An Empirical Analysis of the Utilization of Multiple Programming Languages in Open Source Projects

Philip Mayer
Programming & Software Engineering Group
Ludwig-Maximilians-Universität München
Germany
mayer@pst.ifi.lmu.de

Alexander Bauer
Statistical Consulting Unit
Ludwig-Maximilians-Universität München
Germany
bauer.alexander@campus.lmu.de

ABSTRACT

Background: Anecdotal evidence suggests that software applications are usually implemented using a combination of (programming) languages. **Aim**: We want to provide empirical evidence on the phenomenon of multi-language programming. **Methods**: We use data mining of 1150 open source projects selected for *diversity* from a public repository to a) investigate the projects for number and type of languages found and the relative sizes of the languages; b) report on associations between the number of languages found and the size, age, number of contributors, and number of commits of a project using a (Quasi-)Poisson regression model, and c) discuss concrete associations between the general-purpose languages and domain-specific languages found using frequent item set mining. **Results**: We found a) a mean number of 5 languages per project with a clearly dominant main general-purpose language and 5 often-used DSL types, b) a significant influence of the size, number of commits, and the main language on the number of languages as well as no significant influence of age and number of contributors, and c) three language ecosystems grouped around XML, Shell/Make, and HTML/CSS. **Conclusions**: Multi-language programming seems to be common in open-source projects and is a factor which must be dealt with in tooling and when assessing development and maintenance of such software systems.

1. INTRODUCTION

The software engineering community has come up with numerous programming languages for the various tasks involved in software construction. Anecdotal evidence suggests that software projects often employ combinations of languages for system implementation, which includes general-purpose languages (GPLs) such as C++, Java, or Ruby, and domain-specific languages (DSLs) such as HTML, Make, XML, or SQL. The utilization of multiple languages within one project means that for fully understanding, analyzing, and manipulating the project — which includes not only understanding the code for the runtime of the system but also the construction parts such as building and testing — a developer must have a grasp of each of these languages. Furthermore, if multiple languages are used, they mostly do not stand alone — instead, artifacts in each language reference one another, which increases the complexity of the software and requires care during maintenance [1].

It is thus important to gain an understanding of how language combinations are used in practice. We believe that this knowledge is an enabler for many efforts related to development and maintenance, such as understanding and visualizing software architectures, creating tool support for developing and restructuring software, and finally understanding system runtime behavior, all of which are affected by separating concerns into different languages.

Our aim is to aid this understanding with the work at hand, in which we supply empirical evidence on the phenomenon of multi-language programming with an investigation of 1150 open source software projects retrieved from the hosting site GitHub[1]. We have structured our investigation along the following three research questions.

- *RQ1.* How many languages (GPLs and DSLs) are commonly used in open source software, and what is their relative code size and (for DSLs) their type?
- *RQ2*: Does the number of languages depend on one of the following other project properties: size, main language, age, number of commits, and number of contributors?
- *RQ3*: Which association patterns can be found between the languages (and in particular, GPLs and DSLs) used in the projects?

The answer to the first question serves to illuminate the problem domain and language usage in general. With the second question, we investigate relationships between project properties. The third question looks at GPL-GPL and GPL-DSL associations across the data set, identifying co-occurring languages and language groupings by association frequency. This information identifies common sets of languages that merit attention (e.g. for tool support).

We outline the methods used for project selection and analysis in the next section. Sections 3 and 4 contain the results from our analysis and a discussion of the implications and conjectures we can draw from the data, respectively. We conclude in section 5.

[1] `www.github.com`

2. METHODS

Our analysis is based on concrete real-life open source projects. To be able to find the effects of different shapes and forms of these projects on the number of languages, we used a combined process of *random selection* and *optimal design* (or *theoretical sampling*). The first is used to reduce the number of candidates, while the second is aimed at selecting projects with a maximally *diverse* set of values over interesting attributes (as listed in RQ2). Both are discussed in section 2.1.

The number and names of the languages occurring in a project are not readily available; they must be detected from the source code, a process we describe in section 2.2.

Once selected, we have analyzed the project set by statistical methods, which are described in section 2.3. Besides the numbers, it is also interesting to look at the *concrete* programming languages; we used a frequent item set mining approach, which is described in section 2.4.

The complete data and the source code which creates the data is available on our web page[2].

2.1 Data Collection

2.1.1 Random Project Selection

In this first step, we randomly selected projects from GitHub, which is possible since each project is stored with a numerical identifier. By creating a new project and querying its identifier, we discovered the current maximum number of projects, which was then used as an upper bound for random number generation.

We used the GitHub web service API for querying project information for each generated ID. The information required for each project for answering the research questions is shown in Table 1.

Table 1: Interesting Project Properties

Property	Description
Main language	The general-purpose language with the largest amount of code (in bytes)
Size	The size of the project (bytes written in the main language)
Age	The age of the project since the first commit (in months)
Contributors	The number of (unique) people who contributed to the project
Commits	The total number of commits to the source code repository

Gathering this information requires several API (HTTP) requests; one for the primary metadata (such as name, owner, etc.); one for the languages involved, and a variable number of requests for the commit history (depending on how many commits there are; the information is paged). We call this the *initially queried project set*.

A script was written to download this information, taking care of the limitations imposed by GitHub (5'000 requests per hour for registered users) as well as all error conditions (some projects do not have contributors or languages, etc).

The resulting set of metadata was then processed with several *inclusion criteria* defined as follows:

- The project must (still) exist and be publicly accessible, i.e. the request may not result in a 404 error.
- The project must contain a general-purpose programming language (there are also other projects on GitHub such as documentation efforts).
- The project may not be a *fork*. This avoids duplicates.
- The project must have a minimum of 500 and a maximum of 100 million bytes of code (both in the main language). This avoids both "Hello World" projects and one-of-a-kind large projects.
- The project must have consistent metadata (i.e. at least one contributor and at least one commit).

After applying these inclusion criteria to the *initially queried project set*, we get what we call the *randomly selected project set*, which serves as input for the next step.

2.1.2 Selecting a Diverse Project Set

Our aim in this study was analyzing language occurrence in projects which are maximally *diverse*, that is projects of different forms and shapes, spread out as far as possible over the input parameters listed in RQ2. To achieve this, we used the algorithm provided by Nagappan et al. [2]. In a nutshell, their approach describes how to select a sample of projects to maximize the *diversity* by selecting only one from a set of similar projects — where similarity is defined by certain similarity functions given below.

