The Influence of Technical Variety in Software Ecosystems

Tom Peirs, Freark Westra, Sabine Molenaar, Slinger Jansen
{t.h.peirs,f.f.westra}@students.uu.nl,{s.molenaar,slinger.jansen}@uu.nl
Utrecht University
Utrecht, Netherlands

ABSTRACT

There is a lack of empirical evidence on software ecosystem health metrics, and a need for operationalizable metrics that describe software ecosystem characteristics. This study unveils a new approach for measuring technical variety concisely. Studies show that a high variety opens up new opportunities and thus, better niche creation, and ultimately, improves software ecosystem health. Four different ecosystems are evaluated, and compared. Variety is measured in relation to robustness, and productivity metrics of the ecosystem to uncover the influence of technical variety on software ecosystems. Technical variety indicates a positive correlation with robustness, however acceptance of this statement is not confirmed with certainty due to a weak relation. Furthermore, significant relations indicate differences between ecosystem types.

CCS CONCEPTS

• **Software and its engineering** → *Software system structures.*

KEYWORDS

Software Ecosystems, Software Ecosystem Health, Technical Variety, Niche Creation.

ACM Reference Format:
Tom Peirs, Freark Westra, Sabine Molenaar, Slinger Jansen. 2020. The Influence of Technical Variety in Software Ecosystems. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3387940.3392207

1 INTRODUCTION

For developers, professionals, entrepreneurs, architects, and stakeholders it is a crucial strategic decision in what programming language their software will be developed. A programming language can be considered as an ecosystem with actors that interact with each other by exchanging libraries, repositories, code chunks, and other information. Considering programming as such opens up evaluating programming languages from an ecosystemic perspective. Jansen proposed a framework for evaluating health of software ecosystems: Open Source Ecosystem Health Operationalization (OSEHO) [11]. Health of a software ecosystem is *the longevity and a propensity for growth* [15]. Jansen presents a number of metrics divided into the three pillars: Robustness, Productivity, and Niche Creation, based on Iansiti's et al's and den Hartig et al's works on business ecosystems [10, 12].

In 2010, Dhungana et al. made a comparison, between biological ecosystems and software ecosystems. They state that for software ecosystems, similarly to biological ecosystems, diversity is important for its sustainability [4]. For software ecosystems, it is considered that diversity is beneficial for the ecosystem [11]. In software ecosystem research, diversity is expressed as variety, hence why the term *technical variety* is used hereafter.

Technical variety is interpreted as *"the degree to which projects support different technologies"* [15]. Higher technical variety opens up new opportunities and thus, better niche creation, and ultimately, improved software ecosystem health [11]. A programming language that can be applied to a high variety of different applications becomes a valuable asset.

OSEHO solely mentions metrics and methods for gathering data for that metric on a high-level. Concise units and methods for gathering that data are, mostly, still lacking. Jansen calls for studies on data-gathering to showcase how data can be gathered. Currently, there is a lack of empirical evidence on health inducing characteristics of software ecosystems [9]. There is a need for a set of operationalizable metrics that describe software ecosystem characteristics.

This research proposes a new approach for measuring technical variety with a concise method on how data can be gathered. Herewith, developers can make better-informed decisions on what software ecosystem to invest in, and software vendors can govern their software ecosystem better assuming a large correlation between robustness and productivity is found. Technical variety is compared with metrics for productivity and robustness to validate the influence of technical variety on software ecosystem health. The number of active contributors is considered to be the most important indicator of software ecosystem health [11, 13]. The number of contributions indicates the productivity of the software ecosystem [11]. When assuming OSEHO's principles that imply the three pillars are associated, a high technical variety will lead to improved software ecosystem health.

Firstly, an introduction to the research method to uncover the relation between the conceptual, operational, and quantitative level of the research is made. Where Section 3 analyzes empirical research on software ecosystem health. Section 4 identifies variables on how the method is reacting to technical variety, robustness, and productivity in addressing the impact of technical variety. Results, addressed in section 5 show a positive association with robustness, rejected because of poor relationship. New significant relationships suggest differences of ecosystem types. Section 6 summarizes the proposed approach, defends its novelty, and provides guidance for future studies. Finally, section 7 draws conclusions on the proposed method, defends its novelty, and provides innovative approaches for future research in open source software ecosystems.

2 RESEARCH METHOD

The goal of the research is to uncover the relation between technical variety and software ecosystem health. Two research questions are operationalized. An overview of the research is described via a Goal-Question-Metric diagram in Figure 1.

