Representation of Developer Expertise in Open Source Software

Tapajit Dey
The University of Tennessee
Knoxville, TN, USA
Email: tdey2@vols.utk.edu

Andrey Karnauch
The University of Tennessee
Knoxville, TN, USA
Email: akarnauc@vols.utk.edu

Audris Mockus
The University of Tennessee
Knoxville, TN, USA
Email: audris@utk.edu

Abstract*—Background: Accurate representation of developer expertise has always been an important research problem. While a number of studies proposed novel methods of representing expertise within individual projects, these methods are difficult to apply at an ecosystem level. However, with the focus of software development shifting from monolithic to modular, a method of representing developers' expertise in the context of the entire OSS development becomes necessary when, for example, a project tries to find new maintainers and look for developers with relevant skills. Aim: We aim to address this knowledge gap by proposing and constructing the *Skill Space* where each API, developer, and project is represented and postulate how the topology of this space should reflect what developers know (and projects need). Method: we use the World of Code infrastructure to extract the complete set of APIs in the files changed by open source developers and, based on that data, employ Doc2Vec embeddings for vector representations of APIs, developers, and projects. We then evaluate if these embeddings reflect the postulated topology of the *Skill Space* by predicting what new APIs/projects developers use/join, and whether or not their pull requests get accepted. We also check how the developers' representations in the *Skill Space* align with their self-reported API expertise. Result: Our results suggest that the proposed embeddings in the *Skill Space* appear to satisfy the postulated topology and we hope that such representations may aid in the construction of signals that increase trust (and efficiency) of open source ecosystems at large and may aid investigations of other phenomena related to developer proficiency and learning.

***Index Terms*—Expertise, Developer Expertise, Vector Embedding, Doc2Vec, API, API embedding, Project embedding, Developer embedding, Skill Space, Machine Learning, Open Source, World of Code**

I. Introduction

The number of projects and developers involved with open source software has reached staggering heights, e.g. GitHub reported that over 10 million new developers joined and over 44 million new projects were created in 2019 alone [1]. While many of these developers or projects are based on individual effort, further statistics, such as over 87 million pull requests being merged and 20 million issues being closed in the past year on GitHub alone, demonstrate that open source development is a highly collaborative effort.

The key premise of open source software is not only to share the code, but, more importantly, to enable contributions from the community [1], [2]. However, despite improved tools and practices enabled by social coding platforms such as GitHub, it is not always easy to get contributions accepted, and, as many studies have shown, repeated interactions between the maintainers and contributors are necessary to establish trust and increase the chances of pull request acceptance or issue resolution [3], [4], [5], [6], [7], [8]. However, this method of building reputation and trust by repeated interactions does not scale very well, and, with a growing number of developers and an increasing number of projects their code may depend on [9], other means of establishing trust are becoming necessary. Previous work [10], [7] has shown that both *technical and social aspects of a developer's reputation* can play an important role in building the *trust* between themselves and other developers. While social aspects, such as previous collaboration [11], can greatly increase the trust between two developers, these aspects are not broadly applicable as they enhance trust within an already-established developer circle. For a developer looking to contribute for the first time to a project outside of their social circle, the technical aspect of their reputation, often referred to as *expertise*, may serve as an important *source of trust* for other developers when evaluating a developer as a potential team member or collaborator [12].

We, therefore, concentrate on gauging the *relevant* expertise of a developer based on their previous development activities. Such measure, if it could be obtained, might partially substitute for the traditionally laborious reputation building process as a developer transitions from a peripheral participation in a project to a contributor role [13], and could potentially increase the efficiency of the open source development as a whole. However, previous attempts at measuring developer expertise either focus on very detailed views, e.g. counting "experience atoms" associated with changes made by a developer on a specific source code file [14], or, at the least granular level, counting the volume, frequency, and breadth [15], [16] of a developer's overall activities. Unfortunately, the former approach can not be applied for developers who have never participated in a specific project, while the latter does not account for the specific experience of a developer beyond the aggregated activity traces and projects they've worked on. Aggregates of developer's contributions by programming language was previously proposed by Amreen et al. [16], however, experience in a particular language does not immediately confer experience in the variety of libraries or frameworks in that

[1] https://octoverse.github.com/

language which specific applications might rely on. However, this measure of *domain expertise*, or expertise measured by the fluency of using specific APIs, is something that may be of greater concern to projects [15] than a potential contributor's overall skill in a language.

In this work, we try to measure and evaluate such specific domain expertise by defining what we refer to as *Skill Space*, that can be applied to developers, projects, and individual programming languages or APIs as well. In other words, *Skill Space* provides a vector representation for individual developers, projects, programming languages, or APIs, with the topology of the resulting representations (*skill vectors*) reflecting the conceptual and practical (API-related) relationships among these four entities.

To operationalize this *Skill Space* we use the World of Code (WoC) [17], [18] data that contains APIs extracted from changes to source code files (discussed further in Section IV-B) in 17 programming languages. We employ Doc2Vec [19] text embedding that uses as input the dependencies (APIs) of a file modified in each change made by a developer to produce the *Skill Space* representation for individual APIs, developers, projects, and languages. The topology in this space is defined by the alignment (cosine similarity) between vectors representing any pair of developers, projects, APIs, developers and APIs, developers and projects, and projects and APIs.

Compared to similar other methods (see Section III), using *Skill Space* offers the following practical advantages: a) ability to compare the developers, projects, and the APIs in the same space; b) a more faithful representation of expertise due the completeness of the training data (entire OSS); c) cross-language comparison; d) up-to-date representation of expertise based on the latest version of World of Code (WoC) dataset.

Our key contributions consist of a) conceptualization of developer skill/expertise that transcends individual project boundaries making it specific enough to determine its relevance in a novel context; b) postulating the desirable topology of the resulting *Skill Space*; c) proposing Doc2Vec embedding method for operationalizing the *Skill Space*; and d) an empirical evaluation of the proposed topology for this operationalization. A replication package for this paper is made available at DOI 10.5281/zenodo.4457108 [20].

In the rest of the paper, we start by describing the specific research problems in Section II. The related works are described in Section III. We describe our methodology in Section IV, and the evaluation results for our proposed embedding for the proposed *Skill Space* is described in Section V. Details of the replication package we shared is described in Section VI. We describe the limitations to our study in Section VII, the planned extension of the proposed technique in Section VIII, and conclude the paper in Section IX.

II. Research Problem

Our **aim** in this paper is to *define a feasible representation of a developer's expertise in specific focus areas of software development* by gauging their fluency with different APIs. Such medium-granularity representation of developer expertise might serve as a way to get a better understanding of developer skill, help recommender systems that suggest APIs, projects, or contributors, or to increase the trust between external contributors and maintainers of a project. To achieve these goals we define the concept of *Skill Space* and we propose the desirable properties and an operationalization of this concept. We quantify *Skill Space* based on the World of Code (WoC) [17] data that contains information on the APIs extracted from changes to source code files (discussed further in Section IV-B) in 17 programming languages.

A. Postulated properties of Skill Space

The critical feature of our concept of *Skill Space* is the ability to make direct comparisons among three entities: developers, projects, and APIs. The simplest way to accomplish that is to represent each entity as a vector in a linear space. Once such representation is accomplished, for it to be meaningful it needs to satisfy several simple properties: *First*, we expect that the skill vectors of APIs representing similar skills will be close to each other; *Second*, a developer's skill vector should be similar to the representation of the APIs they use most frequently; *Third*, a project's skill vector should be similar to the representations of the APIs used in these projects; *Finally*, we expect the developer representations to be aligned with their subjective perceptions of their API mastery.

