Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects

Tegawendé F. Bissyandé[1], Ferdian Thung[2], David Lo[2], Lingxiao Jiang[2] and Laurent Réveillère[1]
[1]*Laboratoire Bordelais de Recherche en Informatique, France*
[2]*Singapore Management University, Singapore*
{*bissyande,reveillere*}*@labri.fr,*{*ferdianthung,davidlo,lxjiang*}*@smu.edu.sg*

***Abstract*—Programming languages have been proposed even before the era of the modern computer. As years have gone, computer resources have increased and application domains have expanded, leading to the proliferation of hundreds of programming languages, each attempting to improve over others or to address new programming paradigms. These languages range from procedural languages like C, object-oriented languages like Java, and functional languages such as ML and Haskell. Unfortunately, there is a lack of large scale and comprehensive studies that examine the "popularity", "interoperability", and "impact" of various programming languages. To fill this gap, this study investigates a hundred thousands of open source software projects from GitHub to answer various research questions on the "popularity", "interoperability" and "impact" of various languages measured in different ways (e.g., in terms of lines of code, development teams, issues, etc.).**

***Keywords*-Programming languages; Popularity; Interoperability; Open source; Software projects; GitHub**

I. Introduction

Computers have a unique purpose: to perform instructions given to them by us humans. A natural question arises when a computer tries to understand and perform any given instruction: in what format the instructions should be so that it may be easy for humans to describe and easy for computers to understand. Assembly languages, mostly low-level and specific to certain computer architectures, have early been proposed to ease the programming of computer instructions by using mnemonics. To improve developer productivity and facilitate large scale software development and maintenance, many higher-level languages with a stronger abstraction from the details of a computer have also emerged. 70 years after the apparition of the modern computer, hundreds, if not thousands, of programming languages have existed—the Wikipedia Internet encyclopedia lists about 600 languages[1].

Various programming languages have their advantages and disadvantages. Assembly languages are often accredited to be easier to work with hardware and execute more efficiently, while higher-level languages often provide a rich set of grammatical rules and vocabulary that makes programming much easier for developers. Some programming languages have been introduced fairly recently, while others have been used for decades. For example, C# was only introduced in 2001, while C has been in use since 1972. There are also various ways to classify languages into various categories, including procedural languages (e.g., Fortran, Pascal, and ANSI C), aspect-oriented languages (e.g., AspectJ), object-oriented languages (e.g., Java and C++), and functional languages (e.g., Haskell). Thus, there are a wide variety of options that developers can choose when writing applications.

Due to the wide variety of programming languages out there, there is a need to investigate the popularity, interoperability, and impact of different programming languages. "Popularity" of programming languages is a challenging property to assess. In this study we investigate different correlations of software development metrics with the use of common programming languages. We use the term "interoperability" to refer to the extent to which two programming languages are used together in software projects. An understanding of these factors could help, for example, developers select languages to use or learn, and could help managers assess developer pool sizes in function of their skills in programming languages.

Despite the above-mentioned benefits, there have been, however, limited studies that investigate the popularity, interoperability, and impact of different programming languages. Thus there is a need for a comprehensive analysis of these factors based on a large number of projects written in various programming languages. In this work, we fill this need by investigating a hundred thousands projects and analyze these factors by answering a set of research questions. Since "popularity", "interoperability" and "impact" are concepts that involve many variations, we explore these properties of programming languages across several dimensions. In particular, our study aims at answering the following questions:

RQ1: How popular are the various programming languages in terms of adoption in real-world software projects?
RQ2: How many projects are written in more than one programming language and what is the degree of interoperability of each language towards the others?
RQ3: Is there a correlation between the programming language used and the project success?

[1]http://en.wikipedia.org/wiki/List_of_programming_languages

RQ4: What is the correlation between the programming language used and the number of issue reports?

RQ5: How does the programming language correlate with the size of the development team?

A previous study related to ours is the TIOBE programming community index which ranks various programming languages [15]. The ranking is based on "the number of skilled engineers world-wide, courses, and third-party vendors".[2] The rank is computed by performing searches on a number of search engines including Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube, and Baidu. The data used by TIOBE is not freely available and we could not find a description of the exact methodology for the ranking on TIOBE's site. Different from TIOBE's ranking, we investigate the popularity, interoperability, and impact of programming languages from a different angle: we do not look into search engines or analyze skilled engineers, courses, or third party vendors; rather, we analyze a hundred thousands of open source software projects. We believe that analyzing actively developed software artifacts is a promising way to understand how programming languages *are* being used. We describe our approach in detail in this paper and make our data publicly available[3]. Conway has also provided a ranking of languages based only on their appearance in GitHub projects [2]. In this study we further investigate the amount of lines of code written for each language.

Our study is made possible by the availability of a large amount of data in GitHub. Millions of projects are publicly available for download in GitHub. There are small and large projects; large ones include the Linux kernel (with over 10,000,000 lines of code written by more than 9,000 contributors). The availability of a wealth of project data allows us to explore a variety of interesting questions on the popularity and impact of various programming languages. We are however aware that the findings based on these open source projects are only meant to shed light, to some extent, on the usage of common programming languages. Some languages might be under-represented in the dataset of projects because of the constraints in their development context. For example, the VHDL language, which is seldom in our dataset, is widespread for embedded systems in the industry, a domain that tend not to publish their work to open source platforms. Aside from such cases, our study provides a clear picture on the popularity of programming languages in public domain.

For this study, we have obtained 100,000 git repositories from GitHub. We have analyzed the contents of each of the git repositories to find software code. We count the numbers of lines of code written in various programming languages. Based on this information, we can answer our formulated questions related to the popularity and interoperability of programming languages. We also collect various information related to project success, reported issues, and team sizes. The contributions of this work are as follows:

1) As far as we know, we are the first to analyze the popularity of various programming languages by analyzing a hundred thousands of open source software projects. We report the popularity of various programming languages in various dimensions: the numbers of projects written in a language, the numbers of lines of code written in a language, and the numbers of developers that "read"/"write" a language.
2) We analyze the interoperability of various programming languages based on projects that are developed using more than one programming language. We measure how close a pair of programming languages are based on their interoperability.
3) We describe the correlation between programming languages and project "success" in the developer community, reported bugs, and development team sizes.