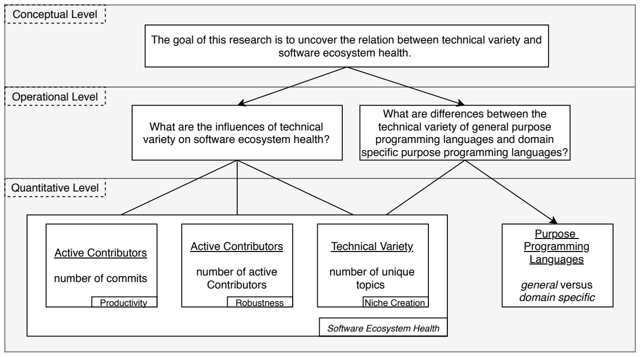
Conceptual Level
The goal of this research is to uncover the relation between technical variety and software ecosystem health.
Operational Level
What are the influences of technical variety on software ecosystem health?
What are differences between the technical variety of general purpose programming languages and domain specific purpose programming languages?
Quantitative Level
Active Contributors
number of commits
Productivity
Active Contributors
number of active Contributors
Robustness
Technical Variety
number of unique topics
Niche Creation
Software Ecosystem Health
Purpose Programming Languages
general versus domain specific

Figure 1: Goal, question and metric model in the context of the study.

To uncover a relation between technical variety and software ecosystem health, technical variety is compared with two OSEHO metrics. One of the metrics belongs to the pillar robustness, the other one to productivity.
RQ 1: *What are the influences of technical variety on software ecosystem health?*
As a higher Technical Variety implies better opportunities for the emergence of niches, it should have a positive influence on software ecosystem health.

Two hypotheses are extracted from this research question. The first hypothesis aims to verify a relationship between robustness and technical variety. The second hypothesis focuses on a relationship between productivity and technical variety.

- **Hypothesis 1:** *Technical variety within a software ecosystem, has a positive effect on the robustness of a software ecosystem.*
- **Hypothesis 2:** *Technical variety within a software ecosystem, has a positive effect on the productivity of a software ecosystem.*

The second research question aims to uncover significant differences in technical variety between programming languages.
RQ 2: *What are the differences between the technical variety of general-purpose programming languages and domain-specific programming languages?*
From the second research question, one hypothesis is extracted. The technical variety for both general-purpose and domain-specific programming languages is measured to uncover significant differences. Assumingly, general-purpose programming languages know a wider array of applications.

- **Hypothesis 3:** *general-purpose programming languages, have a significantly higher technical variety within a software ecosystem, opposed to domain-specific programming languages.*

3 LITERATURE STUDY

The most cited definition for a software ecosystem is built upon prior definitions of software ecosystems and consists out of three core elements: actors, organizations and businesses; networks & social or business ecosystems; and software [12]. Through partnerships, software companies increasingly open up their software ecosystems to external parties.

The emerging community of external parties on a software ecosystem generally consist of keystones, niche players, value-added re-sellers and customers [8]. Lucassen et al. state software ecosystem health entails "*the longevity and a propensity for growth*" [15]. More commonly, an ecosystem is considered healthy when all actors' are satisfied [16]. The three pillars, *productivity, robustness and niche creation* function as the foundation for OSEHO have their origins in business strategy literature [12]. Iansiti et al. propose the success of a business strategy is dependent on external factors, understanding the ecosystem is a prerequisite for designing and deploying successful business strategies [10]. *Productivity* concerns the products and services that are created by the ecosystem versus the input that is put into the software ecosystem. The *robustness* of a software ecosystem explains to what degree the ecosystem can endure interfering factors. *Niche creation* explains the niche opportunities emerging from the software ecosystem. Important to note is that these niche opportunities are not always opportunities for financial gain, the opportunities can also be different. Any metric in one of the three silos of OSEHO is inherently related to other metrics within that same silo because they describe the same concept. Cross-silo relations, on the other hand, are theoretically unrelated.

Jansen's OSEHO framework is the most cited software ecosystem health operationalization [11], other initiatives that capture characteristics of software ecosystems exist. Franco-Bedoya et al. introduced a Quality model for open source software ecosystems (QuESo) [5]. A model developed by gathering user-friendly and operationalizable quality metrics from literature. The model is set-up top-down with three dimensions: platform-related characteristics, community-related characteristics and ecosystem network characteristics. In total, five characteristics, 14 sub-characteristics, and 68 metrics are divided over the three dimensions. Compared to OSEHO, QuESo emphasizes more on the potential for operationalization of metrics and it provides a more complete set of metrics. On the other hand, it lies less emphasis on relating actual software ecosystems or literature to the model. Hereby, QueSo can be considered as a comprehensive collection and categorization of metrics without a theoretical explanation. Several measures for variety are mentioned, technical variety, however, is not represented in QuESo. Instead, they state the variety of products offered by partners in the ecosystem. A measure not operationalizable to open source programming languages due to the number of written repositories, the scatter of data sources and the constraint that not all repositories are publicly available. Except for measures stated in OSEHO and QuESo [12, 15, 20], no literature was found that mentions any new metrics for technical variety.