Apart from these four fundamental properties, for *Skill Space* to be useful in practice, we expect a few additional properties to be satisfied: *First*, in order to predict API usage, we expect that the new APIs a developer will use in the future should have representations more similar to the representations of APIs they have used in the past compared to randomly selected APIs; *Second*, we expect that new APIs added in projects should also follow a similar pattern; *Third*, we expect that developers will be more likely to join new projects that have representations similar to themselves in the *Skill Space*. We also expect other manifestations of "good" *Skill Spaces* in terms of outcomes of developer work, e.g. the closeness between the skill vector of a developer who submitted a pull request (PR) to a project and that of the target project should have a significant impact on the PR acceptance probability.

Skill spaces satisfying these properties can obviously be of practical and theoretical use, hence our objective in this paper is to construct such a *Skill Space* and to evaluate if it satisfies these desirable properties.

B. Operationalization of Skill Space

To produce the representations in the *Skill Space* we follow previous successful approaches such as degree-of-knowledge model [21] and experience atom [14] that take the uncontroversial position that developer's skill increases as they complete and repeat tasks requiring a specific skill. In the context of software engineering, that involves making changes to the source code. Since we are trying to capture the experience of using programming APIs, we capture the APIs that a modified source code file depends upon. We further discuss the pros and cons of this choice and potential alternatives in

Section VII. Since many of the software source code files are an approximation of software modules [22], the collection of the APIs a file depends upon should represent a specific use case of the functionality instantiated by the file and should, thus, provide implicit dependencies between the APIs utilized in that file. The entirety of all source code, thus, should embody all realized relationships among APIs. Once these implicit relationships among APIs based on changes to the source code are captured, the representation of a developer in the *skill space* could simply be derived from the changes they have made, the representation of a project through changes made in that project, and the representation of a programming language through all changes involving that language.

A naive representation of each change would simply be a high-dimensional vector[2] that represents each of the distinct APIs extracted from over 4 billion changes to the source code files of the languages under consideration. However, such representation of APIs in the *Skill Space* is not very effective or practical, and techniques from text analysis [23] may be used to reduce the dimensionality of this vector. The key underlying assumption of text analysis techniques is that words in a natural language are used in certain combinations to express certain ideas or thoughts. The unsupervised approaches where the relationships are learned directly from the corpus of text assume that the words within a document have to be related and represent some underlying idea expressed by that document. For larger documents sliding window techniques are often used to restrict the length of text where these assumed relationships among words pertain to the same idea. Similarly, we assume that a combination of APIs used in a software module would also reflect some aspects of the functionality implemented in that module. The number of APIs in a single file tends to be quite low as we find in Table I, so there is no need for sliding windows when representing the API. However, text analysis methods need a large corpus of natural language text to extract the semantics from word combinations. We, similarly, expect that the *Skill Space* representation would require a very large corpus of software modules to represent these distinct functionalities (and the associated skill of developers who implemented it). In this paper, we use Doc2Vec [19] text embedding approach to produce the *Skill Space* representation not just for individual developers, but also for individual APIs, projects, and even languages. As a result, the proposed *Skill Space* representation can be used to calculate a direct measure of alignment between any pair of developers, projects, APIs, developers and APIs, developers and projects, and projects and APIs.

C. Evaluation criteria

A conceptual definition also needs practical utility, therefore, to evaluate the suitability of our proposed *Skill Space* representation, we investigate several practical scenarios where developer expertise and trust might come into play, and we expect that a closer alignment between developers and APIs or projects in the *Skill Space* will increase the likelihood of a positive outcome in these events. Specifically, we pose the desirable properties of the *Skill Space* (outlined in Section II-A) as hypotheses which we evaluate to determine if the proposed representation of a developer's specific expertise in the *Skill Space* might be useful in practice by evaluating the following topological properties of the *Skill Space*:

H1: A developer is more likely to choose new APIs that are more closely aligned[3] with themselves.
H2: A developer is more likely to join new projects that are more closely aligned to themselves.
H3: A project is more likely to accept contributions from developers who are more aligned with the project.
H4: Developers better aligned with the project's will have better odds to have their pull requests accepted.
H5: A developer's self-reported API skills are closely aligned to their own representation in *Skill Space*.

III. Related Work

In this section, we present an overview of the historic efforts to measure developer expertise and outline the role of word embeddings in the software engineering literature to clarify the existing gaps we try to address with our work.

A. Developer Expertise

The fascination with developer expertise and its variation began in the early days of software development [24], [25], [26], [27]. Early work was primarily motivated by the need for software project cost estimation and focused on various ways to measure the size of software by adjusting lines of code for different languages or attempting to design ways to have a language-independent measure of software size [28]. The later works embraced the idea that beyond language, each software project requires long and arduous work by a developer to comprehend its internal complexities [29]. This suggested that developer expertise is project and file specific with approaches such as Expertise Browser assuming that each change to a source code file represents an experience atom [14], whereby a developer changing code is forced to understand the files' internal design and, perhaps, impart of their own design through implementing that change. However, these early measures of lines of code written and file-specific experience atoms pertain to expertise within a specific project. They do not provide a general enough profile of developer expertise that can be transferred among software projects.

Contemporary social coding platforms (e.g. GitHub) provide a variety of indicators of developer activity (the timeline of commits) and their social status (followers). This has sparked a variety of research into how developer traces and developer profiles can provide insight into a developer's expertise. These studies include qualitative approaches, such as the one by Marlow et. al. [30], who showed that your developer profile on GitHub can help other developers gauge your general

[2]We counted over 100 million distinct import/use/package/etc. statements in the programming languages from WoC version R

[3]Since we use cosine similarity to measure the closeness between entities, the word "alignment" is a better choice than a more conventional "distance."

coding ability and project-relevant skills, but only at a more general level. Similarly, Singer et. al. [31] interviewed developers and employers to observe how they utilize developer profiles to gauge the quality of a potential new hire. The results showed that profile sites with a "skills" word-cloud representing the technologies (languages, frameworks, etc.) a developer claimed to be familiar with proved to be the most helpful assessment of a developer's expertise. These works indicate that more specific measures, such as language-specific technologies and frameworks, help others gauge the relevant expertise of developers in open source.

There have also been several attempts to automate the process of identifying developer expertise through social coding platforms, e.g. CVExplorer [32] is a tool created to expose developer expertise using a word-cloud of all relevant technologies, frameworks, and general skills by parsing their commit messages and README files. SCSMiner [33] is another tool created to help identify experts on GitHub based on an arbitrary input query. The authors also obtain expertise attributes by parsing README files of projects a developer has contributed to, but they extend this by creating a generative probabilistic expert ranking model to rank developers based on certain skills or expertise one might be looking for. Lastly, Hauff et. al. [34] attempt to match developers with job advertisements based on a developer's expertise by extracting relevant terms from README files and mapping them to the same vector space as job advertisements, and ranking all developer profiles based on the cosine similarity they share with the job advertisements. Cosine similarity has been used in similar contexts in a number of earlier studies (e.g. [35]) and was also used for evaluating the performances of the Doc2Vec and Word2Vec techniques [36]. While all of these approaches are a similar step in the same direction as us, they provide a weaker link between developers and their technologies than desired by utilizing README files as the main source of developer expertise, while we extract language-specific APIs from files a developer has modified. Furthermore, along with measuring a developer's similarity to the technologies they use as attempted in previous work, we also aim to use the APIs to measure the similarity between developers, projects, developers and projects, and projects and APIs.

