Noname manuscript No.
(will be inserted by the editor)

ALFAA: Active Learning Fingerprint Based Anti-Aliasing for Correcting Developer Identity Errors in Version Control Systems

Sadika Amreen · Audris Mockus · Russell Zaretzki · Christopher Bogart · Yuxia Zhang

Received: date / Accepted: date

Abstract An accurate determination of developer identities is important for software engineering research and practice. Without it, even simple questions such as "how many developers does a project have?" cannot be answered. The commonly used version control data from Git is full of identity errors and the existing approaches to correct these errors are difficult to validate on large scale and cannot be easily improved. We, therefore, aim to develop a scalable, highly accurate, easy to use and easy to improve approach to correct software developer identity errors. We first amalgamate developer identities from version control systems in open source software repositories and investigate the nature and prevalence of these errors, design corrective algorithms, and estimate the impact of the errors on networks inferred from this data. We investigate these questions using a collection of over 1B Git commits with over 23M recorded author identities. By inspecting the author strings that occur most frequently, we group identity errors into categories. We then augment the author strings with three behavioral fingerprints: time-zone frequencies, the set of files modified, and a vector embedding of the commit messages. We create a manually validated set of identities for a subset of OpenStack developers using an active learning approach and use it to fit supervised learning models to predict the identities for the remaining author strings in OpenStack. We then compare these predictions with a competing commercially available effort and a leading research method. Finally, we compare network measures

Sadika Amreen, Audris Mockus, Russell Zaretzki
University of Tennessee, Tennessee, USA
E-mail: samreen@vols.utk.edu, audris@utk.edu, rzaretzk@utk.edu

Christopher Bogart
Carnegie Mellon University, Pennsylvania, USA
E-mail: cbogart@andrew.cmu.edu

Yuxia Zhang
Peking University, Beijing, China
E-mail: yuxiaz@pku.edu.cn

for file-induced author networks based on corrected and raw data. We find commits done from different environments, misspellings, organizational ids, default values, and anonymous IDs to be the major sources of errors. We also find supervised learning methods to reduce errors by several times in comparison to existing research and commercial methods and the active learning approach to be an effective way to create validated datasets. Results also indicate that correction of developer identity has a large impact on the inference of the social network. We believe that our proposed Active Learning Fingerprint Based Anti-Aliasing (ALFAA) approach will expedite research progress in the software engineering domain for applications that involve developer identities.

Keywords Software Repository Mining, Identity Disambiguation, Random Forest Classification, Record Linkage, Behavioral Fingerprinting, Social Network Analysis

1 INTRODUCTION

Software engineering, especially empirical software engineering, relies on various measures of software developer activity to fit models of developer productivity [27,12], project lead times [34], and code quality [31]. The recently popular software engineering domain of Mining Software Repositories (MSR) focuses on measuring software development based on data available in version control and issue tracking systems. The basic software engineering questions of developer productivity, project lead time, and the source code quality are investigated and modeled using measures of individual developers (such as project team size), developer actions (code changes and issues), level of developer experience, and interactions among developers. Complex industry tools are built based on data from version control systems [11,30].

All this research and tools assume and require an accurate determination of developer identity in order for the research results to be valid, models to be accurate, and industrial tools to work correctly. While mature software development organizations tend to keep good records of developer identity in their issue tracking and version control systems, this is not the case for open source projects or less mature software development groups. In fact, for outsiders, it is not even clear how many people participate in an open source project even though the project's version control system is public. Recent software engineering research heavily relies on plentiful data in open source projects. Unfortunately, the version control systems used in such research and tools do not represent developer identities accurately [4,16,29]. Errors include incorrect and missing values, such as multiple or erroneous spellings, identity changes that occur over time, group identities, and other issues Furthermore, even very small identity errors may strongly affect the downstream analysis [52].

The literature utilizing data from software repositories has to address this issue and includes topics spanning from developer collaboration [49,25], the contributions of companies to open source software projects [20,51], predicting faults in software [35], measuring developer productivity and expertise [7, 27], among numerous other examples. These issues have been recognized in software engineering [15,3] and beyond [10]. To cope, studies in the software engineering field tend to focus on individual projects or groups of projects where the number of IDs that need to be disambiguated is small enough for manual validation and devise a variety of heuristics to solve this formidable problem.

The existing approaches to correcting developer errors tend to be not scalable and is often time consuming. An important reason for that is the the lack of ground truth or the absence of validated identity corrections. This typically requires manual validation and an intensive iterative adjustment of heuristics used to correct the errors. This makes it impossible to correct millions of developer identities in billions of code commits in the open source ecosystem [24].

Another major problem with existing identity correction approaches is the lack of clarity of how to make these heuristics more accurate or easier to tailor to a specific dataset without an extensive amount of effort. This is partly due to the lack of clear understanding of what types of identity errors are common and why they are introduced. Finally, the identity correction approaches rely primarily on string similarity of the spelling of developer names and email. It would seem that traces of developer activities can also be used for identity resolution as, for example, gait can be used to identify a person.

