An Empirical Assessment of Polyglot-ism in GitHub

Federico Tomassetti
Dept. Control and Computer Engineering
Politecnico di Torino
Turin, Italy
federico.tomassetti@polito.it

Marco Torchiano
Dept. Control and Computer Engineering
Politecnico di Torino
Turin, Italy
marco.torchiano@polito.it

ABSTRACT

In this paper we study how the language cocktails are composed. How many languages are used in each software projects, which language types are used and which languages are typically used together. Our study was done on a sample of over 15,000 projects from the largest software forge, GitHub. The results show that many languages are used in each project: 96% projects employ at least 2 languages, over 50% employ at least two programming languages. Finally, there are strong relations between different languages: hence sets of languages tend to be adopted together.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

General Terms

Measurement

Keywords

Polyglot development

1. INTRODUCTION

Several different languages, typically selected from a large set of possible candidates, are typically used to develop software. Each language has its own characteristics which makes it particularly suitable to express certain facets of a system. In practice many different languages can be used together, in the same project: this is called polyglot-ism.

The adoption of each language has many consequences, on productivity and defectivity first of all. Each language determines also the libraries which can be used, the supporting tools that can be adopted, and the set of practitioners able to write and understand the code of the project. Moreover we know, from previous studies, that the effects of combining different interacting languages within a single software project have yet to be understood [4].

EASE '14, May 13 - 14 2014, London, England, BC, United Kingdom

http://dx.doi.org/10.1145/2601248.2601269

We believe that before studying in detail how languages interact within a single project, we need to assess the relevance of the phenomenon and to characterise how the mix of languages – also called language *cocktails* – are used in the software projects. We think the composition of the cocktails of the languages selected to develop a particular software project is a fundamental aspect to understand the nature of that project.

This paper reports an empirical work focused on the open-source projects stored in the GitHub forge. We have chosen GitHub because it is by far the code forge hosting more projects in the world and it is gaining more and more traction. In this work we aim to consider all possible languages involved in software development: not only programming languages but also languages used to produce documentation, data, configuration, building scripts and so on. We believe all these languages play an important role in software development.

The contribution of this paper is twofold: (i) a very-large scale assessment of the relevance of polyglot-ism in open-source software development projects, (ii) an initial characterization of the language cocktails used in such projects.

2. RESEARCH DESIGN

The goal of this research effort can be define, using the GQM template as: *Analyze* software projects *for the purpose of* characterization and relevance assessment *with respect to their* polyglot-ism *from the point of view of* software developers *in the context of* the GitHub forge. In the present paper we address two main research questions:

RQ1) What is the level of polyglot-ism of open-source projects? Through this question we can find the characteristic *size of language cocktails*. The number of languages used in any single project gives us an indication on the amount of different skills required to understand the project (higher if many languages are used) and the extension of the polyglot-ism (present when more than one language is used).

RQ2) Which are the typical pairs of interacting languages? We want to understand which languages are typically employed together. If the adoption of one specific language strongly influences the adoption of other languages, language cocktails should be studied as clusters, connected by different inter-relations. Through this question we can find the *inter-relations inside language cocktails*. Understanding which are the most common interactions could be

useful to direct future research on cross-language interactions.

3. METHOD

To answer our research questions we extracted from GitHub a random sample of relevant software projects and we proceeded to analyze automatically all of them. First we extracted an initial set of projects, performing and initial screening (Set `A`). From the resulting set we filtered out trivial projects (Set `B`). For each project part of Set `B` we examined the most recent version and extracted a list of files. We included in Set `C` exclusively the projects for which we could classify an high fraction of files.

We extracted random IDs from 1 to the ID of the last created project. We excluded non-existing, private and fork projects (Set A). We discarded the projects having either less than 10 files or counting less than 10 commits in the repository (Set B). For each remaining project we considered the list of files present in the `master` branch at the moment of the extraction. We tried to recognize the file type of each file looking at the file name. We started from the file type definitions provided by GitHub Linguist[1] project. Through some iterations we enriched the original set of type definitions, adding a few more file types with the corresponding naming schemes, obtaining 197 distinct languages schemes. We accepted a small portion of unclassified files per project but we decided to excluded all the projects where such proportion is more than $^1/_3$. While this proportion could seem high, it actually corresponds to a very limited number of files for most projects, considering the low median of the number of files per project. In our analysis we did not considered all possible file types. We were interested exclusively in utterances of languages used for software development. Therefore we excluded all the files relative to resources (e.g., images or videos), to packaged files (e.g., zip and tar.gz or tar.bz2), the files connected to version control systems (e.g., `.gitignore` files) and so on.