The variety between programming languages is expected to differ between domain-specific programming languages and general-purpose programming languages. Whereas, domain-specific programming languages are specialized in a particular domain. general-purpose programming languages can be utilized in a plethora of domains. The limited scope of domain-specific programming languages allows for better learnability and thus enables domain experts unrelated to the computing science field to comprehend the language [1]. The extended comprehension allows domain experts to co-develop so the application domain is better enabled to leverage computing power [14]. Some disadvantages of a smaller scope are its limited applicability [14],and a loss of processor efficiency [14].

4 RESEARCH EXECUTION

To capture technical variety, productivity, and robustness, GitHub[1] repositories are scraped. GitHub, a popular online tool for software development and version control offers publicly accessible APIs. Researchers have used GitHub as a valuable resource for both quantitative [6, 21, 22] and qualitative [2, 3, 17, 18] data. The tool has been gaining popularity under developers both as a collaborative platform as an alternative repository for software. Recently, GitHub reached the milestone of having 100 million repositories[2]. On January tenth, 2020, GitHub held 17.30% market share in the source code management domain[3].

Technical variety is interpreted as the degree to which projects support different technologies [15]. To quantify technical variety, the number of unique technologies that are mentioned in the descriptions of repositories in that programming language ecosystem is taken, see Table 1, and filtered trough a unique glossary of topics. The unique glossary of technical topics is acquired by retrieving the 1000 most used topics (in GitHub) for each of the 20 most popular programming languages according to the Tiobe index. See in Table 2 for the Tiobe index. This results in a glossary of 10284 unique technical topics, after duplicates are removed.

For hypothesis 1 and 2, technical variety is the independent variable. Based on that technical variety, the relations with robustness and productivity are explored. *Robustness* in hypothesis 1 is expressed as *active contributors*. It is quantified as the number of developers that made commits in a repository in 2019, see Table 1.For hypothesis 2,*Productivity* of a programming language ecosystem is expressed as the *Number of Commits*. Commits indicate the number of code changes that were published in the repository. Some programming languages have been used for a longer period. To make the number of commits comparable between programming language ecosystems, only the commits for 2019 are compared, see Table 1. It could be argued that lines of code is a better metric to evaluate productivity, however, as different programming languages use a different syntax, the value of lines of code might vary. Which can result in a skewed description of productivity efforts. In hypothesis 3, the programming language ecosystems under investigation are categorized according to their *purpose*; either general-purpose or domain-specific, see Table 1. For comparison, programming languages are grouped in these two categories.

[1]https://www.github.com
[2]https://github.blog/2018-11-08-100m-repos/
[3]https://www.datanyze.com/market-share/source code-management/github-market-share

Table 1: Overview of variables per hypothesis.

Hypothesis	Independent	Dependent	Metric	Type
1	Technical Variety		# Unique topics per repo	Interval
		Robustness	# Contributors in 2019 per repo	Interval
2	Technical Variety		# Unique topics per repo	Interval
		Productivity	# Commits in 2019	Interval
3		Technical Variety	# Unique topics per repo	Interval
	Language Type		General-purpose r Domain-specific	Nominal

4.1 Selection of Programming Language Ecosystems

The 20 most popular programming languages, according to the tiobe index of January 2020 [4], are used to make a preliminary selection of programming language ecosystems under investigation. Then, it is determined whether the languages are domain-specific or general-purpose. A programming language is considered to be domain-specific when it matches van Deursen et al.'s definition: *"A domain-specific language is a small, usually declarative, language that offers expressive power focused on a particular problem domain."* If a language does not match that description it is considered general-purpose, in the context of the research. See Table 2 for an overview of all languages.

Out of these 20 programming languages, four have been selected for investigation. Because domain-specific programming languages naturally rule out a large group of potential users due to its limited scope, domain-specific programming languages are less popular, as seen in Table 2. Expectedly, differences in technical variety are the biggest between the least popular and the most popular programming languages and general-purpose programming languages versus domain-specific programming languages. Therefore, the two most popular general-purpose programming languages and the two least popular domain-specific programming languages are selected. Respectively, Java & Python and R & Matlab. These programming languages are used for analysis. Note that Tiobe's second most popular language *C* is not selected because Java and C bare both multi-paradigm languages. Furthermore, Java is part of "the C family of languages". What this means, principally, is that Java is one of a number of languages that inherit C's straightforward, clean, and fairly elegant syntax. Due to these similarities we chose the next programming language to potentially, detect more diverse results.
R, Matlab, Java, and Python, can all be used for statistical computing, but are highly varying languages in terms of applicability. We have selected these languages based on convenience sampling and as future work aim to study a larger set of languages ranked by popularity. Python and Java are general-purpose languages with more diverse applications. They can be used in a variety of domains such as embedded systems, enterprise applications, web servers, etc. Python's popularity is at a steady increase, whilst Java's popularity has steadily been decreasing for over a decade, excluding a peak in 2015 [4].