Another option would have been selecting a smaller random sample, which would have offered *representativeness* for *all* research questions (for GitHub). We have explicitly opted for diversity since we are interested in language effects over as many shapes and forms of projects as possible. A second reason is that GitHub has its own biases; a large part of projects uses JavaScript as the main language, and there are many small and/or abandoned projects, which would have taken up a large part of a random sample. Note, however, that the sample we have taken is still representative of GitHub for the (Quasi-)Poisson regression which we have used for answering RQ2 (see section 2.3).

In the terms of Nagappan et al., our universe (or population) consists of the metadata of the *randomly selected project set*. We have characterized these projects along the dimensions shown in Table 1. Our *configuration*, i.e. the similarity function determining whether two projects are similar, is defined by equality in the following attributes.

- Main language. Languages are split into 13 categories, with 12 each representing a well-known GPL (the first 12 from Table 3), and the last representing all others.
- Project size (bytes in the main language): 5 groups based on orders of magnitude: Tiny (less than 1000 bytes), Small (1k to 10k), Middle (10k to 100k), Large (100k to 1m), and Very large (over 1m).
- Project age, split into 3 groups: Young (less than one year), Middle (1 to 4 years), Old (5 or more years).
- Number of contributors, split again into 4 groups: Single person (one contributor), small team (2 to 7), medium team (8 to 100), large team (over 100).

[2] `www.xllsrc.net/languagestudy`

- Number of commits, split into 4 groups: Single check-in (one commit), very low (2 to 10), low (11 to 100), medium (101 to 1000), and high (over 1000).

The algorithm was able to fully cover the universe with a reasonable amount of projects for analysis as we will discuss in the results section. We call this set the *final input set*; it was used for all analyses (and thus for all research questions).

2.2 Language Detection

Part of the data available on GitHub is the main language of a project and a list of other used languages. Unfortunately, this list cannot be used for our analysis since it only includes *"programming languages and acceptable markup languages,"* as the source code of *Linguist*[3], the tool GitHub uses for determining the language of source code files, states[4]. Unfortunately, the *unacceptable* languages include very common languages such as XML and HTML which are indeed interesting to us.

However, Linguist also allows the discovery of languages on a file-by-file basis where it does not apply the exclusions discussed above. We have therefore ascertained our own count of languages and sizes by acquiring the source code of each project and detecting the language by file-by-file invocations of Linguist.

We were thus able to re-use this tool which was very helpful since Linguist not only includes detection mechanisms for a large amount of languages but also contains other helpful features which improve the quality of the data. First, Linguist detects languages used for documentation purposes, such as *reStructuredText* or *MarkDown*. These languages are correctly considered to be *prose* by Linguist and were excluded from all language counts. Secondly, Linguist attempts to exclude *generated* files (for example, files generated by the `XCode` IDE or by the Java `JNI` tool) which are identified by name or by text within the file. Finally, Linguist excludes so-called *vendored* files which are well-known libraries included in source (such as the `jQuery` library for JavaScript).

Thus, for all projects in the final input set, the latest version was retrieved from the repository URL provided by GitHub. Each project then was analyzed file-by-file and the results accumulated and stored for further analyses.

2.3 Statistical Analysis

For our first glimpse into the data set and for answering our first research question, simple descriptive statistics such as the mean and the interquartile range suffice.

However, for answering research question 2, we require more complex methods, since we attempt to analyze the associations between the different attributes of our input data. As we have discussed in section 2.1, we use a two-step process of project selection: the first step selects random projects, while the second uses theoretical sampling and aims at selecting a maximally diverse set of projects based on different parameters of the input data.

This second step suggests using a regression analysis including the exact same parameters we used for selection (main language, size, age, number of contributors, and number of commits) as the covariates since it is thus permissible to generalize the results from the analysis. The regression analysis tests the influence of several project properties on the number of languages, i.e. if there is a significant change in the number of languages given a change in one of the input parameters.

In detail, we perform a Poisson regression analysis [3] and employ a (Quasi-)Poisson model with the number of languages found by the Linguist tool as the response variable. Regarding the covariates, all except the main language are metric; the main language itself is a categorical variable and was implemented in the model using dummy coding. The Poisson regression is the standard model for modeling count data as it assumes that the response variable follows a Poisson distribution. In contrast, a linear regression analysis would assume a normal distribution for the response and is therefore only applicable for metric variables.

We used the `R` statistical package with the `mgcv` (Mixed GAM Computation Vehicle) library[5] for this analysis.

2.4 Association Rule Mining

While the statistical analysis gives us answers in terms of numbers and associations, it does not give insights into the *actual* languages and language types. Since research question 3 is about associations between individual languages, we require another approach for querying the search space.

We employ frequent item set mining as well as association rule mining using variants of the two-step approach proposed by Agrawal and Srikant [4] (Apriori). In particular, we use the FP-Growth algorithm for frequent item sets and the "faster algorithm" for association rule mining described in the above paper. For both algorithms, we use the implementations of the SPMF library[6].

We use *languages* as the *items* and *projects* (in the sense of combinations of languages) as the *transactions*; in other words, we discover *frequent sets of languages* across all projects. Of particular interest in our case are one-item sets and the *association rules* between one-item GPL sets and their associated DSLs.

3. RESULTS

3.1 Data Collection

At the time of data collection, the highest repository ID on GitHub was 25'855'878. For data discovery, we thus selected random IDs between 1 and 25 million. Over the course of one week, we downloaded the metadata of 500'000 repositories, which is our *initially queried project set*.

Our defined inclusion criteria were now applied to this project set, which reduced the set to 82'547 projects (the *randomly selected project set*). The reasons for exclusion are shown in Table 2. Most queries resulted in a *resource not found* (404) error; this means either a) the project was deleted or b) the project is private. A sizeable amount of projects are forks, followed by projects that are not software systems (no programming language).

A total of 3'232 projects were too small or too large, with the overwhelming number being too small (3'045). Other reasons are technical in nature (no contributors found, no commits found, project access is forbidden due to DMCA, no size found for main language, etc.).