We try to address these shortcomings, as the following research questions:

1. What are the most common reasons for identity errors in version control data?
2. Are there alternative measures to name and email similarity that can help correct identity errors?
3. Is it possible to design a scalable approach that can improve upon matching techniques used in research and commercial efforts within a software engineering domain?
4. What is the impact of identity errors on actual collaboration networks among developers?

In summary, we find several types of and reasons for identity errors (e.g.,synonyms and homonyms); we introduce innovative behavioral fingerprints in addition to traditional string matching techniques that improve the accuracy substantially; we introduce a supervised learning approach called ALFAA (Active Learning Fingerprint-based Anti-Aliasing) to identity matching in software engineering domain that is highly accurate and scalable, easy to apply, and can, in principle, increase in accuracy as additional training data are collected and utilized. Furthermore, we propose and demonstrate the use of active learning [38] to produce a highly accurate predictor with minimal effort spent on creating the training dataset.

We compare the accuracy of ALFAA on 16K OpenStack contributors to a commercially funded effort and to one of the recent research methods. We also demonstrate that it scales to a larger dataset of 2 million contributors to several large software ecosystems. Finally, we assess how identity errors affect file-induced developer collaboration networks [45]. We find that typos, application defaults, organizational IDs, and the desire for anonymity are the primary cause of errors in developer identity within a very large body of over 1 billion commits. The proposed behavioral fingerprints improve the accuracy of the predictor even with a limited training sample. We find that the commercial and recent research-based identity resolution methods for the OpenStack problem have much lower accuracy than our proposed method and that the errors in the actual identity data in OpenStack strongly impact the social network measures. The identity errors represent a real problem that is likely to affect results of many analysis or development tools, but these errors can be addressed even for very large datasets using the proposed approach.

The novelty of our contribution first involves behavioral fingerprinting that includes Doc2Vec method to find similarities among commit messages, thus providing authorship likelihood measures even for commits with empty or generic author string. Second, we propose the use of machine learning methods in identity resolution within software engineering context that improve accuracy to a level comparable or higher than manual matching. This is a radically different approach from the current state of the art of manually designed heuristics. The trained models can be further improved simply by adding larger training sample instead of requiring effort intensive design and application of customizable heuristics. Models and data will be shared upon publication. Third, we propose to use active learning to minimize effort to generate training samples. Fourth, we identify several new sources of errors in developer identity. Fifth, we evaluate accuracy of our approach on a large sample of 16K OpenStack contributors and compare it to a commercial method and a recent research method on an extremely large sample of 2M contributors in large ecosystems. We manually validate the performance and find ALFAA to have significantly lower lumping errors.

The remainder of this article is organized as follows. Section 2 discusses the current state-of-the-art practices in the domain of identity disambiguation. Section 3 discusses the data collection process and its overview. Section 4 discusses the nature of errors associated with developer identities as well as their reasons. Section 5 discusses the approach in solving the identity disambiguation problem by correcting synonym errors and reports the results we obtained. Section 6 compares the results produced by ALFAA to a commercial effort and recent research method. Section 7 demonstrates the impact of identity errors on networks by using a developer collaboration network and finally, Section 9 summarizes findings and provides conclusions. In addition, the following sections answer each of the research questions – Section 4 answers RQ1, Section 5 answers RQ2, Section 6 answers RQ3 and finally, Section 7 answers RQ4.

2 RELATED WORK

The issue of identity resolution through disambiguation or de-anonymization falls under the broader field of "Record Linkage". The first mathematical model for record linkage introduced in 1969 by Ivan Fellegi and Alan Sunter [13], is used to identify duplicates when unique identifiers are unavailable. This model serves as the basis of many record linkage methods practiced today.

2.1 Relevance of Identity Resolution

Identity resolution has been investigated in many fields such as on patent data [44] to link records of the companies, organizations and individuals or government agencies to which a patent is assigned, on US census data [48], synthetic census data [10] and in the construction of web services that integrate crowd-sourced data such as CiteSeer [22].

With the proliferation of online activities such as collaboration in software development, identity resolution techniques have also become important in the field of empirical software engineering research [15, 3] to disambiguate identities of people in a software ecosystem. Communication and coordination activities are central to development of large software projects. These activities logged on the mailing lists, issue trackers etc. are public for Open Source Software development and these serve as useful traces of communication and coordination between participants. These data can be mined for various purposes such as to build social diversity dataset from thousands of GitHub projects [43], to assess a contributor's total activity within projects [17] in Open Source Software and across platforms [50] and in mailing lists [47]. We discuss some of the applications as follows, outlined in existing research, where correct identity of developers is critical.