R and Matlab are developed specifically for statistical computing and have a more narrow field of application. Popularity for both languages peaked in 2017. Compared to R, Matlab's rise of popularity has been more gradual [4]. We hypothesize that the two general-purpose programming languages will have higher technical variety than the two domain-specific programming languages.

[4]https://www.tiobe.com/tiobe-index/

Table 2: Programming language shortlist selection criteria

Tiobe Index Rank	Programming Language	general-purpose	Domain Specific
1	Java	✓	
2	C	✓	
3	Python	✓	
4	C++	✓	
5	C#	✓	
6	Visual Basic .NET	✓	
7	JavaScript	✓	
8	PHP	✓	
9	Swift	✓	
10	SQL		✓
11	Ruby	✓	
12	Delphi/Object Pascal	✓	
13	Objective-C	✓	
14	Go	✓	
15	Assembly language	✓	
16	Visual Basic	✓	
17	D	✓	
18	R		✓
19	Perl	✓	
20	MATLAB		✓

4.2 Data Collection

The data-gathering objective of this study is to acquire a set of packages that are being developed, with respect to popularity, for each programming language ecosystem in question, through GitHub. Repositories are the storage locations where software projects are stored and retrieved. For the programming languages Python, Java, R, and Matlab 1000 repositories are scraped, which results in a total of 4000 repositories. Data from these repositories is obtained via GitHub's search API. For the programming languages in question, the repositories are sorted by popularity, and the most popular repositories are scraped. Popularity is measured in terms of GitHub's stars [5]. The API is accessed through HTTPS via https://api.github.com.

- **Technical variety** is constructed through a glossary of topics. For each repository, matches between words in the description and the glossary of topics are sought. All referrals to the parent programming language name are removed, and multiple mentions of a topic are processed as one mention.
- The **Commits** are gathered per repository, through GitHub's search API.
- The number of **Active contributors** is seen as people who contribute to the code base of the repository. An active contributor is defined as a contributor who has made a commit to the repository in 2019. More specifically, the number of active contributors is the number of unique contributors in 2019. Due to GitHub's search API's rate limits, the number of active contributors is obtained with the web-scraper "Data Miner"[6]. On the contributor overview page of a repository, a range of dates can be set. Hereby, the contributors that were active in that period are then displayed. The date range can be specified in JQuery selectors. When the top 1000 repositories per language were gathered, the URLs of these repositories were also scraped. The web scraper evaluates the web page's content by evaluating JQuery selectors matching the selector.

[5]https://help.github.com/en/github/getting-started-with-github/saving-repositories-with-stars
[6]https://data-miner.io/

4.3 Data Preparation

To transform the raw data acquired in the previous step, minor data cleaning and construction of new attributes is performed. The following data transformations are executed:

(1) A new attribute was created, which transforms repository URLs, to a new list of URLs that can be used as input to the web-scraper.
(2) The number of active contributors included a '#'-prefix. This is simply removed.
(3) Empty values for active contributors are changed to a numeric value of zero.
(4) Duplicate values in the glossary of topics are removed.
(5) Natural language in descriptions is formatted for interpretation.
(6) Referrals to the parent language are removed.

However to interpret, and analyze *technical variety* a few transformations are performed. Five quick transformations need attention, data cleaning involves the following steps:

(1) Stripping any extra white space.
(2) Transforming everything to lowercase.
(3) Remove numbers.
(4) Remove punctuation, but keep intra-word-contractions, and preserve intra-word-dashes.
(5) Remove stop words.

No additional Data Preparation is necessary as GitHub API's return JSON data, which returns data in the form of attribute-value pairs. The necessary values are extracted through a JavaScript web application, written specifically for this research. The source code has been made publicly available[7].

4.4 Data-gathering Limitations

GitHub's API is designed to return results on a single repository, which limits research efforts in two ways: One being that a search limit of 1000 returns per call is maintained, hence why only the top 1000 repositories are retrieved. Secondly, to acquire unique active contributors in 2019, all commits of the last year need evaluation, where some repositories exceed 1000 commits. Hence, the number of active contributors for that repository could not be evaluated. To overcome the incompleteness of the data, a web-scraper that takes a user interface approach is used. The data-miner maintains a monthly rate limit of 500 URLs, which is not sufficient for gathering the active contributors of the top 1000 repositories per programming language. However, a subscription plan could be purchased to up this limit. Finally, the number of repositories that have specified topics is inconsistent across the programming languages. Removing repositories without topic specification would decrease the

[7]https://github.com/peirstom/githubScraper

number of repositories under investigation drastically, from 1000 repositories per language to 210 repositories per language.