We also motivate our work through some more recent studies. Montandon et. al. [15] present an approach to determine experts for three JavaScript libraries. The authors identify developers who have made changes to projects that depend on these libraries and conduct a survey with 575 developers to obtain their self-reported expertise. Using these survey results as validation, the authors argue that their clustering approach is feasible and can be used to identify relevant experts. However, they also present the shortcomings of using basic GitHub profile features for machine learning classifiers to predict expertise in software libraries. We utilize the survey dataset provided by the authors for our own evaluation and also attempt to better predict developer expertise in software libraries, an area in which the authors achieved poor performance.

The more recent Import2Vec [37] paper produces embeddings for each imported package. The authors do such embeddings for JavaScript, Python, and Java, and provide some qualitative evidence suggesting that these embeddings of APIs accurately reflect different functionality profiles by providing a number of examples where the similar APIs also appear to implement similar functionalities.

Unfortunately, none of the proposed approaches are suitable for directly comparing developers and projects, as neither developers nor projects are accurately represented in the same vector space as the API embeddings. It is, therefore, not clear how Import2Vec embeddings can be used to represent developers' domain expertise nor if such profiles would accurately reflect developer proficiency. Furthermore, the Import2Vec approach can not be applied in a cross-language context. Our proposed approach tries to address this gap by constructing a *Skill Space* representation that, on one hand, may transcend the specific programming languages, and on the other hand, may identify a meaningful representation that can be matched with skill sets of other developers or projects.

B. Vector Embedding in Software Engineering

Vector embeddings have been used in software engineering for various tasks, e.g. using natural language associated with coding to determine sentiment [38], using writing style in commit messages to determine developer identity [39], or improve requirements traceability [40]. In these cases the natural language techniques do not need to be modified substantially as the underlying data represents natural language.

Even more techniques have been applied to model programming language source using text analysis techniques. For example, these approaches can improve Interactive Development Environments (IDEs) by performing next token prediction [41], suggesting better class names [42], or even automatic patching [43]. In a recent paper, Alon et al. [44] proposed a method for representing snippets of code as continuous distributed vectors (code embeddings).

The attempt to provide a common embedding space for natural language and code was proposed by Ye et al. [45] by training the natural language models on the API documentation and the applications that use these APIs.

Unlike these approaches, we focus on training the models on the APIs used in files that undergo a code change. While we do not go to the level of a specific function used in the API, we treat each import/use statement as an indication of the specific functionality provided by the corresponding package. As noted above, the best natural language analysis techniques typically exploit the order of the words in a text document (such as commit messages, requirements, or documentation). The programming language modeling techniques also rely heavily on the specific sequence that is necessary to do an accurate prediction of the next token, for example. In contrast, our work looks at embedding package imports within source code files, where the order of import statements may not be important. Thus, the existing techniques that attempt to model the order of the tokens need to be modified to fit our purpose.

IV. Methodology

To represent our entities in the *Skill Space* we need a very large corpus of software and we turn to World of Code (WoC) due to its size, coverage, data quality, and the ability to obtain desirable subsamples as described below.

A. Data Source: World of Code

WoC is a prototype of an updatable and expandable infrastructure, aimed at supporting research and tools that rely on version control data from open source projects that use Git. It stores large and rapidly growing amounts of data that approximates the entire FLOSS ecosystem, and provides capabilities to efficiently extract and analyze the data at that scale. In addition to storing objects from all git repositories, WoC also provides relationships among them. The primary focus of WoC is on the types of analyses that require global reach across FLOSS projects, so it is the most appropriate choice for answering the research questions we presented here.

WoC data is versioned, with the latest version labeled as R, containing 7.9 billion blobs, 2 billion commits, 8.3 billion trees, 17.3 million tags, 123 million projects (distinct repositories), and 42 million distinct author IDs. This version of WoC data was collected during March, 2020.

As is often the case with datasets of this size, certain data cleaning steps are critical for obtaining meaningful results. Conveniently, in addition to providing access to the raw data, WoC offers advanced data augmentation capabilities. Two such techniques were used in this study for data preprocessing: fork resolution (deforking) and developer identity resolution, since our *Skill Space* representation considers the relationship among projects, developers, and their API usage. Accurately representing all three of these entities is, therefore, necessary.

1) Project Clones: Fork Resolution: Git is a distributed version control system that, inherently, makes it easy to clone or fork Git projects. This, however, creates a unique data cleaning problem for WoC, which has over 116 million projects, many of which are clones or forks of another project. This poses several problems for our expertise analysis. One such problem is that a developer who contributes to a highly-cloned project will have their commits appear in the remaining cloned projects as well, e.g. if a developer contributes to one project using the `flask` module in Python and 10 other people clone this project and make little to no changes, the developer would be attributed with having worked with `flask` on 11 different projects, rather than just one.

To address this, we use the dataset published in [46], which applies the Louvain community detection algorithm to a massive graph consisting of links between commits and projects in WoC (because two projects are highly unlikely to share the same exact commit unless they are clones). We leverage that work to combine commits from the forked projects and ensure that we do not count the same project-related information multiple times due to these forks/clones.

2) Identifying a Developer: Identity Resolution: The WoC dataset contains the author ID for each git commit, which would, ideally, correspond to a single developer, and could be used to aggregate all commits associated with the author ID and perform our expertise analysis. However, this is seldom the case as the author ID is obtained from the git configuration file residing on the developer's laptop/desktop/server where they use git. The author ID tags, therefore, often differ between commits made on different computers used by a developer. As a result, many developers have multiple author IDs (with some that they might not even be aware of) in WoC collection that, collectively, need to represent the same developer.

To address this, we have used a dataset shared by Fry et al. [47] that resolves the 38 million author identities in WoC version Q by creating blocks of potentially related author IDs (e.g. IDs that share the same email, unique first/last name) and then predicting which IDs actually belong to the same developer using a machine learning model. The approach identified over 14 million author IDs belonging to at least one other author ID. From this set, around 5.5 million developers were identified, with a median of two author IDs per developer. When performing the expertise analysis described in this paper, we identify each developer using the new associations created by the identity resolution approach. This allows us to create a much more accurate representation of each developer's API usage and expertise and helps us avoid comparing two author IDs that are in fact the same developer.

B. API Extraction

To obtain developer API usage, we utilize the language mappings inside WoC. These mappings contain APIs extracted from changes to source code files in C, C#, Java, FORTRAN, Go, JavaScript, Python, R, Rust, Scala, Perl, Ruby, Dart, Kotlin, TypeScript, and Julia languages, as well as source code present in Jupyter (iPython) Notebooks [4]. The mappings are created by first obtaining all files in WoC with extensions used by each of the languages listed previously. For each language, the WoC file-to-blob[5] map is used to obtain all blobs associated with language-specific files. The content of the resulting blobs is then parsed for import statements depending on the syntax of each language (e.g. `#include` in C, `import` in Java/Python, `use` in Perl, the dependencies in the package.json file for npm, and so forth).

Each of these blobs (versions of the source code) is further mapped to the commit(s) that produced it and projects that have that commit. Timestamps, authors, and projects of these commits are then associated with the blob as well as with the APIs parsed from that blob resulting in the following tuple (programming language, repository, timestamp, author id, timestamp, API1, ...). We use deforking and author aliasing described above to transform repository into deforked project ID and author id into aliased developer id. The timestamp allows us to perform time-based prediction in some of our models as discussed in Section IV-E.