[3] `github.com/github/linguist`

[4] `github.com/github/linguist/blob/master/lib/linguist/repository.rb\#L162` (line 162)

[5] `cran.r-project.org/web/packages/mgcv`

[6] `www.philippe-fournier-viger.com/spmf`

Table 2: GitHub request results

Initial Amount	**500'000**
404 (project no longer exists or is private)	215'227
Project is a fork (i.e. a duplicate)	126'220
Project does not contain a GPL	71'339
Project too small / too large	3'232
Other reasons	1'435
Selected repositories	**82'547**

This set was then processed with the algorithm of Nagappan et al. [2], which completed with the *final input set* of *1150 projects* that fully represent the input space. These projects were then checked out (around 38 GB of data) and analyzed with the Linguist tool for language occurrence.

3.2 Basic Language Co-Occurrence Data

Our first results relate to the basic numbers of language co-occurrence in the *final input set* of 1150 projects. Linguist has found 151 non-prose languages in total in the projects; we list all languages occurring in at least ten projects in Table 3. For the DSL languages, the *type* is given in the table as well.

Table 3: Single language occurrence in the input set, with project count and language type for DSLs. UI= User Interface; SD= Structured Data; DB= Database; i18n= Internationalization; Stats= Statistics/Math; Trans=Transformation

GPL	**Pr**		**DSL**	**Pr**	**Type**
JavaScript	368		XML	501	SD
C	265		HTML	370	UI
C++	242		CSS	348	UI
Python	229		Shell	321	Shell
Ruby	181		Make	243	Build
Perl	180		JSON	239	SD
Java	167		YAML	204	SD
PHP	154		INI	192	Config
Objective-C	131		Batchfile	113	Shell
C#	96		SQL	80	DB
CoffeeScript	79		Groff	79	Text
Scala	58		HTML+ERB	50	UI
Erlang	28		ApacheConf	49	Config
GAS	28		Gettext	45	i18n
Go	25		Diff	44	Diff
D	21		SCSS	44	UI
Groovy	20		XSLT	41	Trans
Assembly	14		Sass	35	UI
C. Lisp	13		Less	24	UI
FORTRAN	12		PowerShell	22	Shell
Visual Basic	12		R	20	Stats
Scheme	11		ASP	18	UI
ActionScript	10		VimL	18	Script
Tcl	10		Emacs Lisp	17	Script
			CMake	16	Build
DSL	**Pr**	**Type**	Puppet	16	Config
Smarty	15	UI	Jade	14	UI
Haml	13	UI	NSIS	13	Install
JSP	11	UI	Lua	10	Script

A general overview of the language numbers found in the projects is shown in Figure 1. The three boxplots shows the occurrence of GPLs, DSLs, and of their combination (i.e. all languages). For each group, the values of the five-number summary are shown. As usual, the bar represents the median and the box encloses the first and third quartiles. The whiskers present the lowest and highest datum still within 1.5 interquartile range. Outliers are marked "o".

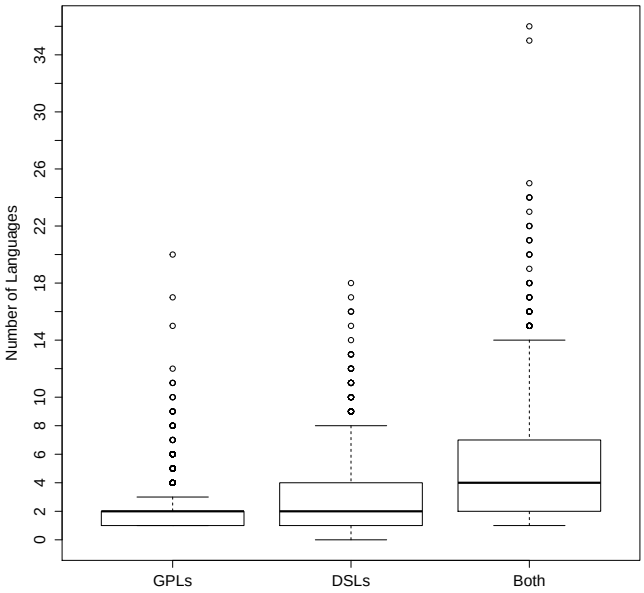

Figure 1: Plot of the number of languages found across all projects

The medians (shown as thick bars in the figure) are 2 for GPLs, 2 for DSLs, and 4 for all languages, respectively. The means and standard deviation are 2.12±1.79, 3.05±2.86, and 5.17±4.3. As usual, 75% of the values lie within the whiskers, which means between 1 and 3 GPLs, 0 and 8 DSLs, and a total of 1 to 14 languages. There are also several outliers, ranging up to 36 languages, which is the maximum found in any project. There are only 2 projects with over 30 languages; both use C as the main language.

For comparing the *relative sizes* of languages within the projects, we use the output of Linguist, which is in source lines of code (SLOC). The results show that the mean number of lines of code of the main GPL compared to all GPLs per project is 91%±14 — thus, in most cases we have a clearly dominant main GPL language. If we compare the lines of code of languages in the GPL group with those of the DSL group, we find the GPL lines of code to amount to a mean of 74%±27 of the DSLs; i.e., about 3/4th of a project's code is written in GPLs on average (albeit with a rather high standard deviation).

For further insights into *DSL usage* it is interesting to look at the *types* of DSLs present in each project, and the number of languages present with each type. There are five DSL types which occur in 100 or more projects; these are *Structure Data* (for example, XML), *User Interface Description* (for example, HTML), *Shell* (for example, Bash), *Build* (for example, Make), and *Configuration* (for example, .INI).

Looking at the number of languages per type per project, we find that the mean of occurring languages lies between 1.0 to 1.35 for all types except UI, where the value is 2.0. The mean numbers for *Build* and *Configuration* are very close to one — they are 1.03±0.17 and 1.08±0.27, respectively. For *Structured Data* and *Shell*, we get slighter higher values of 1.35±0.60 and 1.22±0.42. The languages in the user

interface type have the highest mean as well as the highest standard deviation (2.0±1.07).

All other DSL categories — that, is, those occurring in less than 100 projects — have means of about one.