The rest of this paper is organized as follows. In Section II, we provide preliminary information on various programming languages and GitHub. In Section III, we elaborate the methodology of our empirical study. In Section IV, we present the results of our study. We list the threats to validity in Section V. Section VI discusses related work. We conclude and describe future work in Section VII.

II. Preliminaries

In this section, we introduce various programming languages, and briefly describe GitHub and the kind of projects it hosts.

A. Programming Languages

Programming languages define the grammar and semantics used by human beings to communicate and interact with machines. Programming languages, such as the one used by Ada Lovelace in her first program [5], have existed before the invention of modern computers in the 1940s. With the advent of the computer, programmers started to use assembly languages. Seventy years later, hundreds of programming languages have been invented. While some have been discarded with the arrival of their better alternatives, others have kept their usage trends. A few more recent programming languages have quickly gained momentum in the last years.

Each programming language belongs to a category that is more or less suitable for different programming tasks and computing environments. We only focus on 30 commonly known programming languages that appear in our dataset. Table I describes each language, its category, and its year of apparition. The oldest one, Fortran, was proposed 56 years ago, while the most recent, C# appeared about 10 years ago. These languages are classified into categories

[2]http://www.tiobe.com/index.php/content/paperinfo/tpci as of June 2012

[3]Upon request.

Table I
30 COMMON PROGRAMMING LANGUAGES

Language	Main classifications	Appearance
Fortran	Imperative / Procedural	1957
Lisp	Functional / Interpreted	1958
COBOL	Imperative / Compiled	1959
Yacc	Syntax handling	1970
Pascal	Imperative / Procedural	1970
ANSI C	Imperative / Procedural	1972
ML	Functional / Compiled	1973
Sed	Command Line/Parsing	1973
Lex	Lexical analysis	1975
Shell (sh)	Command Line / Interpreted	1977
Awk	Rule based / Scripting	1977
C Shell (csh)	Command Line / Interpreted	1978
Ada	Concurrent	1980
VHDL	Dataflow	1980s
Modula-3	Imperative / Object-Oriented	1980s
C++	Imperative / Object-Oriented	1983
Perl	Imperative / Object-Oriented	1983
Objective-C	Object-oriented	1983
Erlang	Functional / Compiled	1986
Tcl	Scripting	1988
Expect (exp)	Scripting / Automation	1990
Haskell	Functional	1990
Python	Interpreted / Object-Oriented	1991
Fortran 90 (f90)	Imperative	1991
JavaScript	Object-Oriented	1995
Java	Object-Oriented	1995
PHP	Imperative	1995
Ruby	Imperative	1995
JSP	Imperative	1999
C#	Imperative/Functional/Object-Oriented	2001

with functional and object-oriented features, compiled and interpretive designs, scripting and execution purposes, etc.

Given the capabilities of general purpose programming languages, a programming task can often be implemented in any of them. Then, why a language would be preferred against another? It is important to understand such a question as it may provide insights for designing better languages. Before understanding "why", it is important to know what the "popular" programming languages are and what the impact of a language is on the evolution of a project, which are the topics of this paper.

B. GitHub

GitHub is a project hosting site that has introduced the concept of *social coding* by providing many developer-friendly features. It can be viewed as a socio-technical network. GitHub has thus gained a wide acceptance and adoption among developers, and its website indicates that it is currently[4] hosting more than 3,000,000 project repositories. Thanks to GitHub's extensive REST [4] APIs[5], we are able to identify about 1,300,000 million repositories whose contents are publicly available for use.

GitHub uses *git* [7] as its revision control system for source code versioning. It enables *forking*, which allows to create copies of repositories, and it supports *pull requests* for merging changes from copies to the source repository, to facilitate contributions from developers who are not project team members. Aside from source code revision control, GitHub integrates common software development facilities, such as bug/issue trackers and wiki pages.

[4]http://github.com as of June 2012

[5]http://developer.github.com

A large number of projects, such as those from the Apache community, which are developed outside the GitHub platform, i.e., autonomously or on other platforms such as Sourceforge, have their repositories mirrored in GitHub. This fact indicates that by using GitHub projects, we do not limit ourselves to the GitHub community or to git. Furthermore, a number of famous companies such as Facebook™ host open source versions of their projects on GitHub. This suggests that our project dataset contains projects managed by diverse communities. While the large number of projects hosted by GitHub and its extensive APIs for accessing the projects enable large studies like ours, there are still some challenges that we need to address, such as (1) collecting and managing huge amount of data, and (2) inferring important information which are not directly available from the APIs, such as the number of source lines of code in a project.

III. METHODOLOGY

To perform our empirical study, we need a large number of projects that may represent the universe of all projects. The software projects should be written for various purposes by diverse teams in a variety of languages. We have done a manual exploratory survey of the projects on GitHub, investigating the development teams, the application domains and the range of programming languages used in the projects. We have thus found that projects hosted by GitHub are very diverse and thus suitable to the requirements of our study. We have also found that GitHub is used as the main development platform for over a million projects, and that it may also be used as a mirror for popular projects, such as Apache or Linux.

For our study, we consider the first 100,000 projects returned by the GitHub APIs. There appears to be no distinct ordering scheme in the returned list of projects[6] which also vary on subsequent requests. Given these projects, we extract several types of information for each project:

General project information: Each repository is monitored by GitHub and tracked based on different features. In this paper, we consider the concepts of *watchers* and *forks*. "Watchers" is a metric for measuring interest and activity in a project. It gives an indication of the amount of attention that is given to a project by the developer community: an aspect of project "success". Watchers typically use project releases, report bugs, and incidentally promote the project in their socio-technical network. "Forks" is a metric for measuring the active involvement of the developer community in the growth of a project's code base and the improvement of its quality. While these metrics are not absolute, they provide good insights on the "success" of a project.

