Impact of Programming Language Fragmentation on Developer Productivity: A Sourceforge Empirical Study

Jonathan L. Krein, Brigham Young University, USA
Alexander C. MacLean, Brigham Young University, USA
Charles D. Knutson, Brigham Young University, USA
Daniel P. Delorey, Google, Inc., USA
Dennis L. Eggett, Brigham Young University, USA

ABSTRACT

Programmers often develop software in multiple languages. In an effort to study the effects of programming language fragmentation on productivity—and ultimately on a developer’s problem-solving abilities—the authors present a metric, language entropy, for characterizing the distribution of a developer’s programming efforts across multiple programming languages. Utilizing this metric, the authors present an observational study examining the project contributions of a random sample of 500 SourceForge developers. Using a random coefficients model, the authors find a statistically (alpha level of 0.001) and practically significant correlation between language entropy and the size of monthly project contributions. Results indicate that programming language fragmentation is negatively related to the total amount of code contributed by developers within SourceForge, an open source software (OSS) community.

Keywords: Cognition, Language Fragmentation, Natural Languages, Open Source, Process Engineering, Programming Languages, Software Engineering

INTRODUCTION

The ultimate deliverable for a software project is a source code artifact that enables computers to meet human needs. The process of software development, therefore, involves both problem solving and the communication of solutions to a computer in the form of software. We believe that the programming languages with which developers communicate solutions to computers may in fact play a role in the complex processes by which those developers generate their solutions.
Baldo et al. (2005) define language as a “rule-based, symbolic representation system” that “allows us to not simply represent concepts, but more importantly for problem solving, facilitates our ability to manipulate those concepts and generate novel solutions” (pp. 240, 249). Although their study focused on the relationship between natural language and problem solving, their concept of language is highly representative of languages used in programming activities. Other research in the area of linguistics examines the differences between mono-, bi-, and multilingual speakers. One particular study, focusing on the differences between mono- and bilingual children, found specific differences in the subjects’ abilities to solve problems (Bialystok & Majumder, 1998). These linguistic studies prompt us to ask questions about the effect that working concurrently in multiple programming languages (a phenomenon we refer to as language fragmentation) has on the problem-solving abilities of developers.

In an effort to increase both the quality of software applications and the efficiency with which applications can be written, developers often incorporate multiple programming languages into software projects. Each language is selected to meet specific project needs, to which it is specialized—for instance, in a web application a developer might select SQL for database communication, PHP for server-side processing, JavaScript for client-side processing, and HTML/CSS for the user interface. Although language specialization arguably introduces benefits, the total impact of the resulting language fragmentation on developer performance is unclear. For instance, developers may solve problems more efficiently when they have multiple language paradigms at their disposal. However, the overhead of maintaining efficiency in more than one language may also outweigh those benefits. Further, development directors and programming team managers must make resource allocation, staff training, and technology acquisition decisions on a daily basis. Understanding the impact of language fragmentation on developer performance would enable software companies to make better-informed decisions regarding which programming languages to incorporate into a project, as well as regarding the division of developers and testers across those languages.

To begin understanding these issues, this paper explores the relationship between language fragmentation and developer productivity. First, we define and justify the metrics used in the paper. We discuss our selection of a productivity metric, after which we describe an entropy-based metric for characterizing the distribution of a developer’s efforts across multiple programming languages. Having defined the key terms, we then present the thesis of the paper, and describe, justify, and validate the data and analysis techniques. We then present the results of an observational study of SourceForge, an open source software (OSS) community, in which we demonstrate a significant relationship between language fragmentation and productivity. Establishing this relationship is a necessary first step in understanding the impact that language fragmentation has on a developer’s problem-solving abilities.

**PRODUCTIVITY**

According to the 1993 IEEE Standard for Software Productivity Metrics, “productivity is defined as the ratio of the output product to the input effort that produced it” (Institute of Electrical and Electronics Engineers, Inc., 1993, p. 12). Although this ratio may be as difficult to accurately quantify as problem-solving ability, it has been extensively studied in the context of Software Engineering.

Edward Nelson (1966) performed one of the earliest studies to identify programmer productivity factors. Nelson found that programmer productivity correlates with at least 15 factors. More recently, Capers Jones (2000) identified approximately 250 factors that he claims influence programmer performance. Summarizing this research, Endres and Rombach (2003) state that reducing productivity to “ten or 15 parameters is certainly a major simplification” (p. 190). With so many contributing factors...

---

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
Innovation, Imitation and Open Source
www.igi-global.com/chapter/innovation-imitation-open-source/52249?camid=4v1a

Open Source Software Evolution: A Systematic Literature Review (Part 1)
www.igi-global.com/article/open-source-software-evolution/179923?camid=4v1a