4.5 Reproducibility

To allow researchers to reproduce the data-gathering steps, promote reuse and improve transparency, the source code for accessing GitHub's API is made publicly available on GitHub[7]. The source code contains elaborate setup instructions.

5 RESULTS

Section 5.1 analyzes the differences between general-purpose languages and domain-specific languages to gain a better understanding between them. Section 5.2, on technical variety aims to analyse the relation between technical variety, and the programming language types; general-purpose languages, and domain-specific languages. Section 5.3, on OSEHO aims to analyze if significant factors such as contributors, or the number of commits to a repository influence technical variety.

5.1 General-Purpose vs Domain-Specific Languages

The mentioned topics explain a programming language's DNA. Frequently mentioned topics indicate there has been more productivity related to these topics. Thus, the most popular topics indicate the main application of that programming language ecosystem. The most popular topics are identified by checking whether there are topics with significantly more occurrences than other topics.

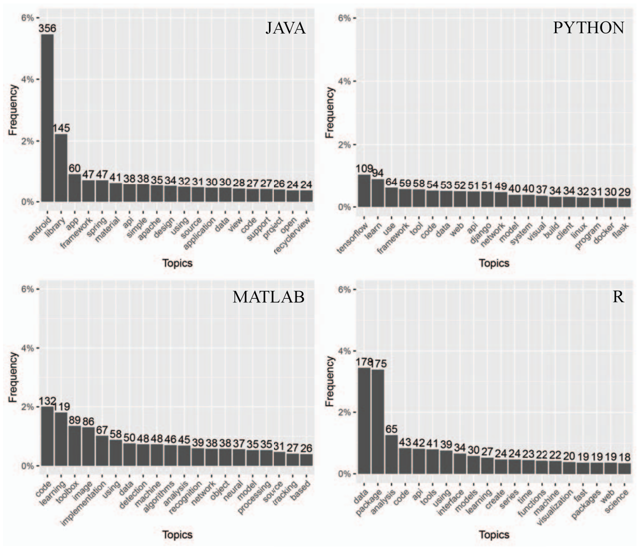

Figure 2: Top 20 topics per programming language with their relative frequencies.

General-Purpose Languages; (Java & Python)

When looking at technical variety data for Java, it becomes apparent that *android* is, by far, the most used topic, as can be seen in Figure 2. *Android* is mentioned 356 times in total, which can be explained by the fact that *android* is written in Java. The second most mentioned topic is a *library* with 145 mentions. Java is set up with a code management system that offers users easily accessible code. In that system, code chunks are managed in libraries. The remainder of mentions has 60 or fewer mentions, see Figure 2.

For Python, the most popular topics know less distance compared to the remainder of topics. The topics *learning* and *library* are most popular, respectively 83 and 72 mentions. The remaining topics are mentioned 48 times or less as can be seen in Figure 2. *Learning* might refer to machine learning, because in the glossary of topics *learning* is denominated as an isolated term. Intuitively, this is the case because Python is widely used for machine learning, this is not verified though. *Library* refers to Python's code management system. Whilst both programming languages are denominated as general-purpose programming languages, the application of Java appears more one-sided due to Android's popularity, compared to Python's. However, nothing can be said about the variety of the remainder of applications. As can be seen when looking at the technical variety of Java and Python, respectively 738 and 698.

Domain-Specific Programming Languages (Matlab & R)

The most popular terms in R are *data* with 178 mentions and *package* with 175 mentions. *Data* relates to R's purpose in statistical computing, a data-hungry discipline. *Package* refers to the *code* organization system, similar to Java's and Python's library system. The most popular for Matlab's terms are *code* with 132 mentions and *learning* with 120 mentions see Figure 2. A fundamental concept of programming languages and therefore caries no significant meaning. Similar to Python, *learning* might refer to machine learning. Matlab's *code* management system groups *code* into toolboxes, the third most popular term of languages with 89 occurrences.

5.2 Technical Variety Interpretation

To be able to compare the technical variety of different programming language ecosystems, the lengths of the descriptions need to be examined when interpreting results. The length of descriptions is expressed as the number of words per repository. Since the length of a description can imply a larger technical variety, there is no need to normalize the lengths. The mean is 9.25, which indicates that on average between nine and ten words are used to describe the repository. The standard deviation is 6.67. Graphs show a positive skew of 2.71 and a kurtosis of 12.41. All four languages had comparable descriptions in terms of word count.
To explain how elaborate descriptions per language are, the ratio of the number of words per number of unique topics mentioned per repository can be calculated. More words per unique topic indicate more elaboration on the topic. Java, and Python indicate to have a lower ratio with a value of 1.44, and 1.63 respectively. R, and Matlab show higher values with ratios from 1.73, and 1.79 respectively. For this data set domain-specific languages have more words per unique topic, which indicates a more elaborate description of the topic.
There is less resemblance in technical variety between the two domain-specific programming language ecosystems than for the two general-purpose programming language ecosystems. For the general-purpose programming language ecosystems, the difference between the two languages is only 40, with Java at 738 topics and Python at 698 topics. For Matlab there are 482 unique topics

mentioned, for R 616 unique topics are mentioned. Between the two languages there is a difference of 134 topics, 335% than the difference between the general-purpose programming language ecosystems.