Source Lines of code: To compute the number of physical source lines of code (sloc), we download each project's code repository and rely on the SLOCCount[7] tool.

[6]The list is available at http://momentum.labri.fr/orion/project_list.txt

[7]http://www.dwheeler.com/sloccount/

This tool computes actual lines of code, ignoring code comments and blank lines. It supports 29 of the languages considered in our study. We have extended SLOCCount to add support for the 30th language, namely JavaScript.

Developer contributions: We also consider the contributors for a given project inferred from the commits. Whether a contributor to a project is registered or not in GitHub, his information is always available in the git repositories that he contributes to. This inference is particularly important for projects that are just mirrored in GitHub but whose development is done on the project's own website.

Issue Reports: Finally, we consider issue reports as an important artefact of software development process. For each project repository, we have crawled the corresponding issues database on GitHub. Though an "issue" may refer to a bug report or a feature request, we study them together as both are good indications of the interest that the programmer community gives to a project.

Based on the aforementioned information for each project, we extensively explore the popularity and impact of programming languages in various dimensions.

IV. Empirical Evaluation

In this section, we investigate the research questions described in Section I to assess the popularity of programming languages. We investigate beforehand the dataset to ensure that most of the projects are non toy projects of substantial sizes. To this end we count the total lines of source code (LOC) in each project. Fig. 1 plots the percentages of projects with different numbers of lines of source code. Over 70% of the projects contain more than 1,000 LOC. Around 35% of the projects include more than 5,000 LOC, while more than 20% contain more than 10,000 LOC. Finally, over 600 projects contain more than 1,000,000 LOC. This distribution suggests that a significant number of the projects in the dataset are of substantial sizes.

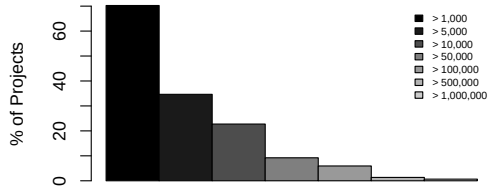

Figure 1. Distribution of the Dataset Projects in Terms of Total LOC

A. RQ1: Popular Programming Languages

To provide insights into the first research question, we investigate the popularity of the 30 programming languages in terms of the number of lines of code in the 100,000 software projects that are written in each particular language, the number of projects that contain code in each particular language, and the number of developers that are involved in a project that contain code in each particular language. We describe the results of our study in the following paragraphs.

Table II
Programming Language Popularity—Lines of Code

Rank	Language	# LOC	% LOC
1	ansi c	1,615,634,331	60.83 %
2	javascript	296,893,761	11.18 %
3	c++	217,566,364	8.19 %
4	php	167,458,938	6.31 %
5	java	99,308,060	3.74 %
6	ruby	59,967,003	2.26 %
7	python	53,850,088	2.03 %
8	c#	31,560,343	1.19 %
9	lisp	27,614,150	1.04 %
10	sh	22,605,731	0.85 %
11	objective c	16,570,836	0.62 %
12	perl	16,413,762	0.62 %
13	pascal	11,766,801	0.44 %
14	erlang	7,335,480	0.28 %
15	yacc	1,646,328	0.06 %
16	ml	1,550,750	0.06 %
17	fortran	1,468,246	0.06 %
18	tcl	1,351,073	0.05 %
19	haskell	1,117,902	0.04 %
20	jsp	972,759	0.04 %
21	ada	878,412	0.03 %
22	f90	588,801	0.02 %
23	lex	573,349	0.02 %
24	vhdl	485,806	0.02 %
25	expect	318,554	0.01 %
26	awk	199,513	0.01 %
27	cobol	135,257	0.01 %
28	csh	74,608	0.00 %
29	sed	26,549	0.00 %
30	modula3	985	0.00 %

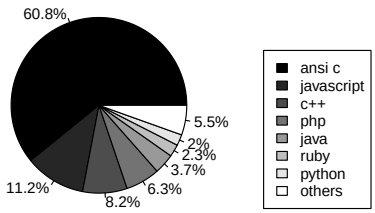

Figure 2. Language Popularity — Lines of Code

Table II shows the ranking of various programming languages in terms of total lines of code written in each language. C code dominates with 1.6 billion lines of code (60.83% of all LOC that we analyze). Next to C code is JavaScript and C++ code, which have 296 (11.18%) and 217 (8.19%) millions lines of code respectively. PHP and Java follow with 6.31% and 3.74% of all LOC respectively. Other well-known languages, such as Ruby, Python, and C#, only account for 2.26%, 2.03%, and 1.19% of all LOC. Functional languages, such as ML and Haskell, appear in more than 1.5 (0.06%) millions and 1 million (0.04%) LOC. The least popular programming languages in terms of LOC are C-Shell, Sed, and Modula3.

Figure 2 represents the distribution of LOC for the top-ranked languages. The predominance of C can be related to the fact that many operating system kernels, including the Linux kernel (10,064,207 lines of C code as of June 2012) and its device-specific flavors, are among the projects with the largest code bases and are written in C.