- Data Consolidation: when trying to combine information from different types of data sources in a coherent way where the available data concerning persons involved in a project may be dispersed across different repositories [18,37]. This can affect other studies that require consolidated statistics on users.
- Code reuse and attribution: A study [2] to understand code usage and attribution required survey of users on using multiple platforms such as StackOverflow and GitHub aimed to answer questions such as how often code is reused but not attributed. This required matching identities across platforms as users are likely to have various representation of their IDs across platforms. These kind of studies are required to address code maintenance and legal issues.
- Developer productivity measure: When a single individual uses multiple IDs, it becomes hard to track the work of individuals such as developers working on various projects on version control systems. This impacts productivity measures of developers by showing lower than actual productivity since a single developers activity may have been logged by more than a single ID, or alternatively, higher than actual productivity because multiple individuals have logged their activity using a single ID. It is important to understand the central/influential players in a network and resolving identities is key to its determination.
- Understanding social connected-ness and influence in developer communities: The information available in software repositories can be analyzed through a variety of statistical and social (graph-based) approach. One such application is in understanding social connected-ness in developer communities [42]. This kind of study helps to identify influential developers and projects through analyzing collaborations - developers connected through codes in version control systems, questions in mailing lists, bug reports and fixes in issue trackers etc. It may be of interest to identify influential developers because their activities can act as guides to other peoples projects. It may also help to identify the teachers in a developer community as active developers frequently answer questions as well [1].
- Assessing Contributions: Developers of many open source projects want to understand contributions from different companies (to ensure that each member company contributes its fair share). OpenStack even hired a com-

mercial firm to address this problem but, as we show in our analysis, they have not achieved very accurate results.

The issue of developer identities has been a serious problem in software repository mining. It remains a challenging issue due to a number of reasons, particularly, due the large volumes of poor quality data. This challenge is further enhanced by the fact that complicated, labor intensive evaluation techniques are required to validate methods of resolution as we will discuss in the following sections.

2.2 Existing Techniques of Identity Resolution

Approaches such as merging identities with similar name labels, email addresses or any combination of these have been used in the past for disambiguation. However, Most of these are still reliant on simple string matching heuristics. For example, an algorithm [3] designed specifically to detect identities belonging to developers who commit to code repositories and people participating in a mailing list uses string similarity based on Levenshtein distance on first, last, and user name fields of developers and mailers coupled with a threshold parameter. This assumes a name will be split into two parts using white-space or commas as delimiters and user names can be derived from the email address string. This algorithm was later modified to include more characters as separators, extended to account for an arbitrary number of name parts and include more individuals from bug repositories and then evaluated using different identity merge algorithms [18]. While these approaches are reported to perform well only through string matching and thresholding, for example, work using more sophisticated heuristics such as Latent Semantic Analysis (LSA) on names of GNOME Git authors which was also used for disambiguation [21], fail to address issues where developer identity strings are problematic, i.e. incomplete or missing.

Other research on the data from the U.S. patent and trademark (USPTO) [44] database uses a supervised learning approach based on a large set (over 150,000) of hand labeled inventor records to perform disambiguation. This, therefore, assumes an availability of sufficient and reliable ground truth data to perform a supervised learning approach. A major challenges we face with disambiguation is the lack of an adequate pool of hand labeled data to use for supervised learning. Furthermore, these prior approaches fail to address the problem of homonyms resolution i.e. where a single label may be used by multiple identities. This is critical because excluding problematic nodes from a network can radically alter the properties of the social network as well as nodes (e.g., developer productivity, tenure with the project, etc).

The fact that there is insufficient ground truth for our dataset of developers from projects hosted on GitHub causes a hindrance to employing any supervised learning approach directly. Past research on de-duplication of authors in citations [38] has leveraged a technique called active learning, which starts with limited labels and a large unlabeled pool of instances, thereby,

Table 1 Comparison of developer productivity with and without disambiguation of IDs

UserID	NumCmt	NumID	Prod(True)	Prod(Min)	Prod(Max)	chngMin(%)	chngMax(%)
User1	148	10	97.93	0.66	52.93	14,700	85
User2	23	4	20.09	2.62	9.60	666	109
User3	2,602	12	346.09	0.13	308.85	260,100	12
User4	1,057	5	219.87	0.20	200.73	105,600	9
User5	13,960	8	282.49	0.02	146.95	1,395,900	92
User6	2,743	5	198.89	0.43	145.52	45,616	36
User7	1,346	6	132.95	0.09	68.94	134,500	92
User8	121	2	37.15	4.29	32.85	764	13
User9	579	7	130.31	0.45	77.87	28,850	67
User10	398	10	77.73	0.19	34.57	39,700	124
User11	200	17	18.13	0.09	6.70	19,900	170
User12	54	3	10.59	0.78	8.43	1,250	25
User13	497	5	75.23	0.45	43.14	16,466	74
User14	914	6	116.56	0.38	46.16	30,366	152
User15	183	4	40.33	0.22	20.05	18,200	101
User16	172	3	52.09	0.90	45.73	5,633	13
User17	193	4	35.23	0.36	20.08	9,550	75
User18	184	7	41.60	0.22	32.56	18,300	27
User19	39	4	2.03	0.05	1.35	3,800	50
User20	179	3	45.00	0.25	33.68	17,800	33
User21	35	5	6.92	0.59	2.37	1,066	191
User22	3,504	3	556.91	0.79	545.62	69,980	2
User23	8,555	3	1284.79	160.09	894.47	702	43
User24	24	3	10.79	1.34	4.94	700	118
User25	39	2	10.81	3.88	6.92	178	56

significantly reducing the effort in providing training data manually. The active learning method uses an initial classifier to predict on some unlabeled instances. The initial classifier produces some results (a higher fraction) with high confidence and some others (a lower fraction) with lower confidence i.e. the classifier's confusion region. This confusion region can therefore be extracted and manually labeled for it to serve as the training data for the actual classifier.

