Understanding Language Selection in Multi-Language Software Projects on GitHub

Wen Li  
Washington State University, USA  
li.wen@wsu.edu

Na Meng  
Virginia Tech, USA  
nm8247@vt.edu

Li Li  
Monash University, Australia  
li.li@monash.edu

Haipeng Cai  
Washington State University, USA  
haipeng.cai@wsu.edu

Abstract—There are hundreds of programming languages available for software development today. As a result, modern software is increasingly developed in multiple languages. In this context, there is an urgent need for automated tools for multi-language software quality assurance. To that end, it is useful to first understand how languages are chosen by developers in multi-language software projects. One intuitive perspective towards the understanding would be to explore the potential functionality relevance of those choices. With a plethora of publicly hosted multi-language software projects available on GitHub, we were able to obtain thousands of popular, relevant repositories across 10 years from 2010 to 2019 to enable the exploration. We start by estimating the functionality domain of each project through topic modeling, followed by studying the statistical correlation between these domains and language selection over all the sample projects through association mining. We proceed with an evolutionary characterization of these projects to provide a longitudinal view of how the association has changed over the years. Our findings offer useful insights into the rationale behind developers’ choices of language combinations in multi-language software construction.

Index Terms—Multi-language software, language selection, functionality relevance, evolution

I. INTRODUCTION

Nowadays, technologies such as components and frameworks, which could provide high reliability and usability, are being utilized increasingly often in order to speed up the development and integration of software products. As software development continues to expand into a plethora of domains, including cloud services and IoT systems, different functional features or components are preferably developed with different languages to take advantage of combining the best of each. As a result, it has been clear that most of the current real-world software applications have used more than one language during construction [1]–[5].

The problem of locating defects across multi-component systems, with an effective program analysis technique, has created a new field of research that continues to grow. As a critical step prior to developing such techniques, it is useful to understand how languages are selected during multi-language software construction and the rationale behind it. For instance, an IoT software system might utilize Java (due to its merits in platform independence) for plug-in development and then use C (due to its efficiency advantages) for use cases involving system programming. These subsystems must then interact with each other as a whole for the entire system to function as required. The combination of C and Java seems to have become the most popular choices for IoT systems indeed. Yet a full picture in this regard remains to be seen.

Prior research has been concerned about programming language selection, but focused on languages used in a specific field (e.g., bioinformatics) or on the relationship between the use of individual languages and a specific property of software (e.g., bug-proneness). Other studies have looked at the potential correlation between bug resolution schemes and language use. These prior studies have not considered an evolutionary view or the rationale underneath language selection.

Towards filling these gaps, we are conducting a systematic study on language use and selection in open-source multi-language projects on GitHub. We collected 10,000 projects across ten years and characterized them as one single dataset for understanding the functionality relevance of language selection. Furthermore, treating the ten yearly sets of projects each as an individual dataset while applying the same association analysis led us to an evolutionary look at the relevance.

We found that language selection was weakly or moderately associated with some functionality domains. Over time, the top language selections for those domains changed considerably, whereas the primary languages appeared relatively stable. We are currently working on an in-depth analysis of various implications of language selection in multi-language software to discover the rationales behind such selections.

II. METHODOLOGY

Our study took open-source projects on GitHub as its primary input. From this data source, we mined different kinds of repository data for a single-period characterization (SPC) and an evolutionary characterization (EVC). We randomly picked projects with 1,000 or more stars (i.e., top-popularity projects). To be useful for our study, projects without a meaningful topic or description information were skipped, and those that use no more than one language were dismissed. We thus obtained 1,000 projects from each of the past ten years (2010 through 2019). Then, the SPC used the entire dataset as a whole, while the EVC treated each yearly sample set separately.

More specifically, in the SPC we extracted the topic metadata of each project, from which each project’s functionality domain (i.e., category) was identified through topic modeling using Latent Dirichlet Allocation (LDA) [6]. With the functionality category (domain) assigned to each project, we computed...
the association between such categories and language selections in a given project set via association rule mining. In particular, we identified frequent if-then associations which consist of an antecedent (if, software domain here) and a consequent (then, language selection here), using the Apriori algorithm [7] implemented in the Mlxtend library [8].

Then, in the EVC, where projects from each different year during 2010–2019 were considered one separate dataset, we looked across per-year (SPC) characterization results to analyze all the projects’ evolutionary dynamics of functionality relevance of language selection during the ten-year span.

III. RESULTS

Table I lists the results of our association analysis on the overall functionality relevance of language selection using the SPC dataset. Our results revealed that there was a noticeable relationship between language selection and certain functionality domains, although the association was relatively weak (e.g., between c c++ shell and OS, and between html java javascript shell) and at most moderate (e.g., between c++ java shell and games). Also, one language selection may not be consistently associated with one particular domain—for instance, c++ java shell was the top selection for both games and communication software. Also, one domain can be associated with more than one language selection—for instance, both c c++ cmake and c++ cmake shell for programming tools.

Figure 1 shows the association evolution for the four common domains across the years. The legend shows the set of languages most frequently included in the top language selections in the EVC dataset. These languages were mapped to fixed colors and cell positions for each domain and year to facilitate observing the evolution patterns. As shown, language selection constantly changed from year to year in any of the four domains. No selection was always associated with a domain, although some associations were relatively stabler than others. For instance, the association of objective-c ruby with mobile application stayed the same for three years: 2013, 2015, and 2016, while for programming tools the associated selection was never the same across the years. Meanwhile, there appeared to be some stable members in the top language selections associated with each domain.

IV. CONCLUSION

Every programming language has its strengths and weaknesses. Some are better for low-level and high-performance programming (e.g., C), while others might be more preferred for UI (user interface) (e.g., Java). Also, some are more relevant to specific software domains than others. While such questions have been explored at the basis of individual languages, the answers remain unclear for holistic language combinations in multi-language software construction. In this study, we aim to find semantic relationships between language use and software domains, and the top language combinations correlated with each popular software topic, if any. The answer provides one way to understand why developers would choose to use certain combinations of languages in their projects. We revealed that there exists a verifiable correlation between software domains and sets of mainstream languages, which were stronger for some language combinations and functionality domains than for others. The results offer an empirical reference for developers when choosing a desirable programming language combination for a partial domain, as well as potential insights for program analysis researchers on what language combinations to focus on and prioritize.

REFERENCES