Thus, the final mapping and data used by some of the models is a compressed file of entries containing:

[4] https://jupyter.org/
[5] https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

`project;timestamp;developer;API1;API2;...`, where each entry represents all modules/APIs included in the file that the developer added to the project at the instance in time. There is a unique set of entries for each language listed earlier, and they are stored in separate compressed files. While this mapping serves as the base data for most of our analysis, there are several intermediate steps that require a transformation of the provided mapping as well.

C. Summaries of API usage

The previous subsection describes the procedures used to obtain the data from WoC (version R) that captures for each modification to the source code the programming language, the timestamp, the developer, the project, and the list of "import" statements.

Table I shows the number of deltas (changed blobs) associated with each language as well as the number of distinct authors and projects involved. The largest number of delta by far involve C and C++ (we do not distinguish between the two), followed by Java and Python. The relatively low number of JavaScript delta relates to the way dependencies are specified in JavaScript projects where a single file (`Package.json`) is used to specify the dependencies while in C, Java, or Python, every source code file needs to include its dependencies explicitly.

Notably, Java language dominates in terms of the number of unique APIs, presumably because the APIs in Java can be specified using global namespace, while for other languages they are defined by the package managers or within the source code files (like .h files in C/C++) that may share the same name but be otherwise unrelated (see Section VII).

As noted above, the total number of distinct APIs we observe is far higher than the number of words in a natural language putting computational strains on the text analysis methods designed to deal with many orders of magnitude smaller dictionaries. Moreover, the order of the APIs in source code files is not important, hence we need to apply methods that do not attempt to model the sequences. While some early text analysis methods, such as LSI, work strictly on the bag of words (BOW) and are immune from this problem. Others, such as continuous bag of words (CBOW), try to predict words within a certain window size. The wider the window, the more complicated and time consuming it is to fit these models. To investigate what window sizes might be appropriate, we investigate the distribution of the number of distinct APIs within a single delta (a modification by a single commit to one source code file).

Table I shows the fraction of delta for each language where the number of distinct APIs is less than 30 and also shows the maximum number of APIs. Again, JavaScript is an outlier here since a single file (package.json) defines APIs for the entire project. We chose to consider the window size of 30 or less for the CBOW models since it captures most of the deltas for all languages. The deltas with huge numbers of APIs used may indicate unusual cases or outliers that may not bring much information to which APIs are used together and it is not unreasonable to exclude those from consideration.

The total number of delta and the number of distinct APIs pose serious computational challenges if we want to fit the complete dataset obtained from WoC with 4.3B delta and over 100M distinct APIs not counting the number of distinct projects and authors. We, therefore, fit several smaller datasets by filtering the data to a more manageable size.

First, for the multi-language model, we focus on developers that made between 100 and 25K commits partially to exclude the bot activities and partly to consider ordinary but productive developers, since by the premises of our proposed hypotheses, we're trying to focus on developers who have a good amount of contributions in social-coding platforms, since our assumption is that they will use new APIs, contribute to multiple projects, and will submit a number of pull requests. This filter reduces the total number of delta down to 1.2B. For language specific models we are dealing with much smaller datasets, but we can decrease that size even further by randomly sampling projects or developers. We used these smaller samples to debug the techniques and to find the parameters for the *Skill Space* embeddings that produce feasible results before running the computation on the entire model.

D. Vector Embedding

Since the total number of possible APIs that can be used by a developer or a project across different languages is extremely large and the naive embedding, representing API usage as a component, of over a 100M-dimensional vector is not practical, we reduce the dimensionality of the *Skill Space*. We chose to employ Doc2Vec embedding method since it is capable of embedding not only the APIs themselves but developers and projects at the same time. It is also one of the most efficient embeddings to compute: an important consideration given the large data corpus we handle.

Word2Vec, [23] is a highly computationally efficient algorithm used to create a numerical representation for a word using a continuous bag of words or skipgram (two distinct algorithms). The primary assumption of Word2Vec is that only words that are close together in a document are semantically related. In our context, that assumption doesn't hold, because there is no semantic order for the APIs used by a developer or a project. We address this potential problem by using the continuous bag of words algorithm with a wide window of 30 words. Since the number of APIs associated with a single blob rarely exceeds 30 as shown in Table I, the algorithm in practice predicts one API of a blob using all remaining APIs.

Doc2Vec is an extension of Word2Vec, where in addition to word (API) embeddings, the model also produces the embeddings for an arbitrary set of tags associated with a group of APIs, as is the case when an author, a project, and a language is associated with the set of APIs extracted from each change of every file. The continuous bag of words analog in Doc2Vec corresponds to obtaining doc-vectors by training a neural network on the synthetic task of predicting a word based on an average of both context word-vectors and the full

TABLE I
SUMMARY OF DATA RETRIEVED FROM WOC-VERSION.R PER LANGUAGE

Language	Delta (Changed blobs)	Authors	Projects	Distinct APIs	Fraction of deltas (changed blobs) with 30 or fewer APIs	Max no. of APIs in one delta (changed blob)
FORTRAN	1,628,760	24,898	15,623	59,349	0.98	106
Julia	1,297,134	18,666	35,723	104,725	0.99	108
R	6,822,662	361,754	516,678	85,255	0.998	117
iPython	12,160,775	793,261	1,154,120	687,085	0.99	1,158
Perl	18,780,774	480,615	547,115	58,942	0.999	109
Rust	13,599,452	95,712	148,327	818,686	0.99	118
Dart	7,036,000	116,317	164,360	467,863	0.99	165
Kotlin	28,129,485	281,469	429,071	6,233,673	0.96	1,096
TypeScript	239,416,852	1,605,563	2,253,291	7,324,019	0.99	1,013
C#	220,871,444	2,092,316	3,092,761	6,648,357	0.997	150
Go	123,432,323	490,967	662,355	245,102	0.995	1,207
Scala	36,361,141	176,414	210,175	3,571,593	0.99	1,288
Ruby	74,618,824	1,222,886	2,343,825	669,297	0.997	1,002
JavaScript	55,609,812	3,362,191	7,347,050	1,105,918	0.67	10,014
Python	612,708,423	4,795,735	6,820,899	17,227,676	0.99	1,001
C/C++	1,780,602,124	3,656,965	4,704,446	2,553,521	0.99	1,007
Java	1,106,084,606	5,063,200	7,512,800	85,079,403	0.92	1,004

document's doc-vector. We used the Gensim framework for evaluation due to its high performance.

E. Evaluation strategies

The evaluation strategy involves fitting a Doc2Vec model on past data, where each document represents the APIs encountered in a single delta and the document tags represent the language, the project, and the developer. The resulting model thus creates vectors for each API, for each developer, each project, and each language. We then obtain new APIs a developer uses during the testing period, the new projects the developer joins, and the new developers who join a project during the testing period. The alignment to these factual APIs/projects/developers are then compared with randomly chosen sets of APIs/projects/developers of the same size.

We chose the dates so that we have a fairly short testing period starting from February, 2019. All changes prior to that date were used to fit the model and the activities past that date to check the predictions. We used these dates for predicting new APIs, developers joining new projects, and projects accepting new contributors.

For PR acceptance and self-reported expertise, we fitted models based on data prior to Feb 14, 2018 and tested on activities after that time in order to have a sufficient number of accepted or rejected PRs during the testing period for most developers. To conduct the study of pull request acceptance, we sourced the pull request dataset [48] used by Dey and Mockus [6] for verifying our hypothesis and studying the effects of technical and social factors on PR acceptance. The dataset contained information on 470,925 PRs from 3349 popular NPM packages and 79,128 GitHub users who created those. We filtered this dataset to only include developers who made between 100 and 25,000 commits, similar to what we did for testing earlier hypotheses. In addition, we removed small projects that didn't have any API calls. After filtering, we were left with 150,173 PRs made by 14,784 developers for 1860 GitHub projects.