In summary, the current state of art in software engineering remains based on designing a set of matching heuristics with manual verification. At the same time, techniques used in other fields need tailoring for the types of problems common in software engineering. We propose an approach that combines information from identity string and behavioral attributes and uses iterative supervised machine learning to achieve highly accurate identity resolution for synonym errors. As the training set for the learner increases, the approach should become even more accurate, since the proposed models can take advantage of the richer training data.

2.3 Illustration of impact of identity resolution on estimates of developer productivity

We used actual data for 25 open source developers who were selected using author's professional network. Colleagues were asked to use an online tool that constructs an activity profile from commits made in public git repositories. Colleagues that we knew are very active and others who we knew not to be very active in open source were included in this sample. Each of the 25 participants confirmed the disambiguation results produced by the method described in the paper. These 25 developers used from two to 17 (median of five) distinct

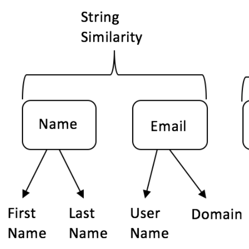
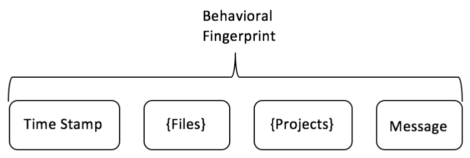



Fig. 1 Extracted Components of Commits that are relevant to this study

identities resulting in a total of 141 distinct identities. If we assume these developers to be working on the same project, that would lead to a dramatic (more than five times) overestimation of the number of developers needed to accomplish the tasks done by these 25 people. In Table 1, we consider how the lack of disambiguation on this sample of developers would affect measures of developer productivity. The first column shows anonymized user ID, the second and fourth columns show the number of commits and productivity (commits per year) made by that developer, and column three shows the number of distinct IDs (used by the developer) that were found in the open source version control systems. The next two columns indicate the minimum and maximum productivity of the IDs belonging to that developer. Instead of aggregating commits over all IDs belonging to the developer we calculate the number of commits for each ID separately (the numerator) and the time-span of these commits (the denominator). The last two columns shows by how many percent the actual productivity exceeds that the minimum and maximum productivity obtained when using a single ID belonging to the developer. As Table 1 shows, the lack of disambiguation would result in severe underestimation of developer productivity. This suggests that the disambiguation is essential in order to produce accurate measures pertaining to software developers.

3 DATA SOURCES

Version Control System (VCS) is an ubiquitous tool in software development and it tracks code modifications (commits). Each time a new commit is made, the VCS records authorship, commit time, commit message, parent commit and the full folder structure after the commit. Author string in a commit consists of the author name (first and last) and their email addresses. We determine files modified in a commit by comparing the full folder structure prior to and after the commit. We have been collecting such data from projects with public VCS since 2007 [26] and currently have 1.5B commits made by over 30M authors in over 60M VCS repositories. Figure 1 shows the information extracted from commits that were required for the studies discussed in this paper.

We set out to find a subset of this data that includes a sizable set of projects where we could compare the results not only to research-based methods but also to approaches used in industry. We, therefore, selected the OpenStack ecosystem as it already had an implementation of disambiguation by Bitergia, a commercial firm, which mapped multiple developer IDs to an unique identifier representing a single developer as well as mapping contributors to their affiliated companies.

OpenStack[1] is a set of software tools for building and managing cloud computing platforms for both public and private clouds. It lets users deploy virtual machines and can handle different tasks for managing a cloud environment on the fly[2]. We discovered 1,294 repositories that are currently hosted on GitHub and have 16,007 distinct author strings in the associated commits. Moreover, to measure the scalability of our method, we selected an even larger collection of projects from several large open source ecosystems having approximately 2M developer identities.

4 CLASSIFYING ERRORS

In order to tailor existing identity resolution approaches (or create new ones), we need a better understanding of the nature of the errors associated with the records related to developer identity. For example, in census data a common error may be a typo, a variation in the phonetic spelling of a name, or the reversal of the first and last names, among others. Previous studies have identified errors as a result of transliteration, punctuation, irrelevant information incorporated in names, etc. [21,9]. Furthermore, complications are at times introduced by the use of tools. Author information in a Git commit (which we study here) depends on an entry specifying user name and email in a Git configuration file of the specific computer a developer is using at the moment. Once Git commit is recorded, it is immutable like other Git objects, and cannot be changed. Once a developer pushes their commits from the local to remote repository, that author information remains. A developer may have multiple laptops, workstations, and work on various servers, and it is possible and, in fact, likely, that on at least one of these computers the Git configuration file has a different spelling of their name and email. It is not uncommon to see the commits done under an organizational alias, thus obscuring the identity of the author. Some Git clients may provide a default value for a developer, for example, the host name. Sometimes developers do not want their identities or their email address to be seen, resulting in intentionally anonymous name, such as, John Doe or email, such as devnull@localhost. Developers may change their name over time, for example, after marriage, creating a synonym and other scenarios may be possible.

