your language sucks, 2016). In addition, many language features, especially language syntax, remain a significant barrier to novice programmers (Denny et al., 2011; Stefik and Siebert, 2013).

We tackle the question of how to systematically understand these features and their usage. Rather than ad-hoc characterizations of features, we propose the use of language grammars to precisely and systematically characterize language features. Indeed, most programming language features quite directly map onto syntactic constructs. Therefore, we study how programmers use language features by analyzing their use of the language syntax.

Knuth conducted the first study to understand how programmers use FORTRAN over 40 years ago (Knuth, 1971). Similar studies were subsequently performed on COBOL (Salvadori et al., 1975; Chevance and Heidet, 1978), APL (Saal and Weiss, 1977) and Pascal (Cook and Lee, 1982) between the 1970s and 1980s. In recent decades, there has been little quantitative study demonstrating how a modern programming language is used in practice, especially from the perspective of language syntax. Previous studies have investigated the use of subsets of language features (*e.g.*, Java generics (Parnin et al., 2011) and Java reflection (Livshits et al., 2005)). Although Dyer et al. (Dyer et al., 2014) investigated the use

* Corresponding author. Fax: 86 25 52090879.
E-mail addresses: dongqiu@seu.edu.cn (D. Qiu), bx.li@seu.edu.cn (B. Li), e.barr@ucl.ac.uk (E.T. Barr), su@cs.ucdavis.edu (Z. Su).

http://dx.doi.org/10.1016/j.jss.2016.10.017
0164-1212/

of newly-introduced features over three main language releases, they only examined a relatively small subset of language features and did not consider pre-existing features.

Studying how a large number of real-world programs use language syntax may help validate or disprove the many popular "theories" about what language features are most popular, most useful, easiest to use, *etc.* that abound in popular literature about programming and on the Internet. In addition, the gap between language features and their actual usage may guide pedagogy, giving teachers insight into how to teach a programming language in a better way. Language designers may leverage data on actual syntactic rule usage to optimize the design of languages, *e.g.* simplifying unpopular features or identifying boilerplate that could be eliminated. We will provide concrete examples when presenting our detailed study results.

To this end, we perform a large-scale empirical study on a diverse corpus of over 5,000 real-world Java projects to gain insight into how syntactic rules are used in practice. We generate abstract syntax trees (ASTs) for approximately 150 million SLoC, and tabulate and analyze the occurrences of all syntactic rules. In particular, to understand how syntax rules are used over time, we have checked out over 13,000 versions from the studied projects' revision histories to understand rule usage evolution.

We also perform depth-2 bounded nesting analysis to investigate dependent rule usage. Indeed, when using a grammar to parse a string, some nonterminals in the grammar can be reached only after another nonterminal has been traversed. For $X, Y \in N$, the set of nonterminals, and $\alpha, \beta \in (N \cup T)^*$ where T is the set of terminals, we write $X \xrightarrow{*} \alpha Y \beta$ to denote that *Ydepends* on *X*. We bound this dependency because, in the limit, all nonterminals vacuously depend on the grammar's start symbol. In this work, we consider $k = 2$ and report our dependency results for $X \xrightarrow{2} \alpha Y \beta$, as these short range dependencies are closer to the sentences that programmers write and think about and thus are better candidates for identifying idioms.

In summary, this paper makes the following contributions:

- It presents the first effort in 30 years to conduct a large-scale, comprehensive, empirical analysis of the use of language constructs in a modern programming language, namely Java;
- This work is the first to study dependent rule usage and quantify its contextual nature. This is also the first to study the evolution of rule usage over time, the adoption of new rules, and how new rules impact the usage of pre-existing ones.
- The results show that: (i) 20% of the most-used rules account for 85% of all rule usage, while 65% of the least-used rules are used $<$ 5% of the time and 40% only $<$ 1% of the time; (ii) 16.7% of the rules are unpopular and are adopted in $<$ 25% of the projects (*e.g.* `assert` statement, `labeled` statement, and `empty` statement); and (iii) for dependent rule usage, 6% of the combinations exhibit strong dependency with $>$ 50% probability.

Taken together, our results permit language designers to empirically consider whether new constructs are likely to be worth the cost of their implementation and deployment. They also identify boilerplate (*i.e.* repetitive rule usage) that new constructs may profitably replace. For example, we have observed a reduced use of anonymous class declarations, while an increased use of the enhanced-for constructs *w.r.t.* all syntactic rule usage. We believe that work like ours enables data-driven language design, analogous to how Cocke's study at IBM in the 1970s on the actual usage of CISC instructions eventually led to the RISC architectures.

Table 1
Overview and evolution of the JLSs.

Version	Release date	#Added rules	#Updated rules
JLS1	1996	115	–
JLS2	2000	4	–
JLS3	2005	12	16
JLS4	2013	1	2

Table 2
Summary statistics on the Java code corpus.

Corpus summary	
Repository	Github
# of projects	5,646
# of files	1,392,528
Lines of code	144,081,228
Project scale range (# of files)	1∼39,247
Project history range (# of years)	1∼17
Project commits range (# of commits)	1∼123, 938

2. Study design and results

This section describes our methodology in detail, with special attention given to the study subject and the research questions, followed by our general findings.

2.1. Study subject

Java syntax. To understand how programmers adopt syntax, we selected Java, a modern, mature and widely-used programming language as our research subject. Java's syntax is the set of rules defining how a Java program is written and interpreted; it is essentially a dialect of C/C++. Major releases of the Java Language Specification (JLS) track its constant evolution.

In this paper, we survey 132 syntactic rules in total, distributed in JLS1∼ JLS4[1] Gosling et al. (1996); 2000); 2005); 2013). Table 1 lists the distribution, including the release date and corresponding updates. In contrast to the study by Dyer et al. (2014), which focuses on the newly imported language syntax rules, we concentrate on the *complete* set of the syntactic rules. The details of the rules can be found online[2].

Code corpus. Our corpus is a large (around 150 million SLoC) collection of open-source real-world Java programs containing 5,646 projects retrieved from Github, one of the most popular repositories. The projects were selected based on their popularity (*i.e.* size of *watchers, stars* and *forks*). The corpus contains not only widely-used Java projects maintained by reputable open-source organizations (*e.g.* Tomcat, Hadoop, Derby from the Apache Software Foundation and JDT, PDT, EGIT from the Eclipse Foundation), but also small projects developed by novice programmers. All these projects are managed by Git, one of the most popular version control systems in the open-source community. Table 2 provides summary statistics on the corpus.

The corpus is also diverse, covering projects of different size and development history. It contains small, medium and large projects, where the number of Java files within projects ranges from 1 to 39,247. The corpus also includes projects with short, medium and long lifecycles, where their development years span from 1 to 17 and the commits with each repository range from 1 to 123,938. The corpus thus provides a wide and comprehensive range of projects on which to study the evolution of syntactic rule usage.

[1] For simplicity, JLS1, JLS2, JLS3 and JLS4 are used to represent the 1st edition, 2nd edition, 3rd edition and Java SE 7 edition of the JLS, respectively.

[2] It is available at: http://dong-qiu.github.io/papers/lang_syntax/appendix.pdf.