Language popularity can also be measured in terms of its adoption by the number of software projects using the language. Thus, for each programming language, we count the number of projects that contain code written in this

Table III
Programming Language Popularity—Appearance in Projects

Rank	Language	# Projects	% Projects
1	javascript	27,873	27.87 %
2	ruby	19,857	19.86 %
3	python	15,224	15.22 %
4	sh	14,444	14.44 %
5	php	11,023	11.02 %
6	java	10,646	10.65 %
7	ansi c	10,142	10.14 %
8	c++	6,865	6.87 %
9	perl	5,741	5.74 %
10	objective c	3,721	3.72 %
11	c#	3,082	3.08 %
12	lisp	2,005	2.01 %
13	pascal	1,165	1.17 %
14	haskell	1,071	1.07 %
15	jsp	958	0.96 %
16	yacc	887	0.89 %
17	erlang	652	0.65 %
18	awk	601	0.60 %
19	lex	567	0.57 %
20	sed	409	0.41 %
21	tcl	255	0.26 %
22	csh	242	0.24 %
23	ml	208	0.21 %
24	ada	195	0.20 %
25	fortran	134	0.13 %
26	expect	102	0.10 %
27	f90	77	0.08 %
28	vhdl	45	0.05 %
29	modula3	13	0.01 %
30	cobol	10	0.01 %

language. Table III details the ranking of the programming languages in this scenario. This metric (i.e., popularity by the number of projects) is similar to GitHub's own metric. Our findings are globally similar to the ranking provided by GitHub and used by Conway in his study.[8] JavaScript largely outranks other programming languages with 27,873 projects, which represents 27.87% of all projects in our dataset. The next ones in the ranking are also other scripting languages that are popularly used in 19.86% (Ruby), 15.22% (Python), and 14.44% (Shell) of the projects, respectively. Then follow general-purpose programming languages, such as Java, C, C++, and C#, which are used in 10.65%, 10.14%, 6.87%, and 3.08% of the projects respectively.

We also note that the Haskell functional language has gained popularity with 1,071 projects (1.07%) containing Haskell code. Based on this metric, Haskell outranks ML, its counterpart functional language. Finally, the least popular programming languages are VHDL, Modula3, and Cobol, which, even when combined, are found in less than 1% of all projects.

Table IV details the distribution of the top-10 main languages in the projects. In this study, we consider a language to be the *main language* for a project when it is the language with the most lines of code. The top ranks are still held by scripting languages. However, when we compare the numbers of projects in Table IV with those in Table III, we can see that scripting languages are often used as supporting languages. For example, although JavaScript appears in almost 28 thousands projects, it is the main language in only about 9 thousands (34.50% of all projects with some JavaScript code); The Shell scripting language appears in over 14 thousands projects, but it is the main language in only 3 thousands (22%) projects. Compiled programming languages, such as Java, C, and C++ are used more often as main languages ($> 50\%$) in projects.

[8]http://redmonk.com/sogrady/2012/09/12/language-rankings-9-12/

Table IV
Language Popularity—Top-10 Main Languages in Projects

Rank	Language	# Projects
1	ruby	12,642
2	python	10,165
3	javascript	9,616
4	java	8,861
5	php	7,886
6	ansic	6,307
7	c++	4,477
8	sh	3,311
9	objective c	2,982
10	perl	2,472

Another metric that we use to investigate the popularity of a language is the number of developers using it in their projects. For computing the ranking, we assume that all developers in a project are knowledgeable in the main language of the project. Figure 3 illustrates the distribution of developers for the top ranked languages. Overall, C appears to be the most popular language among developers, followed by Ruby, Python, Java, and C++. JavaScript and C# are only used by 8.97% and 3.42% of the developers. The detailed results for all languages are shown in Table V. Languages such as C-Shell, Modula3, and Cobol appear to be used by relatively insignificant numbers of developers.

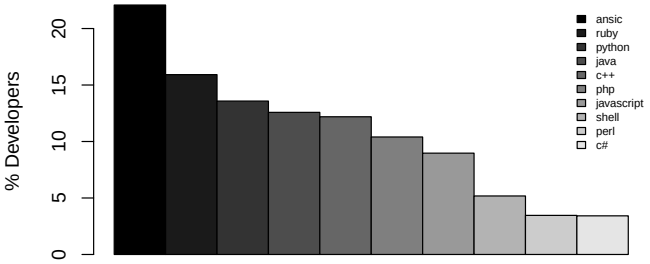

Figure 3. Language Popularity—Adoption by Developers. Percentages of developers that adopt different programming languages.

Ansi C, JavaScript and C++ top our ranking in terms of LOC. Most of the projects in our dataset contain code in JavaScript, Ruby in Python. Finally, Ansi C, Ruby and Python are used in projects with the largest developer pools. These findings suggest that these languages should be learned.

B. RQ2: Multi-Language Projects

Developers often rely on different programming languages to implement different functionalities in a single project. The second research question investigates the interoperability of languages to identify the languages that are popularly used in multi-language projects. We assume that two languages are "interoperable" if they are used in a same project.

Table V
Programming Language Popularity—Developer Pool. Numbers and percentages of developers that adopt different programming languages.

Rank	Language	# Developers	% Developers
1	ansi c	48,373	22.08%
2	ruby	34,878	15.92%
3	python	29,764	13.58%
4	java	27,567	12.58%
5	c++	26,708	12.19%
6	php	22,790	10.40%
7	javascript	19,651	8.97%
8	sh	11,345	5.18%
9	perl	7,587	3.46%
10	c#	7,499	3.42%
11	objective c	6,405	2.92%
12	lisp	3,644	1.66%
13	haskell	2,026	0.92%
14	erlang	1,757	0.80%
15	pascal	1,161	0.53%
16	ml	408	0.19%
17	tcl	213	0.10%
18	jsp	151	0.07%
19	fortran	75	0.03%
20	yacc	67	0.03%
21	ada	48	0.02%
22	vhdl	48	0.02%
23	f90	28	0.01%
24	sed	8	0.00%
25	expect	7	0.00%
26	lex	5	0.00%
27	awk	4	0.00%
28	csh	3	0.00%
29	modula3	3	0.00%
30	cobol	1	0.00%

Table VI
Language Interoperability. Numbers and percentages of projects written in multiple programming languages, that include different programming languages.