In order to correct this, we need to determine the common reasons causing errors to be injected into the system. We therefore, inspected authors strings

[1] https://www.openstack.org/

[2] https://opensource.com/resources/what-is-openstack

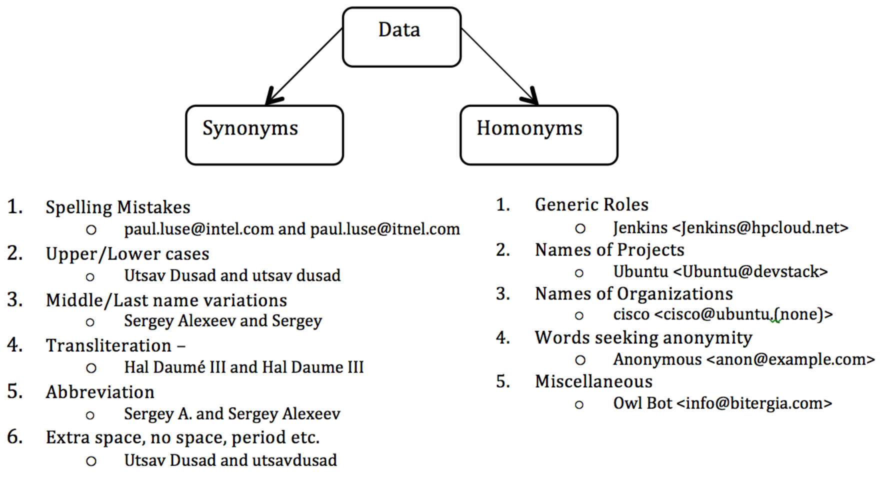

Data
Synonyms
Homonyms
1. Spelling Mistakes
paul.luse@intel.com and paul.luse@itnel.com
2. Upper/Lower cases
Utsav Dusad and utsav dusad
3. Middle/Last name variations
Sergey Alexeev and Sergey
4. Transliteration –
Hal Daumé III and Hal Daume III
5. Abbreviation
Sergey A. and Sergey Alexeev
6. Extra space, no space, period etc.
Utsav Dusad and utsavdusad
1. Generic Roles
Jenkins <Jenkins@hpcloud.net>
2. Names of Projects
Ubuntu <Ubuntu@devstack>
3. Names of Organizations
cisco <cisco@ubuntu.(none)>
4. Words seeking anonymity
Anonymous <anon@example.com>
5. Miscellaneous
Owl Bot <info@bitergia.com>

Fig. 2 Types of Synonym and Homonym Errors Discovered through Card-Sort

from our collection (at the time there was approximately 1B commits). The procedure used to determine the types of identity errors was as follows. First, we inspected random subsets of author IDs to understand how or why these errors occur. We then inspected the most common names and user names. The reviewer was tasked with identifying anything unusual in the name or email. Third, we also consider errors encountered in the manual labeling effort during the active learning phase as discussed in Section 5. The resulting anomalies were then grouped using the open card sort method [41]. This is a technique of organizing information in which a person, given a set of cards (i.e. IDs in this case) classifies them into any number categories named and created by the person. In our case, this was done by two people. Each person read the card (i.e. the IDs) and put them in separate bins representing a type of error. Initially, one of the persons, found three categories of errors (synonym, homonym and missing data) while the other found two (synonyms and homonyms). Upon careful inspection of the three categories and items belonging to the missing data bucket, missing data was merged into the homonym category because it was equivalent to a homonym represented by an empty string. Using this approach we found that the errors resulted in two primary categories: synonym and homonym errors.

– **Synonyms:** Synonyms errors are introduced through spelling mistakes, capitalization (or absence) of names, introduction of a middle name, last name change due to marriage, abbreviation of a name, adding extra space(s), adding period, reversal of first and last names, transliteration of non-ascii characters, irrelevant information incorporated into names. These errors can arise from one or a combination of the above cases and are introduced when a person uses different strings for names, user-names or email addresses. For example, 'utsav dusad <utsavdusad @gmail.com>' and 'utsavdusad <utsavdusad@gmail.com>' are identified as synonyms. Spelling mistakes such as 'Paul Luse <paul.e.luse @intel.com>' and 'paul luse <paul.e.luse@itnel.com>' are also classified as synonyms, as 'itnel' is likely to be a misspelling of 'intel'.