3.3 Regression Analysis

In this step, the final input set was subjected to a Poisson regression analysis with main language, size, age, number of contributors, and number of commits as the covariates and the number of languages as the response variable.

Creating a Poisson model requires several decisions. The first relates to the coding of variables. Regarding the covariates, it appears that size, age, the number of contributors and the number of commits are metric variables. However, the main language is measured on a nominal scale with a total of 48 values. This variable was thus factored; we used dummy encoding to compare each language category against a reference, for which we selected the Java language. As there are many categories that are represented by only very few projects in our sample we furthermore decided to group main languages with less than five projects each into a single category named "Other", reducing the number of language categories to 20. Note that this only refers to the *main GPL* of a project (the one with the highest number of bytes written in the language), not to the overall list of GPL appearances shown in Table 3.

With the Poisson model being designed for modeling count data including zero values it was more adequate to use the number of *additional* languages (i.e., the number of languages subtracted by one) as the response variable, as each project includes at least one language. Since this does not change the model as regards to content, we still refer to this variable as the number of languages.

To avoid the problem of overdispersion, which can occur while using a Poisson model, we also tested a Quasi-Poisson model in our analysis. Overdispersion means that the variance of the response variable is greater than its expected value. In such situations, a Poisson regression is not adequate as it assumes that the variance and the expected value of the response are equal. The Quasi-Poisson model is a modification of the normal Poisson model and estimates an additional parameter — the dispersion parameter — which is multiplied with the variance from the Poisson model. If this parameter is greater than 1, a normal Poisson model should not be used as overdispersion exists in the data. In our case, with the dispersion parameter being 2.07, the Quasi-Poisson model seems clearly more adequate for our data than a pure Poisson regression.

For the metric variables (age, number of contributors, number of commits, and size) only few high values appear; we thus considered it appropriate to use the log_{10}-transformed variables in the regression model to improve the quality of the model and the interpretability of the estimated effects. Since the age variable is measured in months and includes zero, we used age increased by one, which allows us to use the logarithm while keeping the origin of the variable.

We used smooth functions to estimate the effect of the above-mentioned metric variables except the project age; the latter was implemented as a linear term in the final model as the estimated smooth function only led to a linear function. The deviance explained by our final Quasi-Poisson regression model is 50.2%, which is an acceptable value.

For each variable, the Quasi-Poisson regression estimates the effect of the variable on the number of languages, given that all other covariates stay constant.

Table 4: Results from the Quasi-Poisson Regression Model with the outcome variable *number of additional languages*. edf = Estimated Degrees of Freedom. Significance: ** = <0.01, * = <0.05, . = <0.1

Variable	**exp(Estimate)**	**Significance**
log_{10}(Age)	0.918	0.06 .
Variable	**edf**	**Significance**
log_{10}(Contributors)	1.554	0.59
log_{10}(Commits)	2.811	<0.01 **
log_{10}(Size)	2.433	<0.01 **
Main Language	**exp(Estimate)**	**Significance**
ActionScript	0.368	0.09 .
C	1.328	<0.01 **
C#	0.834	0.13
C++	1.167	0.12
CoffeeScript	1.568	<0.01 **
Common Lisp	1.098	0.67
FORTRAN	1.18	0.49
Go	0.588	0.07 .
Groovy	2.195	<0.01 **
Haskell	0.791	0.58
JavaScript	1.529	<0.01 **
Objective-C	1.112	0.36
Other	1.194	0.19
Perl	0.971	0.81
PHP	1.118	0.26
Python	0.971	0.77
Ruby	1.447	<0.01 **
Scala	1.112	0.45
TypeScript	1.965	<0.01 **

These results are shown in Table 4. The covariate names are shown on the left (log_{10} where appropriate). The middle column shows the exponential of the estimate for age and the main language categories, and the estimated degrees of freedom (edf) for the smooth functions of the number of contributors, number of commits, and size. The right column shows the significance. As indicated, age and number of contributors do not have a significant influence on the number of languages found in a project, whereas the influence of size and number of commits is significant.

As mentioned above, our analysis of the influence of the main programming language is based on dummy coding with the language Java as the reference category. The data shows that projects with most other main languages have no significantly different amount of languages. Regarding the categories with significant effects, JavaScript, CoffeeScript, and Ruby show about 52%, 56%, and 44% more languages on average than Java, while TypeScript shows an increase of 96%. C- and Groovy-based projects show about an increase of 32% (C) and even 119% (Groovy) of the languages of Java-based projects.

A graphical representation of the estimated smooth functions is shown in Figure 2, with the solid line representing the estimate which is enclosed in dashed lines representing the 95% confidence interval of the function, while the y axis shows the estimated effect of the response variable for a given x. The first function (number of contributors) shows