Rank	Language	# Multi Language Projects	% Multi Language Projects
1	javascript	18,353	21.81%
2	sh	12,456	14.80%
3	ruby	9,782	11.62%
4	python	7,757	9.22%
5	ansi c	7,646	9.09%
6	php	6,160	7.32%
7	c++	4,577	5.44%
8	java	4,163	4.95%
9	perl	3,841	4.56%
10	objective c	1,593	1.89%
11	c#	1,391	1.65%
12	lisp	1,239	1.47%
13	pascal	958	1.14%
14	jsp	925	1.10%
15	yacc	884	1.05%
16	awk	598	0.71%
17	lex	567	0.67%
18	haskell	464	0.55%
19	sed	405	0.48%
20	erlang	394	0.47%
21	tcl	243	0.29%
22	csh	241	0.29%
23	ada	187	0.22%
24	ml	153	0.18%
25	fortran	129	0.15%
26	exp	101	0.12%
27	f90	71	0.08%
28	vhdl	33	0.04%
29	modula3	10	0.01%
30	cobol	9	0.01%

Table VI shows that scripting languages are ranked the first in terms of interoperability. The top-1, JavaScript, is used in over 18 thousands (21.81 %) of multi-language projects, followed by Shell (14.80%), Ruby (11.62%), and Python (9.22%). The C programming language, PHP, C++, and Java also appear in respectively 9.09%, 7.32%, 5.44% and 4.95% of the multi-language projects. On the other hand, Haskell and ML are less interoperable, appearing in 0.55% and 0.18% of the projects. The least interoperable languages are VHDL, modula3, and Cobol which, together, appear in less than 1% of multi-language projects.

We further survey the interoperability of languages by detailing the relationships among them to establish how languages interoperate and which ones are mostly used together. The relationship graph in Figure 4 gives an overview of how languages interoperate within projects. The thicker the edge between two nodes in the graph, the more projects contain code in the two corresponding languages. We observe that all languages interoperate with one another, though with different degrees of interoperability. Shell is the language that interoperates the most with other languages. Shell is indeed a scripting language that is used in many projects to write automation scripts for compilation, install, launch, etc. A subset of languages that consists of shell, ansi c, ruby, php, java, c++, c#, perl, javaScript, and python, are used together more often than the others. Objective C appears to be less interoperable with others. ML and Haskell have weak relationships with other languages.

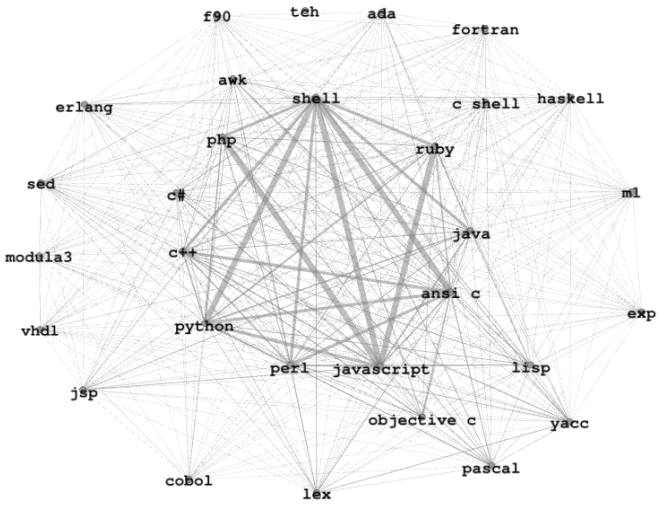

Figure 4. Language Interoperability—All Languages. Relationships among the 30 languages in Table I. The thicker a line between two languages, the more projects contain code written in these two languages.

Figure 5 focuses on the interoperability of a subset of mainstream languages. The graph shows that JavaScript interoperates well with php and ruby, two other common languages used in web programming.

Finally, in Figure 6, we explore the relationships between ANSI C and all other languages. The graph reveals that C interoperates well with many languages. This is possibly related to the fact that mainstream languages, such as C++, Java, Objective C, and C# are derivatives of C. Furthermore, we note that C and C++ interoperate the best with each other.

JavaScript, Shell and Ruby appear to be used together with most of the languages. Ansi C and its derivatives, which are syntaxically close, also interoperate very well. Web programming languages are often used together in projects.

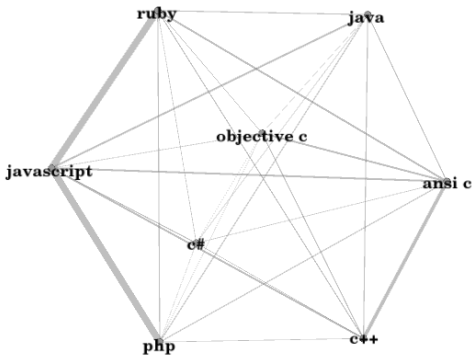

Figure 5. Language Interoperability—Mainstream Languages. Relationships among mainstream languages. The thicker a line between two languages, the more projects contain code written in these two languages.

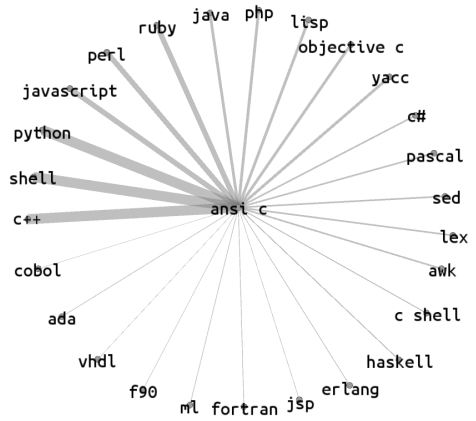

Figure 6. Language Interoperability—ANSI C. Relationships between ANSI C with other languages. The thicker a line between two languages, the more projects contain code written in these two languages.

C. RQ3: Programming Languages and Success

In the third research question, we investigate the languages used by successful projects. To estimate the success of a project, we rely on the amount of interest shown to the project that could be measured based on social coding features of GitHub.