– **Homonyms:** Homonym errors are introduced when an individual provides IDs that cannot be tied to a single individual. For example, these may be identifications related to generic roles ('Admin', 'root', 'dev'), names of projects ('Jenkins', 'Travis CI', 'Ubuntu', 'Openstack', 'Vagrant'), names of organizations ('cisco', 'cmart', 'walmart'). The ID may also be any string that seeks to preserve anonymity or are simply placeholders injected by tools ('nobody', 'your name', 'test', 'anonymous', 'me', 'John Doe'). Other examples include miscellaneous terms such as 'Bot', 'EC2 Users', 'Server' etc. For example, the ID 'saper <saper@saper.info>' may be used by multiple entities in the organization. For example 'Marcin Cieslak <saper@saper.info>' is an entity who may have committed under the above organizational alias. Homonym errors are also introduced when a user leaves the name or email field empty, for example, 'chrisw <unknown>'. A brief frequency analysis showed that the most frequent names in the dataset such as 'nobody', 'root', and 'Administrator' are a result of homonym errors as shown in Table 2

Table 2 Data Overview: The 10 most frequent names and emails

Name	Count	First Name	Count	Last Name	Count	Email	Count	User Name	Count
unknown	140859	unknown	140875	unknown	140865	<blank>	16752	root	72655
root	66905	root	66995	root	67004	none@none	9576	nobody	35574
nobody	35141	David	45091	nobody	35141	devnull@localhost	8108	github	19778
Ubuntu	18431	Michael	40199	Ubuntu	18560	student@epicodus.com	5914	ubuntu	18683
(no author)	6934	nobody	35142	Lee	10826	unknown	3518	info	18634
nodemcu-custom-build	6073	Daniel	34889	Wang	10641	you@example.com	2596	<blank>	17826
Alex	5602	Chris	29167	Chen	9792	anybody@emacswiki.org	2518	me	14312
System Administrator	4216	Alex	28410	Smith	9722	=	1371	admin	12612
Administrator	4198	Andrew	26016	Administrator	8668	Unknown	1245	mail	11253
<blank>	4185	John	25882	User	8622	noreply	913	none	11004

The findings for RQ1 are, therefore, shown in Figure 2, with the various subcategories and corresponding examples for the two broad categories (synonyms and homonyms) we found the errors to belong to. We would like to note that in the remainder of this paper we focus mainly on synonym resolution. While our method, especially the fingerprinting part, is suitable for homonym resolution, the process requires a different experimental setup (assigning authors to commit, not disambiguating author identities) that are beyond the scope of this work.

5 DISAMBIGUATION APPROACH

Following traditional record linkage methodology and identity linking in software [3] we first split the information in the developer ID into several fields and define similarity metrics for all author pairs. We also incorporate the term frequency-adjustment measure for each of the attributes in a pair. Finally, we add similarity between behavioral fingerprints. We generate a table of these similarity measures for all 256,224,049 author pairs generated from 16,007 developer IDs in the OpenStack dataset.

5.1 String Similarity Measures

Each author string is stored in the following format - “name <email>”, e.g. “Hong Hui Xiao <xiaohhui@cn.ibm.com>”. We define the following attributes for each user.

1. Author: String as extracted from source as shown in the example above
2. Name: String up to the space before the first ‘<’
3. Email: String within the ‘<>’ brackets
4. First name: String up to the first space, ‘+’, ‘-’, ‘_’, ‘,’, ‘.’ and camel case encountered in the name field
5. Last name: String after the last space, ‘+’, ‘-’, ‘_’, ‘,’, ‘.’ and camel case encountered in the name field
6. User name: String up to the ‘@’ character in the email field

Additionally, we introduce a field ‘inverse first name’ whereby in the comparison between two authors it is compared to the last name in the other record. We introduce this field to make sure that our algorithm captures cases where authors reverse the order of their first and last names. In the case where there is a string without any delimiting character in the name field, the first name and last name are replicated. For example, bharaththiruveedula <bharath_ves@hotmail.com>would have ‘bharaththiruveedula’ replicated in the first, last and the name field. We calculate both Levenshtein and the Jaro-Winkler similarity as we have seen in previous studies [3,21], which are standard measures for string similarity, for each author pair. To do this, we use an existing implementation of the measures in the RecordLinkage [39] package in R, namely the levenshteinSim() and jarowinkler() functions. In a preliminary investigation, we found that the Jaro-Winkler similarity produces better scores which are more reflective of similarity between author strings than the Levenshtein score and, therefore, use this measure in the proposed method. The Jaro Similarity is defined as

$$sim_j = \begin{cases} 0, & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

where s_i is the length of string i, m is the number of matching characters and t is half the number of transpositions. The Jaro-Winkler Similarity modified the Jaro similarity so that differences at the beginning of the string has more significance than differences at the end. It is defined as

$$sim_w = sim_j + lp(1 - sim_j)$$

where l is the length of a common prefix at the start of the string up to a maximum of four characters and p ($<= 0.25$) is a scaling factor for how much the score is adjusted upwards for having common prefixes.