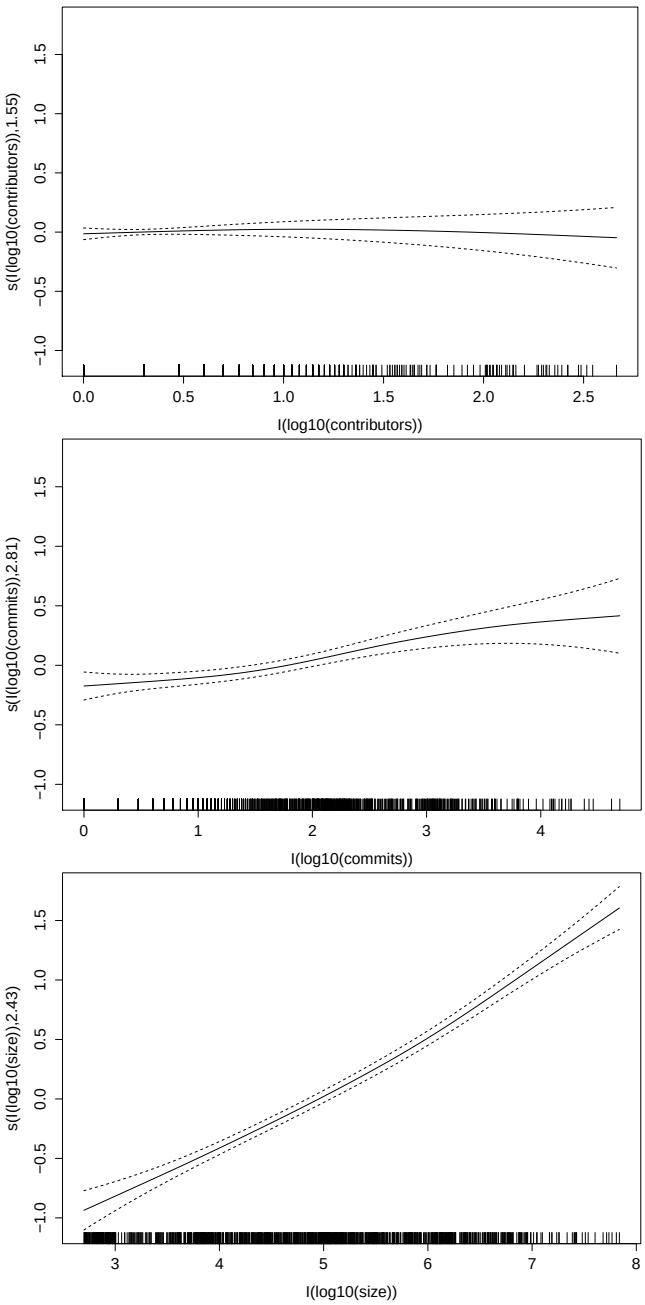

Figure 2: Estimated effect of the number of contributors, number of commits, and the size of the main language on the number of languages

little deviation from zero across the number of contributors and is not significant. The second and third functions shows a clear dependence of the number of languages on the number of commits and on the size of the main language, respectively.

3.4 Language Associations

For frequent item set mining and association rule mining, we first had to select appropriate parameters for support and confidence. As research question 3 states, our interest are GPL-to-GPL and GPL-to-DSL associations, i.e. association rules between single languages and thus single-item sets.

Accordingly, we chose a low minimum set support value of 5% to include as many language sets as possible, which means that a set must occur in around 57 projects to be recognized. As the association rule confidence value, we chose a higher value of 45% to only include associations with a good basis in the data. Thus, we gain association rules between sets of languages in which, given a first set `A`, set `B` occurs in at least 45% of those projects in which `A` occurred.

From the resulting data, we selected the association rules between the top 10 GPLs according to our project table (Table 3) (left side of the rule) and any (single) language (on the right side of the rule) with a confidence of at least 45% for further inspection. The result is best shown graphically and is thus displayed in Figure 3.

Each *rectangle* in the figure represents one of the top 10 general-purpose languages. The number indicated in each rectangle is the support for the single-item frequent item set with this language; that is the percentage of projects in which this language occurred. This number is also used for the size of the rectangle. For example, JavaScript is used in 32% of projects in the input set which is the highest count; it is thus the largest rectangle. Each *circle* represents a domain-specific language that has an association rule from a general-purpose language with a confidence of at least 45%. Again, the circle size represents the support for the single-item frequent item set (also shown numerically).

The arrows between the languages represent association rules. Light-grey arrows show an association rule between GPLs, while black arrows show rules between a GPL (source) and a DSL (target). Association rules between DSLs or from DSL to GPL are not shown. Each arrow has a label that shows the confidence for this rule. The thickness of the arrow represents this size, scaled from the lowest value (45%) to the highest (90%).

4. DISCUSSION

4.1 Answering the research questions

4.1.1 RQ1

We begin with answering our first research question: *How many languages (GPLs and DSLs) are commonly used in open source software, and what is their relative code size and (for DSLs) their type?*

We can answer this question with the basic language co-occurrence data selected by our dual random/theoretical sampling approach, and thus over projects with differing age, size, main languages, number of contributors, and number of commits. As discussed in the results section, the median number of GPLs, DSLs, and all languages is 2, 2, and 4, respectively, while the means and standard deviations are 2.12±1.79, 3.05±2.86, and 5.17±4.3. The maximum number of languages found in any one project is 36.

Intuitively, the low number of GPLs per project makes sense: We can expect one GPL to be used as the main programming language of the project, and the use of additional ones for specific tasks (such as using JavaScript for improving the client user interface, or using C for implementing native functionality in interpreted languages). In fact, we already saw in the result sections that one GPL usually dominates all others with 91%±14 of all GPL code.

Interestingly, the mean number of DSLs found is not much higher than that of the GPLs. This seems surprising given