1) Watchers: A project is found successful user-wise if many GitHub users *watch* its development. Table VII details the average numbers of watchers per project containing code in each programming languages. In this scenario, we consider only one main language per project to reduce outliers in multi-language projects. Objective-C is thus the language of the most number of watched projects: on average 91 watchers per project. The median number of watchers, i.e., 5, is also the highest. Erlang has the same median number of watchers as Objective-C, but there are 6 times more Objective-C projects than Erlang's. In terms of the average number of watchers per project, JavaScript-based and Ruby-based projects are also popular. Among mainstream compiled languages, C# and C++ appear the first, each of which has on average 23 watchers per project. Projects using C or Java as the main language have on average 19 or 16 watchers respectively. Once again, Modula3 and Cobol appear at the bottom of the ranking.

Table VII
Language and Project Success — Watchers. Means and medians of numbers of watchers for projects written in different languages.

Rank	Language	Mean	Median	Significance
1	objective c	91.6	5	+
2	javascript	65.8	2	+
3	ruby	60.5	2	+
4	erlang	52.6	5	+
5	sh	28.8	1	+
6	python	26.8	1	-
7	php	24.8	2	-
8	yacc	24.3	1	-
9	c#	23.6	1	+
10	c++	23.1	1	-
11	ansi c	19.7	1	+
12	lisp	17.3	1	+
13	java	16.8	1	-
14	ml	15.5	1	-
15	tcl	15.0	2	-
16	perl	12.2	1	-
17	haskell	10.0	1	-
18	fortran	9.9	1	-
19	f90	9.5	1	-
20	ada	8.9	1	+
21	pascal	7.2	1	-
22	lex	3.0	2	+
23	jsp	2.8	1	-
24	awk	2.2	2	-
25	csh	2.0	3	-
26	vhdl	1.9	1	-
27	sed	1.8	1	-
28	exp	1.0	1	+
29	modula3	1.0	1	+
30	cobol	1.0	1	

We use the Mann-Whitney-Wilcoxon (MWW) test, a non-parametric statistical hypothesis test, to assess the statistical significance of the difference between the distribution of projects for a given language and the distribution of projects for the next language in the table. The + sign, in the *Significance* column, indicates that the mean value for the corresponding language in a row is significantly higher than the mean for the language in the following row. Otherwise, a − sign is indicated. Projects written in Objective C have significantly more watchers than those written in JavaScript which in turn have significantly more watchers than those written in Ruby.

2) Forks: GitHub also tracks the number of forks for a given project. This number is a useful indication of the number of non-team developers volunteering to participate in the development of the project. Similar to the estimation of interest using the number of watchers, we present mean and median numbers of forks per project per language. The detailed results are presented in Table VIII. Objective-C still tops the ranking with an average of 12 forks and a median value of 2. The Lex and C-Shell languages have the same median value as Objective-C, but they are actually used in relatively fewer projects (Cf. Table III). JavaScript and Ruby are also still highly ranked. Overall, the results of analyzing interest using the number of forks are in line with those using the number of watchers.

Once again, the MWW test reveals that the differences of mean values are statistically significant for Objective C, Ruby, Erlang, JavaScript and a few more languages.

Table VIII
Language and Project Success — Forks. Means and medians of numbers of forks for projects written in different languages.

Rank	Language	Mean	Median	Significance
1	objective c	12.1	2	+
2	ruby	10.9	1	+
3	erlang	9.9	1	+
4	javascript	8.9	1	+
5	c#	6.4	1	-
6	php	6.1	1	+
7	sh	5.7	1	+
8	python	5.6	1	+
9	c++	5.2	1	+
10	java	5.1	1	+
11	lisp	4.4	1	-
12	ansi c	4.2	1	-
13	ada	3.7	1	-
14	pascal	3.6	1	-
15	tcl	3.0	1	-
16	perl	2.9	1	-
17	ml	2.8	1	-
18	yacc	2.6	1	-
19	fortran	2.4	1	-
20	haskell	2.4	1	-
21	lex	2.3	2	-
22	f90	1.6	1	-
23	csh	1.5	2	-
24	jsp	1.4	1	-
25	vhdl	1.4	1	-
26	exp	1.0	1	-
27	sed	1.0	1	+
28	awk	1.0	1	+
29	modula3	1.0	1	+
30	cobol	1.0	1	

Objective C and Ruby appear to be the languages that are used in projects that tend to draw the most interest. A quick sampling of projects written in Objective C shows that many of them are for iOS (iPhone) applications.

D. RQ4: Programming Languages and Issues

To investigate the correlation between programming languages and issues, we compute the mean and median numbers of issues per project for each language. Since many projects do not enable its issue tracker on GitHub as their development activities occur in their own websites where they have their own issue/bug tracking systems, we only consider those projects with at least 1 issue reported in GitHub. The results are presented in Table IX. C++ is the language used in projects with the most issues reported (64 issues per project on average, and a median value of 6). ML and C# follow in the ranking with an average of 55 and 50 issues per project.

Figure 7 shows the distribution of the number of issues for the top-10 languages drawn as boxplots. Each boxplot presents 5 vertical lines. From left to right, the first line indicates the MINIMUM, i.e., the least value, excluding outliers which are identified by the R system[9] (when constructing a *Modified Boxplot*). Data points on the left of this line are outliers (determined by the R statistical computing tool). The second line indicates the LOWER QUARTILE, i.e, 25 % of data points are on the left side of this line. The third line is the MEDIAN, the middle of the dataset. The fourth line indicates the UPPER QUARTILE, i.e., 25% of data points are on the right side of this line. Finally, the fifth line indicates the MAXIMUM, i.e., the greatest value, excluding outliers (determined by the tool). All data points on the right side of this line are outliers.

[9]http://r-project.org

Table IX
Language and Issues. Means and medians of number of issues for projects written in different programming languages.