The average number of topics per repository amongst all programming languages is 5.18, which implies that on average each repository mentions between 5 and 6 technical topics. The average number of topics for Java (μ = 5.86), and Python (μ = 5.43) is higher than for Matlab (μ = 5.00), and R (4.46). With Java, and Python having a higher average in topics per repository opposed to Matlab, and R, the indication is that general-purpose languages have a higher technical variety then domain-specific languages. This indication is concluded by performing a Welch two-sample T-test.

A Welch two-sample T-test proves a statistically significant difference between general-purpose languages and domain-specific languages. The technical variety per repository is significantly higher for general-purpose languages with t = 7.4987 and p-value = 7.973e-14. Hereby hypothesis 3 can statistically be accepted. Yet, the hypothesis is rejected because of the large difference between Matlab's and R's technical variety. Another way of looking at variety is by counting the number of topics that make up the top 25% of all mentions for that programming language. For R and Matlab nine topics make up the top 25% of all mentions. For Java, there are 13 topics and for Python, there are 23 topics. A Welch two-sample T-test again shows statistically significant results with t = 1.7509 and p-value = 0.08745.

5.3 OSEHO Metrics Interpretation

Has technical variety within a software ecosystem a positive effect on the robustness or the productivity of a software ecosystem?

A Pearson product correlation coefficient was computed to assess the relationship between technical variety and the number of commits to a repository. The results indicated that there was no correlation between the two variables, r = 0.019, n = 2, p = 0.221. Hereby, hypothesis 1 is rejected as it indicates no association between technical variety and the number of commits to the repository. The correlation factor of 0.019 does indicate a slight positive relation, the p-value of 0.221 does indicate there is no certainty of the relation. Hence why the correlation is denied.

A second Pearson product correlation coefficient was computed to assess the relationship between technical variety and the number of active contributors. The results indicated that there was a positive correlation between the two variables, r = 0.075, n = 2, p = 2.3e-06. Hereby, hypothesis 2 is accepted as it indicates a positive association between technical variety and the number of active contributors. A scatter-plot as seen in Figure 3 summarizes the results. The positive relation is extremely small, hence we cannot easily accept the hypothesis. As can be seen in Figure 3, the higher the technical variety, the more uncertain the relation becomes. Overall, there was a light, positive correlation between technical variety and active contributors. Increases in technical variety were correlated with increases in active contributors to the repository.

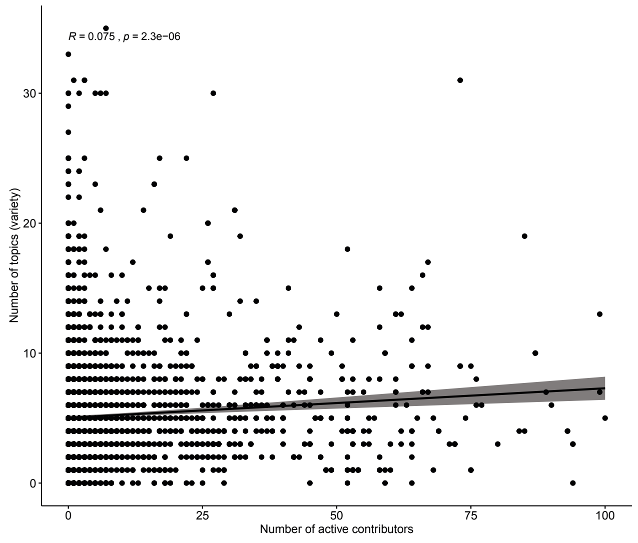

Figure 3: The number of active contributors is positively correlated to the number of topics, but the relationship is weak, as contributors often specialize in fewer topics.

5.4 Variety versus Specificity

There are many ways of comparing the technical variety within the programming language ecosystems. Three methods have been applied in this research.

Firstly, by looking at the number of topics, a statistically significant difference between general-purpose programming languages and domain-specific languages was found. The difference is deemed invalid due to the large difference between R and Matlab.

Secondly, because the median and quartile 1 are identical for all languages, quartile ranges can indicate technical variety. With this mindset, R and Java relatively have the highest variety as these languages have a larger share of topics that are mentioned just once. No distinction can be made between general-purpose programming languages and domain-specific programming languages. However, it is questionable if having a large number of topics with very few mentions implies technical variety.