We classified languages according to their function. The functions we identified are:

Programming languages we included in this category all the languages typically used to describe business logic in an computable way. The different paradigms used to describe the program (e.g., imperative, functional, object-oriented) are not considered.

Documentation languages we indicate all the languages which are used to describe architecture and design decision, as well as code or the other artifacts used in project development.

Presentation languages this category includes all the languages mainly focused on describe the appearance of applications.

Data languages by data languages we consider all the languages usable to describe datasets either small or large. A sub-category of these languages include configuration languages (e.g., INI or properties files).

Automation languages includes all kind of scripts used to automate tasks. Example include but are not limited to shell scripts and build automation scripts.

[1]See `https://github.com/github/linguist`

Analysis. To answer **RQ 1:** *What is the level of polyglot-ism of open-source projects?* we assume that polyglot-ism level correspond to the number of languages used. We report the distribution of the number of distinct languages (N_{lang}) that appear in a project. In addition we report also the number of languages per type per project; in particular focusing on the number of programming languages (N_{lang_prog}), which represent the cornerstone of any software development project.

Concerning **RQ 2:** *Which are the typical pairs of interacting languages?* we adopt a common approach in data mining: association rules. In particular we compute the *support* and *confidence* of binary language associations ($L_A \implies L_B$). In general, the *support* of a set of languages X is the proportion of projects where both languages appear together. The support of an association is the support of the set composed of the left-hand side and the right-hand side of the association:

$$supp(L_A \implies L_B) = supp(\{L_A, L_B\})$$

The *confidence* of an ordered pair of languages (L_A, L_B) is the conditional probability of L_B given L_A:

$$conf(L_A \implies L_B) = P(L_A|L_B) = \frac{supp(\{L_A, L_B\})}{supp(\{L_A\})}$$

In particular we will select the associations whose support is greater than 3% and confidence is greater than 50%.

4. FINDINGS

We extracted 72,000 projects (Set A) and after our filtering process we considered 15,216 projects (Set C). The distributions of the number of commits and files of the selected projects are positively skewed (skewness 33.6 for commits and 16.3 for files), they present relatively low medians (30 commits and 55 files) and larger means (206 commits and 415 files).

RQ1: What is the level of polyglot-ism of open-source projects? A few projects (34, i.e. 0.2%) are reported to have zero languages: they contain only files which cannot be considered belonging to a development languages (e.g., images or zip files); 3% of projects employ one single language, while the remaining (roughly 97% of projects) employ 2 or more languages. The most common configuration is a cocktail containing four distinct languages in a single project. Projects using between 2 and 12 languages represent 90% of the projects in GitHUB, while 3.8% of projects employ 15 or more languages. The median number of is 5 languages, while the mean is 6.01. The maximum number of languages used in a single project is 52.

Concerning the language types that make-up such language cocktails, we focus first on the main type: programming languages. More than 50% of projects employ two or more programming language, while the single most common configuration is one programming language per project. The median is 2, while the mean is 1.99 . The maximum number of programming languages employed in one single project is 36.

The distribution of the other languages types per project, including a few summary statistics are reported, in Table 1. The median shows that a typical configuration include

Table 1: Adoption of language per type per project

Language	No. projects				Statistics		
	0	1	2-4	5+	Med	μ	Max
Programming	2.5%	46%	47%	4.5%	2	2.0	36
Presentation	42%	13%	43%	2.7%	1	1.4	10
Automation	57%	32%	11%	0.5%	0	0.6	11
Document.	29%	52%	18%	0.4%	1	1.0	7
Data	32%	27%	31%	0.2%	1	1.1	6
Total	0.2%	3.3%	41%	55.5%	5	6.0	52

Table 2: Top most used pairs

Rank	Languages		No. projects	Support
1	CSS	JavaScript	6083	40.0%
2	CSS	HTML	5825	38.3%
3	HTML	JavaScript	5769	37.9%
4	HTML	TXT	5383	35.4%
5	CSS	TXT	5183	34.1%
6	JavaScript	TXT	5168	34.0%
7	TXT	XML	4654	30.6%
8	HTML	XML	3330	21.9%
9	CSS	XML	3326	21.9%
10	TXT	YAML	3108	20.4%

2 programming languages, 1 presentation language, 1 documentation language and 1 data language. Commonly no automation languages are instead used.