5.2 Frequency Adjustment Score

If two developer IDs share an uncommon name, it provides greater confidence than the IDs that share a more common name such as "John". Denote $N('John')$ as the number of developers using this first name. If each developer produced the same number of commits, then the conditional probability that one specific developer was an author of a randomly selected commit (having author first name 'John') would be $\frac{1}{N(John)}$. This demonstrates that common names do not provide strong evidence of a specific developer identity, while the rare names do. In the extreme case, if there is only one developer with a particular name, all commits containing that name should come from the same developer (the conditional probability $\frac{1}{N(UniqueName)} = \frac{1}{1}$ would be 1). Furthermore, certain names like "nobody" or "root" do not carry any information about the authorship and should be disregarded in the similarity detection. This extra information, if properly encoded, could be exploited by a machine learning algorithm making disambiguation decisions. We, therefore, count the number of occurrences of the attributes for each author as defined in Section 5.1 i.e. name, first name, last name, user name and email for our dataset. We provide this information to the machine learning algorithm as a separate variable we refer to as frequency adjustment score. We could provide two variables: an indicator variable that takes a value of one when the name is fictitious and zero otherwise and another variable that contains the absolute frequency of the specific attribute (.i.e., $\frac{1}{N(John)}$). This approach would, unfortunately, double the number of predictors and, in turn, slow the fitting procedure and increase the chances of overfitting the data. To avoid that problem, we chose to combine these two variables into a single predictor. Specifically, this variable depends on the absolute frequency of the string in the corpus. The more frequent the string is, the smaller this score is. The algorithm can then learn that for the perfectly matching names that are highly frequent the match does not provide much evidence that they belong to the same person. Furthermore, we set this variable for a pair of name, a first name, a last name, or a user name to $exp(-10)$ if at least one element of the pair belongs a string identified as potentially fictitious. $exp(-10)$ was chosen because we found the value to be much smaller than that for the most frequent non-fictitious names. We further took the logarithm of the frequency adjustment score because the resulting scores were distributed highly unevenly. In such cases a logarithm is useful is when we are discussing measurements with a different orders of magnitude i.e. 100 vs 1,000 vs 1,000,000. The logarithm smooths our measurements so that it is easier for a user (or, in our case, an algorithm) to distinguish between significant scores, similar to the Richter scale in earthquake dynamics where each unit corresponds to a ten-fold increase in energy. Its more user friendly to score things -7 versus -8 as opposed to 10ˆ-7 vs 10ˆ-8.

Specifically, we calculate the frequency adjustment score between author pairs, authors a_1 and a_2, for each of these attributes as follows:

$$f_{freq} = \begin{cases} \log_{10} \dfrac{1}{f_{a_1} \times f_{a_2}} & \text{if } a_1 \text{ and } a_2 \text{ are non-fictitious} \\ -10 & \text{otherwise} \end{cases}$$

where f_{a1} and f_{a2} are the absolute frequency of names of authors a_1 and a_2 respectively. We generate a list of 200 common strings of names, first names, last names and user names and emails from the full data set of authors (the first 10 shown in Table 2) and manually remove names that appear to be non-fictitious, i.e. names that could truly belong to a person such as Lee, Chen, Chris, Daniel etc. This gives us a compilation of a set of fictitious names.

5.3 Behavioral Fingerprints

In addition to the spelling of the name and contact information, developers might leave their individual signatures in the version control systems. For example, the way commit messages are composed, the set of files are modified, or the time zones of the commits, may all contain information that can be used to identify an individual. To capture such traces of developer actions, we designed three similarity measures - (1) Similarity based on files modified — two author IDs modifying similar sets of files are more likely to represent the same person. (2) Similarity based on time zone — two author IDs committing in the same time zone indicate geographic proximity and, therefore, have higher likelihood of being the same individual. (3) Similarity based on commit message text — two author IDs sharing writing style and vocabulary increase chances that they represent the same entity. Operationalizations of these behavioral fingerprints are given below.

5.3.1 ***Files modified***

Each modified file is inversely weighted using the number of distinct authors who have modified it (for the similar reasons common names are down-weighted as evidence of identity). The pairwise similarity between authors, a_1 and a_2, is derived by adding the weights of the files, $\mathrm{W_f}$, touched by both authors. A similar metric was found to work well for finding instances of succession (when one developer takes over the work of another developer) [28]. The weight of a file is defined as follows where $\mathrm{A_f}$ is a set of authors who has modified file *f*.

$$W_f = \frac{1}{A_f}, \text{where } A_f = |a_1^f, ..., a_n^f|$$

$$Sim_{a_1a_2} = \sum_{i=1}^{n_{a_1a_2}} W_{f_i}, \text{where } n_{a_1a_2} = |f_{a_1} \cap f_{a_2}|$$

5.3.2 ***Time zone***

We discovered 300 distinct time zone strings (due to misspellings) from the commits and created a 'author by time zone' matrix that had the count of commits by an author at a given time-zone. All time zones that had less than 2 entries (authors) were eliminated from further study. Each author was therefore assigned a normalized time-zone vector (with 139 distinct time zones) that represents the pattern of his/her commits. Similar to the previous metric, we weighted each time zone by the inverse number of authors who committed at least once in that time-zone. We multiply each author's time zone vector by the weight of the time zone. We define author $\mathrm{a_i}$'s time-zone vector as:

$$(TZV^t_{a_i}) = \left(\frac{C^t_{a_i}}{A_t}\right),$$

Here, $(C^t_{a_i})$ is the vector representing the commits of an author a_i in the different time zones t and (A_t) is the vector representing the number of authors in the different time zones. The pairwise similarity metric between author $\mathrm{a_1}$ and author $\mathrm{a_2}$ is calculated using the cosine similarity as:

$$tzd_{a_1a_2} = cos\_sim(TZV_{a_1}, TZVa_2)$$

where $\mathrm{TZV_{a1}}$ and $\mathrm{TZV_{a2}}$ are the authors' respective vectors.