Rank	Language	Mean	Median	Significance
1	c++	64.4	6	-
2	ml	55.3	3	-
3	c#	50.6	9	-
4	fortran	46.7	8	-
5	pascal	46.3	3	-
6	php	41.8	7	-
7	ruby	40.3	8	+
8	java	37.8	6	-
9	ansi c	36.7	10	-
10	python	35.7	7	-
11	sh	31.1	6	-
12	erlang	30.7	10	-
13	lisp	30.5	5	-
14	haskell	30.1	4	+
15	javascript	29.3	8	+
16	objective c	23.3	6	+
17	perl	21.9	3	-
18	tcl	20.8	7	-
19	yacc	15.0	10	-
20	ada	13.0	25	-
21	jsp	5.3	2	-
22	f90	3.0	1	-
23	vhdl	2.0	2	
24	csh	-	-	
25	sed	-	-	
26	awk	-	-	
27	lex	-	-	
28	exp	-	-	
29	cobol	-	-	
30	modula3	-	-	

Aside from Pascal, ML, and Fortran, which are used by few projects, the box plots are similar for the rest of the languages. This suggests that there is no clear correlation between a programming language and the number of issues for the language. Finally, the results of the MWW tests, which are included in the table, reveal that the differences are not, for most of the languages, statistically significant.

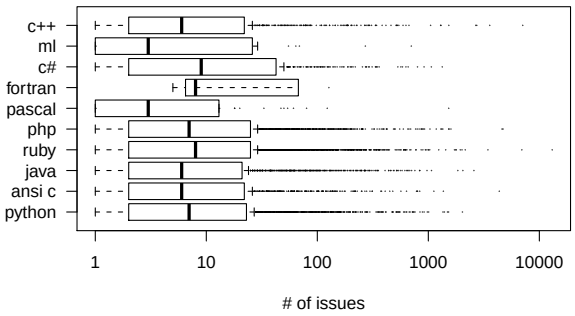

Figure 7. Number of Issue Reports for the Top-10 Languages with the Highest Average Number of Issue Reports

The prevalence of issue reports for a given project does not appear to be correlated to the programming language used to write the software code.

E. RQ5: Programming Languages and Teams

Finally, we investigate the constitution of development teams based on the used language, so as to identify languages that are more "collaboration friendly". We compute the average and median numbers of developers per project for each programming language.

Table X details the results on the average team size for each programming language. The C programming language is the top-1 language with an average of 46 developers per project but a median value of 1. Once again, this average is likely pulled up by operating system projects, such as the Linux kernel (9395 contributors as of June 2012), which usually involve a large number of developers. C++ ranks the second; Ruby, another "popular language" in terms of appearance in projects, is ranked the 5th.

Tcl, Erlang, Yacc, Expect, and C-Shell have a high median of 2, but are used in a small set of projects (Cf. Table III), making this finding statistically less reliable. Ruby, python, and JavaScript projects show very distinct distributions of team sizes and respectively have about 5 (5th), 4 (10th), and 3 (19th) developers on average per project.

In Figure 8, the boxplot shows that the distributions of team sizes may not be clearly correlated to languages. The significance tests also confirm this fact as few the differences of mean values are statistically significant for a limit number of languages.

Table X
Language and Team Size. Means and medians of team sizes for projects written in different programming languages.

Rank	Language	Mean	Median	Significance
1	ansi c	46.5	1	-
2	c++	9.4	1	-
3	tcl	7.7	2	-
4	erlang	6.1	2	+
5	ruby	5.3	1	+
6	sh	4.8	1	-
7	java	4.7	1	-
8	perl	4.7	1	-
9	c#	4.5	1	-
10	php	4.3	1	+
11	python	4.1	1	+
12	lisp	4.1	1	-
13	ml	4.1	1	-
14	haskell	3.7	1	-
15	pascal	3.5	1	-
16	fortran	3.2	1	-
17	ada	3.0	1	-
18	objective c	2.9	1	+
19	javascript	2.8	1	-
20	yacc	2.4	2	-
21	exp	2.3	2	-
22	vhdl	2.0	1	-
23	jsp	1.8	1	-
24	f90	1.8	1	-
25	lex	1.7	1	-
26	sed	1.6	1	-
27	csh	1.5	2	-
28	modula3	1.3	1	-
29	awk	1.0	1	+
30	cobol	1.0	1	

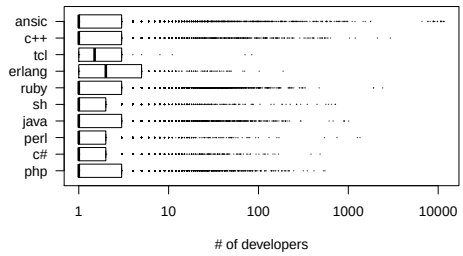

Figure 8. Size of Development Teams for the Top-10 Programming Languages with the Highest Average Size of Development Teams

The top-3 languages with projects having highest average team sizes are Ansi C, C++ and TCL. Ansi C has a high mean due to the presence in our dataset of highly distributively developed projects, such as Linux, whose code are in Ansi C.

V. Threats to Validity

This study bears some threats to validity mainly related to the datasets we rely on.

Origin of datasets: Our empirical findings are based on open source projects found on GitHub which lead to results that may not generalize to the universe of software projects produced by developers. We have not considered legacy projects that are still in use but not actively developed on GitHub either. Other practices behind the corporate wall may lead to a different popularity ranking of the languages that programmers are paid to work with. Nonetheless, our findings remain relevant as many companies, such as Facebook and Google, release some of their software as open source and invest in open source projects that appear on GitHub. Furthermore, companies are likely to hire developers based on the skills they can justify and that can be leveraged, implying that languages used for open source development are unlikely unrelated to those used in corporations.

Size of datasets: This study is limited to 100,000 projects. Though a sizeable sample, it cannot equate the millions of software programs whose code can be retrieved from the World Wide Web [1]. GitHub alone contains over 1 million public repositories. It is to be noted however that our findings on 100,000 projects, in the case of language appearance in projects, are inline with the ranking provided by GitHub[10]. This suggests that our sample dataset is representative of the universe of GitHub projects.