RQ2: Which are the typical pairs of interacting languages? The top most used pairs are reported in Table 2. Figure 1 reports a graph with the associations between the languages. While representing associations we excluded TXT because it was involved in too many relations, cluttering the resulting figure. The nodes represent the languages, in particular the rectangle is used for programming languages, the triangle for automation languages, the ellipse for documentation languages, the diamond for presentation languages and the parallelogram for data languages. Edges between nodes represent associations having support $> 3\%$ and confidence $> 50\%$.

We can observe a few clusters of languages. In the upper right corner of the figure we find the C/C++ cluster that includes the two programming languages and one automation language (Makefile). In the bottom part of the graph we can observe the triangle HTML-CSS-Javascript that are strongly associated with each other. This group ties together the other groups that refer to different programming languages. On the upper left side of the picture we can observe the language cocktails belonging to the articulated ecosystem around Ruby and Rails, its flagships framework for web applications development. On the right side of the graph we find a set of programming languages all linked to XML: PHP, Objective-C, and Java. Finally directly linked to HTML we have Python.

5. THREATS TO VALIDITY

We analyze the potential threats to validity according to four categories.

Construct. In our work we assume that the polyglot-ism level corresponds to the number of distinct languages used in a project. There are other aspects, such as the quantity of files for each language as well as the interactions among languages [4]. Clearly having different independent components developed using distinct languages is very different from combining several tightly integrated languages into a component.

Internal. Having used an automatic, unsupervised approach, there is the possibility of miss-classification. To reduce this threat we re-used an already existing classifier (GitHub linguist), introducing only small and careful improvements. Another threat is due to the fact that we were not able to classify all the files in all the projects. This means that more languages could be present in respect to the data we report. To reduce this risk we excluded the projects with too many files not attributed to any language. Since we used files as our smallest unit of analysis, in our work we did not consider the case of files containing more than one language.The actual threat is to underestimate the number of languages adopted in a software development project and to miss some associations between languages.

External. GitHub can be not representative of the whole open-source movement, other communitees can prefer other platforms. For example communities operating on .NET could tend to use CodePlex.We do not claim our results are more generalizable outside the GitHub community (with the exclusion of trivial projects), therefore we should not be affected by external threats to validity.

Conclusion. The aim of this work is mainly descriptive. We do not draw conclusions based on statistical tests. Concerning the language association strength, we used commonly used indexes, that is support and confidence. The thresholds used – 3% for support and 50% for confidence – are large enough to report only relevant associations. In particular we would like to stress that 3% support means that the pair of languages was found in at least 456 projects.

6. RELATED WORK

Research on Polyglot software development can be organized in three main directions: *i)* to offer a classification of cross-language relations, *ii)* to characterize empirically the effects of cross-language relations.

Classification: in [3] the authors presented a classification of different forms of cross-language relations. The types we identified are six: *shared ID, shared data, data loading, generation, description,* and *execution.* This is the only work we know which consider all possible forms of relations, while other works consider exclusively cross-language relations based on the *shared ID* mechanism. Meyer and Schroeder [1] classify exclusively cross-language links implemented in Java frameworks in respect to XML artifacts. The relations they consider correspond to the category *shared ID*, according to the classification proposed in [3]. They built metamodels of the languages involved (Java and XML) for this particular purpose and specified the rules controlling the cross-language relations of three Java framework, deriving from them common patterns.

Empirical results: in [4] the authors investigated how many of the commits of the Hadoop project involved more than one language and the effect of being involved in cross-language (i.e. polyglot) commits on defectivity. Results show that some relations are particularly negative.Pfeiffer *et al.* [2] used TexMo in a controlled experiments with 22 subjects to demonstrate the effects of tool support for cross-language references. Their results show great benefits due to the adoption of specific tool support for cross-language