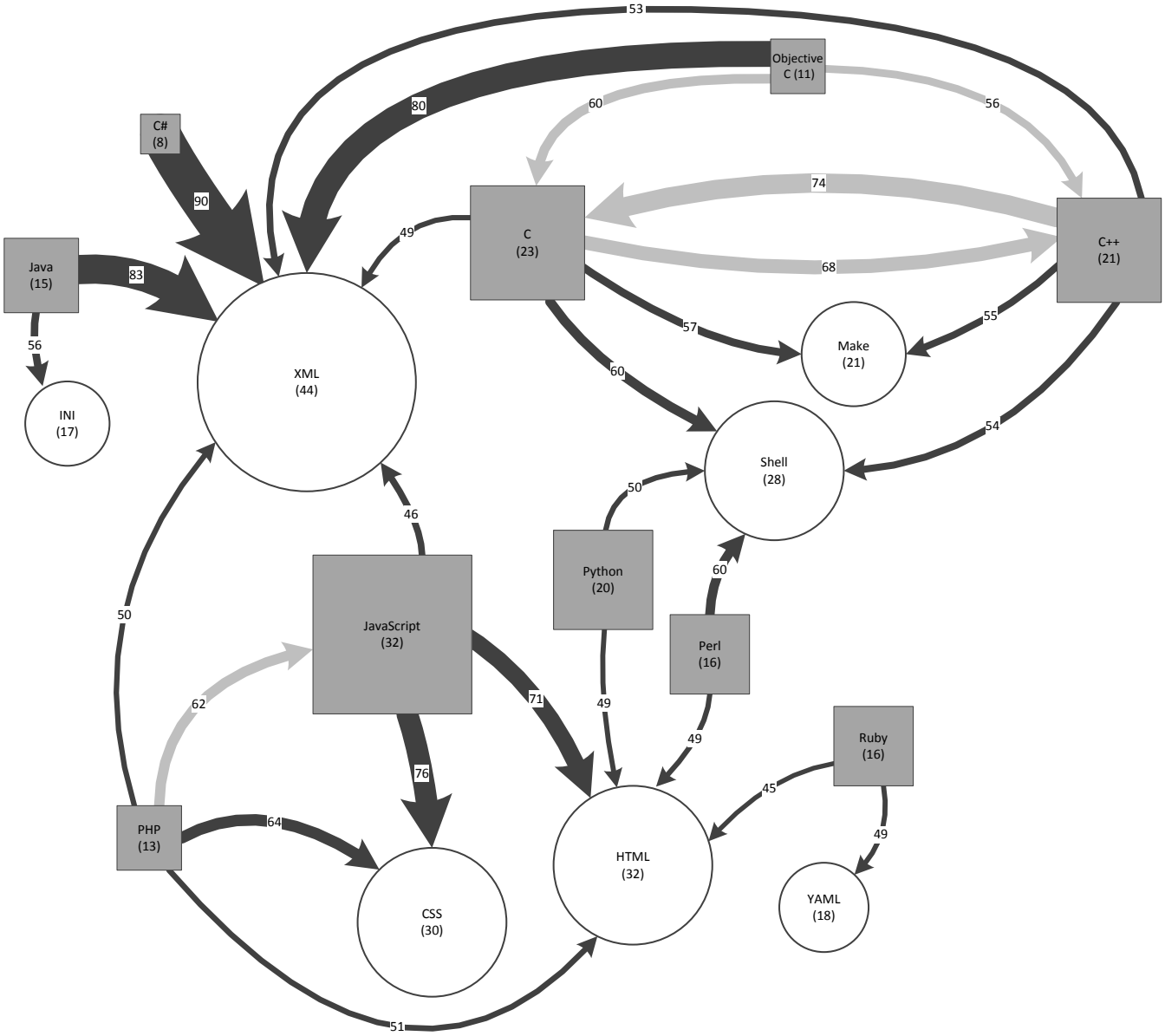

Figure 3: Single-Item Frequent Item Sets and Association Rules between Languages

the vast number of DSLs out there. We discuss two probable reasons for this. First, many DSLs share the same *type*, i.e., they are used for one certain purpose and thus we can expect to see only one of these languages in any given project. We have found five often-used DSL categories, which are *Structured Data*, *User Interface Description*, *Shell*, *Build*, and *Configuration*. The categories in themselves are not surprising. Only the UI category has a median (and mean) of languages per project of 2; all other categories have only one language per project on average.

A second and more technical reason lies in the way languages are detected. Some of the most-used languages in our projects include XML, JSON, and YAML. All three are languages that can be used for multiple purposes ranging from data storage via configuration and messaging to actual executable code. These different *dialects* are not detected by Linguist. The detection mechanism also has no support for DSLs that are used inline (such as SQL) or those handcrafted for a specific project or purpose, and for the use of multiple languages within one file. Detecting such languages would mean a more thorough investigation of files, possibly even with an understanding of the syntax of each individual language, which is a major increase in effort.

Thus, it is probable that the number of DSLs reported is too low; a follow-up investigation with a more fine-grained detection mechanism seems worthwhile, although — due to the effort involved — it will probably have to focus on individual languages instead of providing a generic overview.

As shown in the results section, 3/4th of a project's code on average is written in GPLs, although the mean number of GPLs and DSLs is the same (and the spread is higher in the number of DSLs). This indicates that DSLs are indeed used

as their name implies — that is, for *specialized* purposes with *less code* required than for the main program logic written in the respective GPL.

To sum up RQ1, we conclude that with the mean number of languages being 5 — with 2 GPLs and 3 DSLs — we can empirically support the expectation of the utilization of multiple languages per project; however, the number is not as high as might have been expected, which may be due to the reasons listed above.

4.1.2 RQ2

Our second research question is *Does the number of languages depend on one of the following other project properties: size, main language, age, number of commits, and number of contributors?*

This question is answered by our regression model. We first discuss the result regarding size, age, number of commits, and contributors: Increasing the size of the main language (bytes written in that language) as well as increasing the number of commits — while keeping all other variables constant — significantly increases the number of languages found. On the other hand, changing the age and the number of contributors does not have a significant effect on the number of languages used.

An increasing number of languages with an increase of size seems to make sense. Small projects may not have need of certain DSLs, for example dedicated build scripts or shell scripts for startup or maintenance. The need for these will become more pronounced as the size of the project grows. A similar argument can be applied to the number of commits: Although we disregard commit size, it can be expected that most commits *add* to the project, which obviously will include additions in languages.

On the other hand, the missing effect of the variables age and number of contributors on the number of languages comes as a bit of a surprise. One could intuitively have expected older projects (i.e. projects with an earlier creation date) to have a combination of legacy and new languages, and a greater number of contributors to lead to an increase in the number of languages since there are different languages an individual developer may be more proficient in.

There may be several reasons for this lack of effect. First, the age of projects on GitHub is a difficult issue in general. GitHub was founded in 2008, which means that the oldest projects *by date of creation on GitHub* are 6 years old. However, some projects have been imported with their commit history, reaching back to the seventies in some cases, as indicated by the *date of first commit* which we have used as the project age. Unfortunately, it is unknown for how many projects this was done, i.e. how many of the projects with their first commit *after* 2008 have been truly *created* at that date — or whether they have simply been migrated to GitHub without history.

Thus, the age of some projects is probably effectively much higher than reported. Repeating this analysis using other repositories (such as SourceForge) may indeed yield different effects. As it is, the age of a project (i.e., months since the first commit) does not seem to have any bearing on the number of languages used. Taking a different angle, one could also take the view that the number of commits is a sort of *working age* of the project — which may indeed be more relevant than the calendar age.

The amount of contributors to a project has a range from 1 to 462, although the data of projects with a contributor count of higher than 100 is very thin (38 projects). A possible explanation for the non-influence of contributors on languages is that GitHub encourages using forks and reintegration of code into the original repository, which will add many contributors with very small contributions each. Thus, the number of contributors on GitHub does probably not relate to larger development teams, but to an added number of bug fixers. However, it may also be the case that the addition of new languages is more of a team decision and thus indeed more contributors do not automatically mean more languages. As it is, we do not see a significant association between the number of contributors and the number of languages in our data set.