Then, as in the other cases, we proceeded to obtain embeddings for the developers and projects using past data and then model the acceptance rate during the future PR activity using the binomial regression with the independent variable representing the alignment of the developer and project vectors where the PRs have been submitted to together with the predictors used by [6]. We once again use February, 2018 to separate training and test data.

Finally, we use a previously reported survey [15] of JavaScript developers to compare how aligned each surveyed developer is to the the API in which developers were reported to be proficient. Since the survey did not include APIs where developers reported being not proficient, we randomly chose ten other APIs under the assumption that they might not be equally proficient in these 10 randomly chosen APIs. As in other comparisons, we report the difference in alignment between the self-reported expert APIs and the randomly chosen APIs. To make the *Skill Space* representations commensurate with developer self-reported expertise, we only use the data close to the time when the survey was conducted (also February, 2018).

Given the very large vocabularies for the APIs, we chose a relatively high-dimensional vector of 200 for *Skill Space*, to make sure there is enough flexibility to represent the extremely large number of potential skills. We excluded APIs that occur in fewer than five deltas to increase computational efficiency and, also, avoid highly uncertain embeddings. As discussed above, we chose a window size of 30 to ensure that the order of APIs in the delta does not matter. Finally, we chose the negative sampling parameter to be 20. It tends to speed up the convergence by creating synthetic samples (API combinations) that do not exist in the data and penalizes the model if it produces a good fit for such "negative" samples. All of these parameters were chosen after extensive experimentation fitting the models on manageable-size datasets.

TABLE II

SUMMARY OF PER-LANGUAGE RESULTS OF T-TEST SHOWING THE DIFFERENCE OF ALIGNMENTS BETWEEN A DEVELOPER'S REPRESENTATION IN THE *Skill Space* AND THE APIS THEY USED IN FUTURE VS. RANDOM APIS THEY DIDN'T USE (IN THE SAME LANGUAGE). P-VALUES <1E-200 ARE SHOWN AS 0.

Language	Estimated Difference in Means	95% Confidence Interval	p-Value
Dart	0.41	0.39 - 0.43	3.12e-92
Julia	0.21	0.15 - 0.27	8.57e-05
R	0.14	0.09 - 0.20	1.46e-06
iPython	0.20	0.18 - 0.22	6.68e-65
Perl	0.05	0.03 - 0.06	2.85e-13
Rust	0.21	0.20 - 0.22	2.01e-151
Kotlin	0.21	0.20 - 0.22	1.09e-139
TypeScript	0.23	0.22 - 0.24	0
C#	0.25	0.23 - 0.26	6.16e-137
Go	0.15	0.14 - 0.15	0
Scala	0.20	0.19 - 0.22	8.45e-89
Ruby	0.17	0.16 - 0.18	3.80e-188
Java	0.13	0.12 - 0.13	0
C/C++	0.13	0.13 - 0.13	0
Python	0.12	0.12 - 0.12	0
JavaScript	0.10	0.10 - 0.10	0
FORTRAN	-0.11	-0.73 - 0.51	0.268

V. RESULTS

A. Qualitative Evaluation of Skill Space Embeddings

For a qualitative evaluation of our proposed embedding, we decided to observe which APIs are reported as similar to others in the same language, and also which APIs provide similar functionality across different languages. For the Python package "pandas", we observed that the APIs reported to be most similar are indeed the ones that are most frequently used with it, primarily for data manipulation/ data visualization/ machine learning applications.

```
>>>mod.most_similar('pandas')
>>>[('matplotlib.pyplot', 0.8), ('numpy', 0.8), ('
    seaborn', 0.78) ]
```

We can also do some arithmetic with the resulting vectors by asking what are packages the most similar to Python "pandas" package in R language:

```
>>> mod.wv.similar_by_vector(-mod.docvecs['PY']  +
    mod.docvecs['R'] + mod.wv.get_vector('pandas'))
>>> [('data.table', 0.83), ('dplyr', 0.82) ]
```

As we see, the most popular data frame (after which "pandas" was modeled) packages are most similar. Also, only R packages appear in the most similar list even though we start from the python package and move in the direction of R.

B. Examining H1: New APIs used by developers are closely aligned to themselves in the Skill Space

We follow the process outlined in Section IV-E to get the alignment between embeddings of each developer, created by the APIs they used during the training period, and the new APIs used in the testing period and a set of random APIs *in the same language* that they did not use. We did the calculation separately for each language to get a clearer understanding of the performance of our proposed *Skill Space* embeddings at that level.

We were unable to fit model for the entire corpus (it would have taken several months on a fast multi-processor server). Instead we sampled 36K projects that contain 1.2B delta by 690K authors in all 17 languages. The amount of data for each language is similar to that in the entire corpus.

The paired t-test results in Table II show that the APIs used in the future were indeed more closely aligned as compared to random APIs they didn't use. The amount of data for the FORTRAN language in the sample was too small to get a statistically significant difference.

C. Examining H2: A developer is more likely to join a new project that is more closely aligned to them in the Skill Space

Here we try to validate the expectation that the new projects a developer will join (make an accepted contribution to) would be more closely aligned with the developer's *Skill Vector* than a randomly selected project.

As described in Section IV-E, we calculated the alignment between embeddings of each developer and the projects they contributed to and a set of random other projects *in the same language* that they did not contribute to, and measured if there is any significant difference between them using t-test. We found there is indeed a significant difference (p-value $<$ 2.2e-16) with a difference between the estimated means of the cosine similarity of 0.017 and 95% confidence interval of $[0.013, 0.021]$. This supports our hypothesis that there is a similarity between the developers vectors and vectors of the projects they contribute to in future.

D. Examining H3: A project is more likely to accept contributions from developers who are aligned to the project in the Skill Space

One of the potential *Skill Space* applications is increasing trust. New contributors who have *Skill Vectors* aligned to a project's *Skill Vectors* should be more likely to have their contributions accepted all other factors being equal. Their skill (if it exists) should manifest itself in the technical aspects of the PR and, therefore, might be recognized by the maintainers of that project. Once again, we constructed skill vectors for the developers who contributed to a project, measured the alignment between them and the skill vectors of the corresponding projects, and compared them with the alignment between skill vectors of a project and the skill vectors of randomly chosen developers who did not contribute to that project. The differences between the alignments were found to be significant using t-test, with p-value $<$ 2.2e-16, an estimated difference of means between the alignments being 0.141, and a 95% confidence interval of $[0.126, 0.156]$.