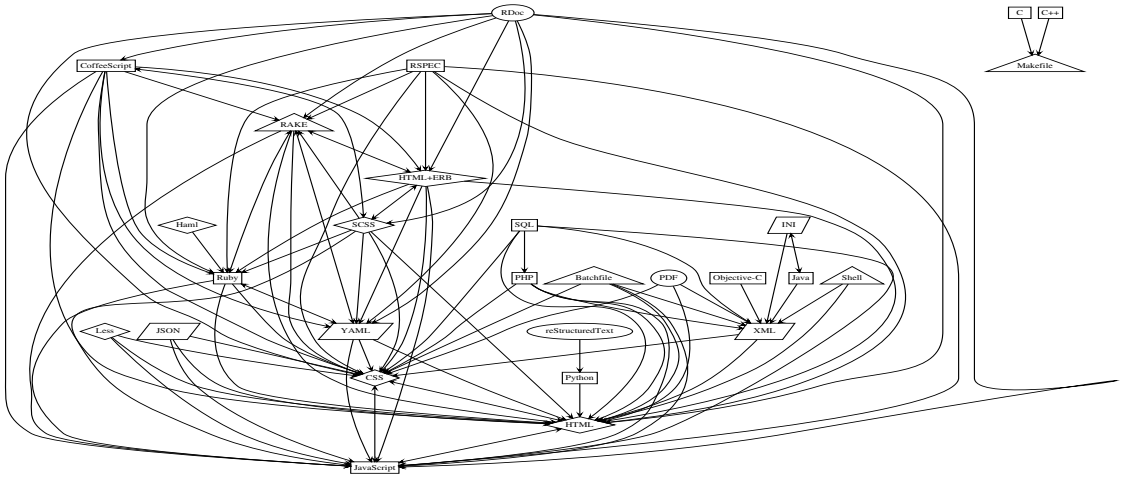

Figure 1: Clusters of languages.

relations.

7. DISCUSSION

Our results show that in practice a **vast majority of software development projects employ many languages**. Half of them use one single programming language, but they are typically supported by data, automation, presentation and documentation languages, while the other half use also different programming languages. Considering a loose definition of Polyglot-ism – counting all languages types – we can conclude that it is a highly relevant phenomenon. which is present in practically all projects. Even sticking to a more strict definition of Polyglot-ism – considering only programming languages – the phenomenon is still very relevant and encompasses the majority of projects.

A second point is that languages are not picked up for adoption singularly but there are whole ecosystems which are chosen. When a practitioner choose to program in Java he will tend to adopt XML, while if Ruby is chosen, probably he will opt for YAML as a data language. If Python is chosen, the automation language of reference will be DISTUTILS, if instead C++ is picked up, CMake or Makefile will be probably also chosen.

In this paper we started to investigate the relations between languages. We had to choose thresholds to limit the number of relations selected, so that we could present and discuss them conveniently in this work. For sure there is a number of relations which are strong (high confidence) but are related to languages not so common on GitHub (and therefore eliminated because of the threshold on support). We believe that more of these relations are worthy to be investigated in future work.

The choice of the languages is complex because it implies which other languages can be conveniently included, to obtain a set of languages, to be used for different tasks, which can operate well together, leading to a manageable cost of integration.

Results from our work are obtained exclusively from GitHub. We do not claim them to be generalizable to software development in general. Still, GitHub is the largest software forge in the world; we therefore think it is interesting to study GitHub per-se.

8. CONCLUSIONS

Software development is typically operated using many languages together. The set of languages complexly adopted in a software project is a web of elements with strong connections. They have to be integrated together to realize a fully working system. We believe that the techniques to compose them, and the cost of integration are crucial elements in software development, and this topic would deserve to get more attention.

This work could be complemented by investigating other software forges, to study which of our findings can be generalized to the whole open-source software development and which ones are specific of the GitHub community. This work could be also complemented with other means of investigation. For example with surveys among practitioners or with the examination of closed source software.

9. REFERENCES

[1] P. Mayer and A. Schroeder. Patterns of cross-language linking in java frameworks. In *21st Int. Conf. on Program Comprehension (ICPC'13)*, 2013.

[2] R.-H. Pfeiffer and A. Wasowski. Cross-Language Support Mechanisms Significantly Aid Software Development. In *Model Driven Engineering Languages and Systems*, volume 7590 of *LNCS*, pages 168–184. Springer Berlin Heidelberg, 2012.

[3] F. Tomassetti, M. Torchiano, and A. Vetro'. Classification of Language Interactions. In *7th Int. Symp. on Empirical Soft. Eng. and Measurement (ESEM'13)*, 2013.

[4] A. Vetro', F. Tomassetti, M. Torchiano, and M. Morisio. Language interaction and quality issues: an exploratory study. In *Proc. of the ACM-IEEE int. symp. on Empirical software eng. and measurement*, ESEM '12, pages 319–322, New York, USA, 2012. ACM.