Finally, the third procedure showed a statistically significant difference between the number of topics that comprised the top 25% of all topic mentions of general-purpose programming languages versus domain-specific programming languages. Similar to domain-specific programming languages in the first method, a large difference between general-purpose languages exist, Java with 13 topics and Python with 23. Additionally, the number highly influenced by the occurrence of relatively extremely popular terms such as Android for Java.

Many high-level terms appear in software ecosystems, and thus it is plausible a new niche opportunity related to that concept can arise. Therefore, this type of variety is called *regular variety* which describes the convenience a software ecosystem offers to seek for business opportunities in established niches.

The second form of variety rather explains the *specificity* of a software ecosystem. In programming languages with higher specificity, typically, have a larger amount of unique topic mentions.

For example, the term *FFmpeg* is mentioned once for Python. FFmpeg is a software platform for processing video and audio files via a command-line tool. For R, the topic *tesserect* is stated. A four-dimensional geometric shape. Finding niche opportunities from these topics is far sought as they have to open up isolated niches because the topics are highly specific. Both specificity and the variety benefit from more topics but they do describe different concepts of a software ecosystem.

6 DISCUSSION AND FUTURE WORK

The goal of this research is to analyze the influences of technical variety on software ecosystem health. Furthermore differences between technical variety in general-purpose, and domain-specific programming languages are evaluated. However, results show no relation between technical variety and productivity and robustness.

The chosen metrics which quantify the hypothesis are deemed poor proxies for this study. Where Number of active contributors and number of commits seem to be a bare minimum number of metric to measure robustness and productivity. Evaluating metrics such as "Bug fix time" or "Usage" could be additional metrics to evaluate productivity. Furthermore "Contributor ratings and reputation" could be an additional metric to evaluate Robustness. This study evaluated a limited number of metrics as these were available through GitHub in the scope of this project.

Another threat to this study is that the findings are hard to generalize. Since general-purpose languages can be used in more context, and hence it should have a higher technical variety. Second, some languages could be both general-purpose and domain-specific at the same time. For example, Python could be seen a a general-purpose Object Oriented language, however in the presence of machine learning (ML) frameworks it could be turned into a domain-specific ML language. Ultimately, the choice of sorting the programming language ecosystems based upon popularity, and general-purpose versus domain-specific could be questioned.

Technical variety is measured through text mining on descriptions of repositories. Improvements in linguistic techniques should be made for a more accurate analysis of the descriptions. For instance, certain words have more meaning together. A second problem reveals that the glossary might not include all correct topics or might be incomplete.

Linguistic techniques could improve retrieval of essential topics by analyzing repository descriptions. Linguistic is the scientific study of language and its structure [7]. It involves the meaning of words, which would benefit a more accurate list of topics for which technical variety would have a more accurate representation.

A second method to gain more accurate representations for technical variety is *topic mining*. Topic mining is a method for finding a group of words, which represents the information from a collection of documents as a topic [19]. However, as this exceeds current limits of this research a new approach which steers away from text-mining is taken. Topic-labels of repositories might give a better representation of variety within a repository opposed to descriptions, but might give similar results as topic mining. This approach is possible by evaluating topic-labels for the repositories in question. This study analyzes descriptions of repositories, as a normal distribution for word count on descriptions is present. Future studies could retrieve the most popular repositories, which do not lack the topic-label field, and work with topic-labels instead of descriptions. To evaluate this assumption, a new data set was retrieved by mining topics through labels apposed through descriptions, and revealed more interesting topics for all programming languages, which is shown in Table 3. The results indicate a weakness of this study can be overcome by analyzing topic-labels instead of descriptions. However, these results from the current data set cannot be taken seriously as a vast difference in the number of topic-labels for each repository is present. Table 3, indicates the frequency of topics for general-purpose programming languages, and domain-specific programming languages.

Table 3: Frequency of Topics for General-Purpose and Domain-Specific Programming Languages

(a) General-Purpose

Nr	Java Topics	Count	Python Topics	Count
1	Android	228	Machine-learning	94
2	Android-library	45	Deep-learning	89
3	Rxjava	28	Tensorflow	60
4	Animation	27	Pytorch	43
5	Spring-boot	24	Data-science	29
6	Material-design	20	Natural-language-processing	26
7	Library	20	computer-vision	26
8	Database	17	security	24
9	Elasticsearch	14	linux	23
10	Python	12	flask	20
11	SQL	12	Docker	20
12	HTTP	12	neural-network	20
13	Docker	11	django	17
14	Big-data	11	CLI	16
15	Microservices	10	HTTP	16