E. Examining H4: A developer whose Skill Space is aligned more closely to the project's Skill Space will be more likely to have their pull requests accepted

To more directly evaluate the previous hypothesis, here we restrict our attention to Pull Requests (formal external contributions) where we can see not only the cases when the contribution was accepted as above, but also cases where the

TABLE III
RESULT OF LOGISTIC REGRESSION MODEL PREDICTING PR ACCEPTANCE. *Cosine Similarity between Developer and Project* IS THE VARIABLE WE INTRODUCED IN THIS STUDY (HIGHLIGHTED IN GRAY). OTHER VARIABLES ARE ADOPTED FROM [6]. THE NON-SIGNIFICANT VARIABLE IS HIGHLIGHTED IN RED, BINARY VARIABLES ARE IN BLUE

Predictor	**Coefficient ± Std. Error**	**p-Value**
(Intercept)	0.654 ± 0.093	2.24e-12
Cosine Similarity between Developer and Project	*0.396 ± 0.084*	*2.10e-06*
creator_submitted	-0.120 ± 0.009	$< 2e-16$
creator_accepted	0.874 ± 0.033	$< 2e-16$
repo_submitted	-0.026 ± 0.005	1.62e-06
repo_accepted	2.864 ± 0.056	$< 2e-16$
dependency:1	-0.212 ± 0.021	$< 2e-16$
age	-0.221 ± 0.004	$< 2e-16$
comments	-0.173 ± 0.013	$< 2e-16$
review_comments	0.342 ± 0.011	$< 2e-16$
commits	-0.360 ± 0.015	$< 2e-16$
additions	-0.015 ± 0.008	0.05
deletions	-0.035 ± 0.006	$< 2e-16$
changed_files	-0.151 ± 0.016	$< 2e-16$
contain_issue_fix:1	0.123 ± 0.020	1.89e-09
user_accepted_repo:1	1.326 ± 0.027	$< 2e-16$
creator_total_commits	0.086 ± 0.009	$< 2e-16$
creator_total_projects	0.015 ± 0.007	0.029
contain_test_code:1	-0.418 ± 0.324	0.197

contribution was made but not accepted. As previously, we hypothesize the developers' alignment with projects in *Skill Space* should have a significant impact on PR acceptance probability, with a better alignment being associated with a higher chance of acceptance.

We used a regression model for this analysis, as mentioned in Section IV-E. The result of the Logistic Regression model is presented in Table III, which shows that the alignment between developers and projects remains a significant variable even after accounting for the other social and technical factors described in [6], i.e. this variable describes a factor which is not captured by other technical and social factors. We also notice that the coefficient for this variable is positive, i.e. the closer a developer's alignment is to a project, the higher the chance of their PR being accepted, which validates our proposed hypothesis. We checked the Variance Inflation Factors for these variables and found the values to be less than 2.5 in all cases, signifying that there is no multicollinearity effect. The variable 'contain_test_code' was found to be insignificant, similar to [6]. However, the variable 'deletions' was found to be insignificant in [6] but it's significant here, which could be because we're only focusing on a subset of the data used in that study.

F. H5: A developer's self-reported API skills are closely aligned to themselves

The final question we pose is whether the representations in *Skill Space* align with developer's self-reported opinions about their own expertise related to a specific technology.

We obtained data from the replication package of [15] that surveys a sample of GitHub users to create a ground truth for self-reported developer expertise in the studied libraries. In this survey, the participants declared their expertise (on a scale from 1 to 5) for three JavaScript libraries: *mongodb*, *react*, and *socketio*.

TABLE IV
RESULT OF LINEAR REGRESSION MODELS: (A) EXPLAINING DEVELOPER-API ALIGNMENT (R^2 VALUE: 0.90); (B) EXPLAINING SELF-REPORTED SKILL SCORE (R^2 VALUE: 0.92)

(A)

Predictors	Estimate ± Std. Err.	p-Value
API:mongodb	0.249 ± 0.013	< 2e-16
API:react	0.307 ± 0.011	< 2e-16
API:socketio	0.422± 0.012	< 2e-16
log(No. of Commits)	0.000± 0.001	0.9
Self-Reported Score	0.014± 0.003	1.8e-6

(B)

Predictors	Estimate ± Std. Err.	p-Value
API:mongodb	2.5 ± 0.10	< 2e-16
API:react	2.9 ± 0.08	< 2e-16
API:socketio	1.9 ± 0.12	< 2e-16
log(No. of Commits)	1.1±0.012	< 2e-16
Developer-API Alignment	0.98 ± 0.21	1.81e-6

Similarly to previous experiments, we obtain *skill space* representations for survey participants and the three APIs. We investigate if the skill space similarity can be explained by the self-reported score by fitting a linear regression model and find that the self-reported score explains increases in alignment to each API as self-reported expertise score increases. The result of the linear regression model is shown in Table IV(A).

Finally, we try to model the self-reported score using the amount of activity (commits) as reported in [15] and adding the *Skill Space* similarity. Again, we find that the increase in skill alignment has a statistically significant positive relationship with the self-reported score even after adjusting for the direct measure of experience based on the number of commits. The result of the model is shown in Table IV(B).

In summary, we find that the proposed *Skill Space* embedding based on Doc2Vec models of the APIs in files changed by a developer has a strong and statistically significant relationship with the self-reported developer expertise. Furthermore, even after adjusting for the less granular measure of experience (number of commits), we still see that *Skill Space* representation has a strong explanatory power.

VI. REPLICATION PACKAGE

The replication package for this paper is made available through Zenodo under CC 4.0 license at DOI 10.5281/zenodo.4457108 [20]. The data we share include the input data (processed), with the details of the APIs in each blob modified by OSS developers who made between 100 and 25,000 commits, all the scripts used by us for the evaluation, and the steps for replicating the results presented in the paper (in the README file). Although we do not share it as a tool/package, which would be difficult to run without access to the World of Code dataset (we are working on extending the publicly available capability of the World of Code dataset, which would make such a tool practical in near future), we share the input data, so that researchers can fit their own models and experiment with the dataset. We provide a detailed account of the steps we took and share the scripts we used so that researchers can replicate our findings. We also share the pre-trained Doc2Vec models, so that researchers can use them for their applications without having to re-train the model.

VII. Limitations

It is important to note the primary objective behind introducing the concept of *Skill Space*: the ability to compare developers, projects, languages, and APIs with the ultimate goal of better measuring developer skills and at facilitating ways to make open source software development more effective by creating signals about the developers' expertise that is more general than the modification of individual files, but more specific than their volume of overall activity.

The objective of this work is to conceptualize *Skill Space*, to list some of its properties, and to demonstrate that it is possible to construct it on a very large corpus of programming languages and APIs. As such, we focus on demonstrating the feasibility and novel applications enabled by the proposed measure rather than trying to compare our method with existing ones since existing developer expertise measures are not suitable for directly comparing developers, projects, and the APIs used by/in them.

Our results, consequently, have to be interpreted with care. First, our definition of developer skill is constructive and practical. We are only concerned that it reflects postulated measures of performance and has some agreement with developers' subjective perceptions. Further work is needed to ascertain if it satisfies any additional properties or is suitable for non-constructive definitions of skill.

Specifically, the definition of *Skill Space* we chose is based on API usage, but the skill embeddings can be conducted for other types of skills as well.

We validate the proposed *Skill Space* by checking if it would satisfy the intuitive properties the *Skill Space* should exhibit, but there may be additional properties we do not consider (and the proposed *Skill Space* does not satisfy). For example, our primary concern in this work is to capture the aspects of developer expertise related to the APIs they use and we are not concerned with other types of expertise, such as their proficiency to do good design, architecture, testing, and so forth, or with their ability to communicate with other developers.

The particular mechanism of what it means to use an API may be refined. We only consider if the version of the file modified by a developer has certain import statements, but do not verify that the API is actually exercised in the file, and we also do not check if the developer made a change to the part of the code that exercises a specific subset of the API used in the file. Moreover, it can be argued that just because a developer uses some API in a file doesn't mean that they are expert in using that API since code snippets are often copied and pasted from different sources. However, our assumption is that a developer should have a basic familiarity with the APIs used in the files they modify, at least more than a random other API they have never been associated with, and, as noted by Lucassen and Schraagen [12], *"domain familiarity can be seen as a weaker form of domain expertise."*

Since our aim is to capture the profile of expertise as a trust-building support and we attempt to create such measures that equally apply to individual APIs, projects, and developers, there are no golden datasets that could be created to evaluate the objectivity of all such measures. Specifically, there is no convincing test everyone would agree upon that a developer is a good fit for a project. As such, we can evaluate the goodness of the measures we propose through several indirect means e.g., can a specific developer be trusted when they make a contribution if there has been no prior interaction between the developer and maintainer? As we noted above, different languages have different conventions in which APIs are declared and these differences may play a role or need to be taken into account in order to improve upon the proposed implementation of the skill space.