We finally come to the question of the main general-purpose language of a project and its effect on the number of languages. As discussed, we use the Java programming language as a reference. We first assert that there is no significant difference in the number of languages between Java projects and projects with other well-known and much-used GPLs, namely C++, PHP, Objective-C, C#, Perl, Python, and Scala. Thus, changing the main programming language of a project to another in this group doesn't have much bearing on the number of languages used in each project. This result is probably expected since each of these languages has the same primary purpose (and in particular, does not include the particular functionality of most DSLs).

However, there are some programming languages with a significantly different number of languages than Java projects. Firstly, JavaScript, CoffeeScript, and Ruby show a significant deviation with about 44% to 56% more languages used. We believe that this is due to these languages serving the web domain, that is, they will always use HTML and CSS, possibly a template language, *plus* all of the languages the other projects use; it is however interesting that this does not seem to apply to PHP, the cause of which is unknown. TypeScript is an exception: Six of the seven projects with this main language are implementations of BitCoin. Thus, the number of languages here is probably more related to BitCoin than to TypeScript.

There are two additional languages that show a significant deviation from the reference category. First, projects using the C language use about 33% more languages than Java; a fact we attribute to the low-level nature of C, requiring more helper languages. The second is Groovy, which is the exact opposite, being a very new, dynamic language built on top of the Java platform and uses twice as many languages as Java, for which there is no obvious reason. However, there are only six projects with this main language; thus, we should be careful in reading too much into this result.

We can conclude that most of the main programming languages used (C++, Java, C#, Python, Objective-C, PHP, Scala, Perl) do not exhibit a significant difference in the number of languages used. However, there is a significant and relevant effect in web-related languages (JavaScript, Coffeescript, and Ruby) as well as in the C language, where the number of languages is increased by about 33% to 56%.

4.1.3 RQ3

Our last research question is *Which association patterns can be found between the languages (and in particular, GPLs and DSLs) used in the projects?*

This question is related to the *actual* languages, not the

number of languages, found in the projects. As discussed in the previous section, our primary result is shown in Figure 3.

When interpreting this figure, we need to keep in mind that the sizes of the boxes depend on the number of projects in the underlying set, which are GitHub-dependent; the arrow sizes are relative associations between the projects, which are project-dependent.

The figure can be divided into three parts — languages grouped around Shell and Make (top right), XML (left), and HTML/CSS (bottom). When looking at the GPLs, most are indeed grouped to one or at most two of these groups; none covers all three. We now look at each of these ecosystems in more detail.

The first ecosystem is grouped around Shell (which includes all Unix-based shells) and Make. Here, we find the languages C and C++ (for Make and Shell) as well as Python and Perl (for Shell). Objective-C is added to the area via C and C++ to which it is associated. C# is conspicuously missing here, which is probably due to it being part of the Microsoft .NET platform and thus a different setup and community. The languages in this area also show the highest amounts of GPL-to-GPL associations: Objective-C occurs with both C and C++, and C++ and C are associated with one another. Note also that all C-based languages have an association with XML as well. Thus, given C or C++, we can expect to see both the other language, respectively, as well as make files, shell scripts, and (perhaps not as intuitively) XML.

The second ecosystem is grouped around XML, which is the most-associated language in the set with 7 of the 10 languages referencing it directly (all except Perl, Python, and Ruby). The strongest associations come from C#, Java, and Objective-C with 90%, 83%, and 80% of the respective projects also featuring XML. In fact, C# references nothing else, while Java only additionally references the INI format (which includes many line-based formats such as .properties, .ini, .cfg, etc.). As we have discussed before, the XML format is problematic (and probably referenced as much) due to its flexibility and many use cases. In fact, we can expect different XML *dialects* to be used; in particular, the functionality of Make is probably replicated in the Java ecosystem using Maven or Ant scripts, which are based on XML.

This ecosystem seems to confirm the anecdotal association of Java and C# with XML; however, due to the generic nature of XML any further insights would require, again, XML dialect recognition. Note also that C# and Java are not associated with any other GPL (in contrast to C, C++, and Objective-C on the upper right).

Finally, the third part of the figure is grouped around HTML and (to a lesser extent) CSS. The JavaScript language shows the strongest associations to these two languages with 71% and 76% of JavaScript projects using these languages. PHP projects use CSS in 64% of cases.

Besides JavaScript, we have associations to HTML from Ruby, PHP, Python, and Perl. PHP is the only language in the lower ecosystem associated to another GPL — in this case JavaScript, and it is to be expected that this sort of relationship is very different compared to the association of C++ and Objective-C to C in the ecosystem at the top. Regarding the Ruby language, it can be assumed that YAML is used to take the place of XML.

Our three ecosystems also use three different DSL *types*: The first (Shell, Make) shows languages related to the process of building the software and generic file-oriented tasks; the second (XML) shows the use of a very generic language for structured data, while the third (HTML and CSS) is concerned with user interface tasks.

Thus, the answer to our third research question shows clear associations between languages, and these differ markedly between the three ecosystems shown, both in GPL-to-GPL associations as well as GPL-to-DSL associations.

4.2 Comparison with Related Work

To our knowledge, this is the first empirical study that discusses the association between different properties of software projects and the number of languages (i.e., RQ2), and the first study that uses the combined random/theoretical sampling approach. However, there are studies with a partial overlap to our answers to RQ1 and RQ3 with different sampling approaches, which we compare in the following.

In 2014 [5], Tomassetti and Torchiano have inspected a set of 15'000 *randomly* selected GitHub projects with the aim of investigating multi-language applications (using the term polyglotism). Their findings yield a mean of 6 languages per project, of which 2 are GPLs and 4 DSLs, which is close to our result (which is 5, 2, and 3, respectively). We suspect that the additional language reported may be a *prose* language (such as MarkDown) which they include and we exclude. Furthermore, they investigate pairs of interacting languages, which is similar to what we report as results for RQ3. Here, the random vs. theoretical selection of projects (and in particular our criteria of differing main languages) seems to be a major factor, since the languages and relationships reported are quite different (for example, C# and Perl are missing and C/C++ are not connected with the rest of the graph; instead, several web-related languages show up which we suspect is due to the large percentage of web projects on GitHub). Still, two of our clusters can be found in these results as well (XML as well as HTML/CSS).