(b) Domain-Specific

Nr	Matlab Topics	Count	R Topics	Count
1	Deep-learning	35	Rstats	137
2	Machine-learning	24	ggplot	45
3	Computer-vision	24	Cran	44
4	Image-processing	14	Data-visualisation	42
5	Octave	11	Shiny	31
6	Caffe	11	Data-science	30
7	Robotics	11	Machine-learning	20
8	Convolutional-neural-networks	11	peer-reviewed	19
9	Cvpr	10	Rmarkdown	16
10	Dataset	10	Tidyverse	14
11	Super-resolution	7	Dplyr	12
12	Image-retrieval	7	Rstudio	9
13	Semantic-segmentation	7	API	9
14	Neuroscience	7	Data-analysis	8
15	Statistics	6	Text-mining	7

7 CONCLUSION

In this research, no relation between technical variety and productivity and robustness can be proven. The research does show there is a higher technical variety for general-purpose programming language ecosystems than there is for domain-specific programming language ecosystems. The interpretation of the retrieved measure for technical variety can, however, be done in multiple ways. More research is required to further specify the term variety in the context of software ecosystems. As the goal of the metrics is to describe the degree to which it opens up niche opportunities, empirical evidence is required to determine from what kind of varieties are beneficial for creating niche opportunities. Two forms of technical variety are uncovered, regular variety and specificity. The first one describes a large number of topics with considerable traction in the ecosystem. The second one describes a large number of unique topics that imply a higher degree of specification in the software ecosystem. The proposed method for gathering technical variety does open up opportunities for creating clusters through social network analysis in the form of correlation clusters in word clouds.

REFERENCES

[1] Ankica Barišić, Vasco Amaral, Miguel Goulão, and Bruno Barroca. 2011. Quality in use of domain-specific languages: a case study. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*. 65–72.

[2] Andrew Begel, Jan Bosch, and Margaret-Anne Storey. 2013. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Software* 30, 1 (2013), 52–66.

[3] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 1277–1286.

[4] Deepak Dhungana, Iris Groher, Elisabeth Schludermann, and Stefan Biffl. 2010. Software ecosystems vs. natural ecosystems: learning from the ingenious mind of

nature. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ACM, 96–102.

[5] Oscar Franco-Bedoya, David Ameller, Dolors Costal, and Xavier Franch. 2014. Queso a quality model for open source software ecosystems. In *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*. IEEE, 209–221.

[6] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*. 345–355.

[7] Michael AK Halliday. 2006. Jonathan Webster On Language and Linguistics.

[8] Geir Kjetil Hanssen and Tore Dybå. 2012. Theoretical foundations of software ecosystems.. In *Proceedings of the International Workshop on Software Ecosystems*. Citeseer, 6–17.

[9] Sami Hyrynsalmi, Marko Seppänen, Tiina Nokkala, Arho Suominen, and Antero Järvi. 2015. Wealthy, Healthy and/or Happy—What does 'ecosystem health'stand for?. In *International Conference of Software Business*. Springer, 272–287.

[10] Marco Iansiti and Roy Levien. 2004. Strategy as ecology. *Harvard business review* 82, 3 (2004), 68–81.

[11] Slinger Jansen. 2014. Measuring the Health of Open Source Software Ecosystems: Beyond the Scope of Project Health. *Information and Software Technology* 56 (11 2014).

[12] Slinger Jansen, Michael A Cusumano, and Sjaak Brinkkemper. 2013. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*. Vol. 1. Edward Elgar Publishing, Chapter Measuring the health of a business ecosystem, 221–245.

[13] Corey Jergensen, Anita Sarma, and Patrick Wagstrom. 2011. The onion patch: migration in open source ecosystems. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conf. on Foundations of software engineering*. ACM, 70–80.

[14] Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Varanda João Maria Pereira, Matej Črepinšek, Cruz Daniela Da, and Rangel Pedro Henriques. 2010. Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems* 7, 2 (2010), 247–264.

[15] Garm Lucassen, Kevin Van Rooij, and Slinger Jansen. 2013. Ecosystem health of cloud PaaS providers. In *International Conference of Software Business*. Springer, 183–194.

[16] Konstantinos Manikas and Klaus Marius Hansen. 2013. Reviewing the health of software ecosystems–a conceptual framework proposal. In *Proceedings of the 5th international workshop on software ecosystems (IWSECO)*. Citeseer, 33–44.

[17] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*. 117–128.

[18] Nora McDonald and Sean Goggins. 2013. Performance and participation in open source software on github. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. 139–144.

[19] Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*. 404–411.

[20] Walt Scacchi and Thomas A Alspaugh. 2012. Understanding the role of licenses and evolution in open architecture software ecosystems. *Journal of Systems and Software* 85, 7 (2012), 1479–1494.

[21] Ferdian Thung, Tegawende F Bissyande, David Lo, and Lingxiao Jiang. 2013. Network structure of social coding in github. In *2013 17th European conference on software maintenance and reengineering*. IEEE, 323–326.

[22] Jason T Tsay, Laura Dabbish, and James Herbsleb. 2012. Social media and success in open source projects. In *Proceedings of the ACM 2012 conference on computer supported cooperative work companion*. 223–226.