VI. Related Work

The history, the fundamentals, the trends, and the evolution of programming languages have been the subject of numerous studies in the literature. As early as 1963, the proliferation of programming languages has lead Rosen Saul to produce a "historical" survey of programming languages [9]. That study was then enriched in 1966 to account for further developments [10], and the author has proposed a 10-year summary in times where Cobol and Fortran were the languages used in most of the world production systems and were believed to remain so into "the foreseeable future" [12]. Sammet has also discussed in this epoch the fundamentals of programming languages, addressing both used languages and unimplemented concepts [13].

A number of research work has also gone into defining what makes a programming language popular while others have proposed explanations to how some languages have outlived community expectations and others were early abandoned. In his work on the longevity of languages,

[10]https://github.com/languages

Mashey has suggested that early successes in programming languages such as C have built such an ecosystem that it may have become prohibitively costly to move to new, and possibly, better languages [8]. In earlier work, Wadler has discussed why no one uses functional languages [16]. We have however found that, 15 years later, Haskell and ML have not disappeared and are still lead languages in a significant portion of software development projects. More recently, Derk has written an essay from a historical perspective [3] where she concludes that the language quality itself is not important, one must rather look at the application domain the language is fit for. This suggestion appears to relate to some of our findings: JavaScript and Ruby are fit for web programming which, with today's rush into social media, is a strongly evolving application area. Thus the popularity of these languages appears to be high, despite common critics on different aspects of the languages.

Listing programming languages is relatively easy. Many websites, such as Wikipedia, enumerate hundreds of languages. Ranking languages in terms of popularity is however more challenging. Most attempts have indirectly inferred the popularity of languages by using different non-programming indicators: what languages are most sought in the job market; which ones are most referred to in fora; etc.

Ranking based on search engines' hits: The TIOBE software research firm, based in the Netherlands, publishes every month the TIOBE programming community index [15] where they provide the trends for Turing Complete [14] programming languages that have entries in Wikipedia. They base their index on the number of hits from Alexa's[11] top web sites with search facilities, including known search engines such as Google, Bing, Yahoo! and Baidu, and others: Blogger, wikipedia, Amazon, Youtube, etc. In their June 2012 report, TIOBE's analysts confirm the increase in trend for the Haskell programming language that they have been observing for some time. Nonetheless, it is to be noted that the TIOBE index is not about the *best* programming language or the language in which *most lines* of code have been written. Instead, TIOBE's index is an indication of the buzz surrounding a programming language.

Ranking based on real usage: The above depicted websites indirectly investigate the popularity of programming languages supposedly because it is impossible to "look over programmers shoulders and note what languages they're coding in" [6]. Yet, the momentum of Open Source software has flooded the world wide web with all-size, all-type and all-purpose software projects in repositories that track information on every aspect of the development. GitHub is one example of hosting platform where such studies can be conducted on real-world projects. To the best of our knowledge, Drew Conway was the first to exploit information from GitHub to investigate the popularity of programming languages [2]. He directly relied on GitHub's language popularity ranking which is based on the number of projects they appear in, and compares[12] it with data on developer questions on StackOverflow. In our study however, we investigate beyond the *appearance* in projects. We consider the actual usage of the language in the code base compared to other the languages.

[11] http://www.alexa.com

[12] See http://redmonk.com/sogrady/2012/09/12/language-rankings-9-12/

VII. Conclusion and Future Work

Programming languages come and go. Only a few are adopted and thrive. While many scientific studies have discussed how to create a good programming language and why some languages become popular, less attention has been paid to the actual popularity of languages.

In this paper, we describe the findings of our empirical study on a sizeable dataset of 100,000 projects. This study corroborates different assumptions made in the literature on the popularity of programming languages. We have found that earlier popular languages, such as C, are still current with a large code base, while the rush in web development has made JavaScript and Ruby pervasive. Finally, Objective-C, a language tightly related to the products of a successful vendor, namely Apple, has also been gaining much attention.

In future work, we plan to investigate the evolution in the popularity of programming languages. We also plan to consider a bigger dataset with 1 million projects from both GitHub and other project hosting sites.

References

[1] J. M. Bieman and V. Murdock, "Finding code on the world wide web: A preliminary investigation," in *SCAM*, 2001.

[2] D. Conway, "Ranking the popularity of programming languages," http://www.dataists.com/2010/12/, 2010.

[3] M. Derk, "What makes a programming language popular?: an essay from a historical perspective," in *ONWARD*, 2011.

[4] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[5] J. Fuegi and J. Francis, "Lovelace babbage and the creation of the 1843 'notes'," *Annals of the History of Computing, IEEE*, vol. 25, no. 4, pp. 16–26, 2003.

[6] R. S. King, "The top 10 programming languages," http://spectrum.ieee.org/at-work/tech-careers/.

[7] J. Loeliger, *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly, 09.

[8] J. R. Mashey, "Languages, levels, libraries, and longevity," *Queue*, vol. 2, no. 9, pp. 32–38, Dec. 2004.

[9] S. Rosen, "Programming systems and languages: a historical survey," in *AFIPS*, 1964, pp. 1–15, reprinted in [11].

[10] ——, *Programming systems and languages. Some recent development*, 1966, in [11], pp. 23-27.

[11] ——, *Programming systems and languages*. McGraw Hill, 1967.

[12] ——, "Programming systems and languages 1965-1975," *Commun. ACM*, vol. 15, no. 7, pp. 591–600, Jul. 1972.

[13] J. E. Sammet, *Programming Languages: History and Fundamentals*. Prentice-Hall, Inc., 1969.

[14] M. Sipser, *Introduction to the Theory of Computation*, 1st ed. International Thomson Publishing, 1996.

[15] TIOBE, "Tiobe programming community index definition," http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm.

[16] P. Wadler, "Why no one uses functional languages," *SIGPLAN Not.*, vol. 33, no. 8, pp. 23–27, Aug. 1998.