Deloray et al. [6] (2007) analyzed 9'997 open source projects from SourceForge.net[7]. They list the most popular languages found in these projects, which are very similar to what we have found (if we replace Ruby and Objective C from our list with Pascal and Tcl from theirs, we get the same top 10 — this might be due to the publication date difference). Furthermore, they investigate combinations of (GPL) programming languages they call "language profiles" from the perspective of the author, i.e., how commonly languages are used together by an author compared to a standalone use. The top combinations are C/Perl, C/C++, and Javascript/PHP; two of these (C/C++ and Javascript/PHP) also show up in our association rules. Thus, we can (partially) confirm their findings on our (smaller, but differently sampled) GitHub-based data set.

In another study of 22 open source projects, Karus and Gall [7] (2011) focused on language usage evolution. They explicitly list DSLs. The analysis is based on revision data in version control systems, i.e. file types in developer commits. Thus, instead of being focused on projects as in our case, an author perspective is taken. It was found that developers work with 4 different languages on average (including DSLs) which is close to our mean number of languages. Also, the analysis of common (programming language) file types in commits shows several co-changing languages. Intersections with our association rules are Java/XML, C/Make,

[7] `http://www.sourceforge.net/`

and JavaScript/CSS. A direct comparison with our data is difficult due to the different focus (projects vs. authors) and the different input set (1150 projects without version history in our case; 22 projects with full version history in theirs).

A different take on multiple programming languages is presented by Vasilescu et al. [8] (2013). They investigate pairs of languages in the sense that developers have knowledge about both of them based on language tags associated to users on StackOverflow[8]. It is noted that they expect similarity by usage to imply similarity by knowledge; and in fact, the presence and confidence of the association rules shown in Figure 3 seem to be reflected, at least by tendency, in the *mutual intelligibility measures* by Vasilescu et al.

4.3 Limitations of our analysis

In general, it should again be noted that our project selection focused on *diversity*, which means that the results of RQ1 and RQ3 are not *representative* of GitHub. Here, we follow the line of argument brought forward by Naggapan et al. [2]: By selecting projects with a maximally *diverse* set of properties, we include as many shapes and forms of projects as possible (which are all actually existing open source projects), which in our opinion is actually *more* useful than proportional coverage. In any case, GitHub itself has its own biases — towards JavaScript as a main language as well as small ("Hello World") projects.

Second, we only report languages that are detectable by the *Linguist* tool. Since this tool is in industrial use and is continually being worked on, we believe that the number of undetected relevant languages to be low. However, Linguist does not inspect the code *within* a file, and thus will miss multiple languages within one file or dialects of languages such as XML. As discussed before, this means that the actual number of languages might be higher than reported.

Finally, two of the covariates we have used in the regression model have important limitations. The first is the age of the project, which as discussed is problematic due to the founding date of GitHub and the question of migration with or without history. We have found no association between the age and the number of languages; this may be different if a repository provider with a longer history is used. The second is the number of contributors, which — due to the way GitHub works — includes users that have simply provided bug fixes or small changes. Contributors thus do not equal full team members, and again the missing correlation might be attributed due to this fact.

5. CONCLUSION AND OUTLOOK

This work has examined 1150 open source software systems selected for diversity regarding language usage. Our analysis has shown that typical projects use a mean of 5 languages with a clearly dominant main GPL. The main DSL categories used are *Structured Data*, *User Interface Description*, *Shell*, *Build*, and *Configuration*. The average number of languages in each DSL category per project is 1 except for UI (where it is 2). Also, on average, only about 1/4th of the code of a project is written in a DSL.

Second, we have shown that there is a significant association between the size and the number of commits in a project and the number of languages; and no significant influence of age of the project and the number of contributors. Furthermore, the number of languages used does not seem to vary significantly between major languages — except for C and the languages used in the web domain (JavaScript, CoffeeScript, and Ruby).

Finally, we have identified three language ecosystems within the top 10 GPLs using association rule mining. These systems seem to be grouped around a) Shell and Make; b) XML, and c) HTML and CSS.

We believe that these answers can help us focus both practical efforts and further research. Regarding practical applications, it seems beneficial to provide developer tool support for much-used DSL *types* (RQ1) and the concrete DSLs identified as part of RQ3; having IDE support for multi-language systems seems important, especially for larger systems as indicated by RQ2. Our work also the highlights key multi-language combinations that developers (and students) should be aware of today.

There are several avenues for future research. First, it will be interesting to inspect generic DSLs such as XML for the *dialects* used. Secondly, we have seen that the variables of age and number of contributors of a project suffer from issues on GitHub; other repositories should be investigated here. Finally, project-specific and internal DSLs are not supported by current language detection tools; here, a qualitative approach to investigating example projects may yield interesting results.

[8] `www.stackoverflow.com`

Acknowledgments

The authors would like to thank Helmut Küchenhoff of the Statistical Consulting Unit of LMU for his support in performing the statistical analysis for this research.

6. REFERENCES

[1] P. Mayer and A. Schroeder, "Automated Multi-Language Artifact Binding and Rename Refactoring between Java and DSLs used by Java Frameworks," in *ECOOP 2014*. Springer, 2014, pp. 1–26.

[2] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *ESEC/FSE 2013*. ACM, 2013, pp. 466–476.

[3] Ludwig Fahrmeier and Thomas Kneib and Stefan Lang and Brian Marx, *Regression: Models, Methods and Application*. Springer, 2013.

[4] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *VLDB 1994*. Morgan Kaufmann, 1994, pp. 487–499.

[5] F. Tomassetti and M. Torchiano, "An Empirical Assessment of Polyglot-ism in GitHub," in *EASE 2014*. ACM, 2014, pp. 1–4.

[6] D. P. Delorey, C. D. Knutson, and C. Giraud-Carrier, "Programming language trends in open source development: An evaluation using data from all production phase sourceforge projects," in *WoPDaSD 2007*. Springer, 2007, pp. 1–5.

[7] S. Karus and H. Gall, "A study of language usage evolution in open source software," in *MSR 2011*. IEEE, 2011, pp. 13–22.

[8] B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "The Babel of Software Development: Linguistic Diversity in Open Source," in *SocInfo 2013*. Springer, 2013, pp. 391–404.