It may be surprising how a relationship between APIs in different languages can be established using our methodology (APIs are language specific and every blob contains only APIs from a single language). However, a large fraction of developers have modified files in multiple languages and many projects contain files in several languages. Since developers and projects serve as tags in Doc2Vec model, such instances appear to provide information needed to establish association of the APIs across languages.

There are a few other shortcomings associated with our approach, e.g. our method of measuring expertise can't be applied to complete newcomers, since they likely have worked with very few APIs, and their representation in the *Skill Space* is likely to be unstable. However, these developers are not our target audience, we are trying to focus on developers with a moderate amount of contribution record who are trying to join a new project, trying to use a new API, or aiming to get their contributions accepted in a project. Similarly, rare APIs may not be accurately represented as the corpus may not have sufficient number of instances of using such API.

Many potential improvements to the embedding approaches could be considered. Since our concern was to demonstrate the feasibility of the approach, we chose an established and computationally efficient *Doc2Vec* method. With the field of text analysis rapidly evolving, we expect that future work will develop more accurate methods that are likely to vary with the task (API/developer/project/PR prediction), vary with the programming language, or use alternative embedding techniques. We also expect further work to refine the parameters of embedding methods as well. Our largest model took more than three weeks to fit, limiting the ability to run performance-optimization experiments.

While demonstrating the use of *Skill-Space* based embedding, we only compared our results with a random selection of APIs/developers/projects. A more practical application would be to use our method to predict, for example, which APIs a developer will use in future, and test the prediction accuracy.

Another potential shortcoming of our approach is that it is not completely resistant to hacking (similar to most other existing methods of reporting developer expertise) since it is possible to generate a number of toy projects that use a specific set of APIs to give an impression that the developer who set up those projects is skilled with such APIs. However, this is not completely straightforward either, since it involves the creation of several toy projects. Further refinements of our method are in progress to make it more robust.

While we model a very large corpus of software, it all represents open source development. The activity of developers in non-public repositories and non-public software are not captured in this analysis. Future work is needed to apply our techniques on proprietary code bases to ascertain if *Skill Spaces* can be operationalized in the same way or some adaptations are needed to take into account the differences in the development process.

In terms of external validity, our method can only account for the developers' expertise, while it is possible that other factors (e.g. change in job responsibilities) might influence developers when choosing APIs to use (H1), or which projects to contribute to (H2), which won't be captured by our approach.

VIII. Future Work

Previous sections discussed a variety of promising approaches for future work to improve the quality of *Skill Space* representations and to evaluate alternative ways to capture to what extent a particular change may require/increase API-related skills. We can use the *Skill Space* embeddings of the developers, projects, and APIs together with more efficient machine learning models to further test the applicability of our approach.

More far-reaching extensions of *Skill Space* would be to include non-technical skills, such as communication and collaboration skills that are also very important in establishing trust. We could, potentially, use traces of development activity related to developers ability to communicate, write high-quality code, respond to issues, get pull request accepted and other important skills. This, however, would require a way to evaluate the quality of the artifacts a developer produces and the quality of the practices they employ.

A recent paper [16] utilized WoC as a way to estimate the reputation of a developer using a tool (DRE) that serves up developer profiles and provides a broad overview of many facets of a developer's activity, focusing on both technical and social aspects. The *Skill Space* embedding presented in this paper can be used to enhance such developer profile tools, and can also provide recommendations for both the developers (e.g. similar projects that they might consider joining, similar developers they might want to work with in the future, and similar technologies/APIs they might consider working with etc.) and the project maintainers (e.g. potential contributors who might possess relevant skills).

Further application of our approach might include: a) detecting if a developer is actually a bot by analyzing the concentration of their *skill vector* (similar to [49], [50]); b) checking the alignment between *skill vectors* of different developers for identity resolution (similar to [47]); c) analyzing the *skill vectors* of the developers in a project to infer the transparency of the corresponding software supply chain [51], [52], [53], [54], [55], [56].

IX. Conclusion

We have established a proof-of-concept for *Skill Space*: an approach to represent packages (APIs), developers, languages, and projects in the same vector space with a topology that satisfies several practically-relevant criteria, such that the representations of developers (projects) in *Skill Space* are similar to the representations of the APIs they use (contain). Furthermore, *Skill Space* representations are predictive of the future API usage by developers, developers joining new projects, and it also affects the probability of a developer's pull requests being accepted. Finally, these representations are aligned with developers' self-reported expertise.

As with all data-intensive techniques, only entities that have sufficient data can be accurately represented, but a large volume of public data from OSS projects can help. The simplicity of the proposed estimation techniques make it easy to apply them within enterprises, producing company-specific *Skill Spaces* that could be integrated with the OSS data.

Two observations were primary motivator for us to conceptualize the medium-granularity expertise created from the implicitly defined relationships among APIs in the vast corpus of open source software projects:

1) Contemporary software development increasingly involves complex dependency chains with much of the software product depending on software developed by unknown and unfamiliar teams;
2) The ability of developers to use specific libraries and frameworks (in the dependency chains noted above) is an important factor that determines their ability to complete programming tasks.

We hope that the progress on measuring and understanding technical aspects of expertise may prove helpful in developing approaches that establish trust between maintainers and contributors who had no prior interactions. We also hope that it may shed some light on the causes of the vast differences in programmer productivity and help research on developer learning trajectories. We shared source code and the datasets used in this work, and are also working on making them accessible via a web interface through the World of Code website (https://worldofcode.org/), which can be used to calculate individual vectors and similarities between different entities, with the intention of facilitating replications, further improvements in the approaches to construct *Skill Space*, and, more generally, supporting further studies in this area.

Acknowledgements

The work has been partially supported by NSF awards 1633437, 1901102, and 1925615.

REFERENCES

[1] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.

[2] A. Mockus, R. T. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 1–38, July 2002. [Online]. Available: http://dl.acm.org/authorize?39725

[3] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 345–355.

[4] J. Zhu, M. Zhou, and A. Mockus, "Effectiveness of code contribution: From patch-based to pull-request-based tools," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 871–882. [Online]. Available: https://dl.acm.org/authorize?N27515

[5] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, 2010, pp. 495–504.

[6] T. Dey and A. Mockus, "Effect of technical and social factors on pull request quality for the npm ecosystem," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ser. ESEM '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–11. [Online]. Available: https://doi.org/10.1145/3382494.3410685

[7] J. Tsay, L. Dabbish, and J. Herbsleb, "Influence of social and technical factors for evaluating contribution in github," in *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2014, p. 356–366. [Online]. Available: https://doi.org/10.1145/2568225.2568315

[8] T. Dey and A. Mockus, "Which pull requests get accepted and why? a study of popular npm packages," *arXiv preprint arXiv:2003.01153*, 2020.

[9] P. Audris Mockus, C.-P. James Herbsleb, c.-P. Randy Bradley, c.-P. Bogdan Bicescu, and c.-P. Russell Zaretsky, "Bigdata: Collaborative research: Ia: Oscar - open source supply chains and avoidance of risk: An evidence based approach to improve floss supply chains," National Science Foundation, IIS-1633437 1,300K, 2016-2018.

[10] J. Marlow and L. Dabbish, "Activity traces and signals in software developer recruitment and hiring," in *Proceedings of the 2013 conference on Computer supported cooperative work*, 2013, pp. 145–156.

[11] J. Hahn, J. Y. Moon, and C. Zhang, "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties," *Information Systems Research*, vol. 19, no. 3, pp. 369–391, 2008. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/isre.1080.0192

[12] T. Lucassen and J. M. Schraagen, "Factual accuracy and trust in information: The role of expertise," *Journal of the American Society for Information Science and Technology*, vol. 62, no. 7, pp. 1232–1242, 2011.

[13] C. Jergensen, A. Sarma, and P. Wagstrom, "The onion patch: migration in open source ecosystems," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 70–80.

[14] A. Mockus and J. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *2002 International Conference on Software Engineering*. Orlando, Florida: ACM Press, May 19-25 2002, pp. 503–512. [Online]. Available: http://dl.acm.org/authorize?24835

[15] J. E. Montandon, L. L. Silva, and M. T. Valente, "Identifying experts in software libraries and frameworks among github users," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 276–287.

[16] S. Amreen, A. Karnauch, and A. Mockus, "Developer reputation estimator (dre)," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1082–1085.

[17] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus, "World of code: An infrastructure for mining the universe of open source vcs data," in *IEEE Working Conference on Mining Software Repositories*, May 2019.

[18] Y. Ma, T. Dey, C. Bogart, S. Amreen, M. Valiev, A. Tutko, D. Kennard, R. Zaretzki, and A. Mockus, "World of code: Enabling a research workflow for mining and analyzing the universe of open source vcs data," *arXiv preprint arXiv:2010.16196*, 2020.

[19] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, 2014, pp. 1188–1196.

[20] T. Dey, A. Karnauch, and A. Mockus, "Replication Package for ICSE'21 paper - Representation of Developer Expertise in Open Source Software," Jan. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4457107

[21] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *ICSE '10: Proceedings of the 32th international conference on Software engineering*. ACM, 2010, pp. 385–394.

[22] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[23] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[24] V. Basili and R. Reiter, "An investigation of human factors in software development," *IEEE Computer*, vol. 12, no. 12, pp. 21–38, December 1979.

[25] A. J. Albrecht and J. R. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," *IEEE Trans. on Software Engineering*, vol. 9, no. 6, pp. 638–648, 1983.

[26] B. Curtis, "Substantiating programmer variability," in *Proceedings of the IEEE 69*, July 1981.

[27] B. Boehm, *Software Engineering Economics*. Prentice-Hall, 1981.

[28] C. Behrens, "Measuring the productivity of computer systems development activities with function points," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 648–652, November 1983.

[29] M. Zhou and A. Mockus, "Developer fluency: Achieving true mastery in software projects," in *ACM SIGSOFT / FSE*, Santa Fe, New Mexico, November 7–11 2010, pp. 137–146. [Online]. Available: http://dl.acm.org/authorize?309273

[30] J. Marlow, L. Dabbish, and J. Herbsleb, "Impression formation in online peer production: Activity traces and personal profiles in github," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. New York, NY, USA: Association for Computing Machinery, 2013, p. 117–128. [Online]. Available: https://doi.org/10.1145/2441776.2441792

[31] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. New York, NY, USA: Association for Computing Machinery, 2013, p. 103–116. [Online]. Available: https://doi.org/10.1145/2441776.2441791

[32] G. J. Greene and B. Fischer, "Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2016, p. 804–809. [Online]. Available: https://doi.org/10.1145/2970276.2970285

[33] Y. Wan, L. Chen, G. Xu, Z. Zhao, J. Tang, and J. Wu, "Scsminer: mining social coding sites for software developer recommendation with relevance propagation," *World Wide Web*, vol. 21, no. 6, pp. 1523–1543, 2018.

[34] C. Hauff and G. Gousios, "Matching github developer profiles to job advertisements," in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015, p. 362–366.

[35] T. Dey, J. L. Massengill, and A. Mockus, "Analysis of popularity of game mods: A case study," in *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*. ACM, 2016, pp. 133–139. [Online]. Available: https://dl.acm.org/doi/10.1145/2968120.2987724

[36] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," *arXiv preprint arXiv:1607.05368*, 2016.

[37] B. Theeten, F. Vandeputte, and T. Van Cutsem, "Import2vec: Learning embeddings for software libraries," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 18–28.

[38] E. Biswas, K. Vijay-Shanker, and L. Pollock, "Exploring word embedding techniques to improve sentiment analysis of software engineering

texts," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 68–78.

[39] S. Amreen, Y. Zang, C. Bogart, R. Zaretzki, and A. Mockus, "Alfaa: Active learning fingerprint based anti-aliasing for correcting developer identity errors in version control systems," *International Journal of Empirical Software Engineering*, 2019, accepted.

[40] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 3–14.

[41] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–37, 2018.

[42] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Suggesting accurate method and class names," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 38–49.

[43] A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, and A. Scott, "Sapfix: Automated end-to-end repair at scale," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 269–278.

[44] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[45] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 404–415.

[46] A. Mockus, D. Spinellis, Z. Kotti, and G. J. Dusing, "A complete set of related git repositories identified via community detection approaches based on shared commits," in *IEEE Working Conference on Mining Software Repositories: Data Showcase*, May 2020. [Online]. Available: https://arxiv.org/abs/2002.02707

[47] T. Fry, T. Dey, A. Karnauch, and A. Mockus, "A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 518–522. [Online]. Available: https://doi.org/10.1145/3379597.3387500

[48] T. Dey and A. Mockus, "A Dataset of Pull Requests and A Trained Random Forest Model for predicting Pull Request Acceptance," May 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3858046

[49] T. Dey, S. Mousavi, E. Ponce, T. Fry, B. Vasilescu, A. Filippova, and A. Mockus, "Detecting and characterizing bots that commit code," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 209–219. [Online]. Available: https://doi.org/10.1145/3379597.3387478

[50] T. Dey, B. Vasilescu, and A. Mockus, "An exploratory study of bot commits," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 61–65. [Online]. Available: https://doi.org/10.1145/3387940.3391502

[51] T. Dey, Y. Ma, and A. Mockus, "Patterns of effort contribution and demand and user classification based on participation patterns in npm ecosystem," in *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 36–45. [Online]. Available: https://doi.org/10.1145/3345629.3345634

[52] T. Dey and A. Mockus, "Are software dependency supply chain metrics useful in predicting change of popularity of npm packages?" in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 66–69. [Online]. Available: https://doi.org/10.1145/3273934.3273942

[53] S. Amreen, B. Bichescu, R. Bradley, T. Dey, Y. Ma, A. Mockus, S. Mousavi, and R. Zaretzki, *A Methodology for Measuring FLOSS Ecosystems*. Singapore: Springer Singapore, 2019, pp. 1–29. [Online]. Available: https://doi.org/10.1007/978-981-13-7099-1_1

[54] T. Dey and A. Mockus, "Modeling relationship between post-release faults and usage in mobile software," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, ser. PROMISE'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 56–65. [Online]. Available: https://doi.org/10.1145/3273934.3273941

[55] Y. Ma, "Constructing supply chains in open source software," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, pp. 458–459.

[56] T. Dey and A. Mockus, "Deriving a usage-independent software quality metric," *Empirical Software Engineering*, vol. 25, no. 2, pp. 1596–1641, Mar 2020. [Online]. Available: https://doi.org/10.1007/s10664-019-09791-w