5.3.3 ***Text similarity***

We use the Gensim's implementation [3] of the Doc2Vec [23] algorithm to generate vectors that embed the semantics and style of the commit messages of each author. All commit messages for each individual who contributed at least once to one of the OpenStack projects were gathered from the collection described above and a Doc2Vec model was built. DocVec, unlike Word2Vec, allows to embed (estimate vectors) not only for each word in the text, but for the document descriptors (tags or author IDs in our case) as well. We used distributed memory version of paragraph vector with the vector size 200, short window of three (since commit messages are quite short), negative sampling of 20, hierarchical sampling, and removed words that were present in fewer than 15 commit messages. The 200-dimensional vector was short enough for quick computation, yet it was long enough to encapsulate the variation of information in the commit messages [4]. Our earlier experiments on commit messages found this set of parameters to yield satisfactory results (of 80% accuracy for top on a sample of 600 commit messages done by 21 developers). The resulting

[3] https://radimrehurek.com/gensim/index.html

[4] On large and diverse bodies of text, a larger vector size of 300 is recommended [36]

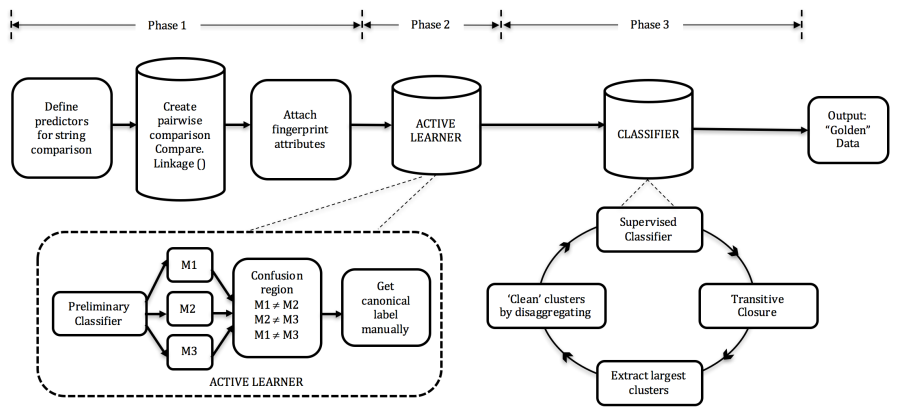

Phase 1
Phase 2
Phase 3
Define predictors for string comparison
Create pairwise comparison Compare. Linkage ()
Attach fingerprint attributes
ACTIVE LEARNER
CLASSIFIER
Output: "Golden" Data
Preliminary Classifier
M1
M2
M3
Confusion region M1 ≠ M2 M2 ≠ M3 M1 ≠ M3
Get canonical label manually
ACTIVE LEARNER
Supervised Classifier
Transitive Closure
Extract largest clusters
'Clean' clusters by disaggregating

Fig. 3 Concept of the Disambiguation Process

vectors for each of the 16,007 authors were used to calculate pairwise cosine similarity between authors.

However, there are several potential drawbacks of these distance metrics. For example, high scores for files touched may mean that two different individuals are working on the same project thereby editing the same files at alternating times. High document similarity may mean that the authors share similar vocabulary in the commit messages which may also be influenced by the work on the same project. Fortunately, the machine learning algorithm would discover these patterns from the training data and would find combinations of the similarity scores (that may be either high or low), that correspond to true matches. As we show later in the results, the behavioral similarity measures are important predictors for disambiguation.

5.4 Data Correction

The data correction process can be divided into 3 phases as shown in Figure 3.

1. Define predictors - Compute string similarity, frequency adjustment score, and behavioral similarity
2. Active learning - Use a preliminary classifier to extract a small set from the large collection of data and generate labels for further classification.
3. Classification - Perform supervised classification, transitive closure, extract clusters to correct, and dis-aggregate incorrectly clustered IDs.

5.4.1 ***PHASE 1: Define Predictors for the Learner***

Once we have defined the attributes (name, first name, last name, email, username) for which we want to calculate string similarity, we use relevant

Table 3 Confusion region from the preliminary classifier

Model1	Model2	Model3
Link	Link	No-Link
Link	No-Link	Link
No-Link	Link	Link
No-Link	No-Link	Link
No-Link	Link	No-Link
Link	No-Link	No-Link

functions implemented in the RecordLinkage library [39] to obtain the Jaro-Winkler similarity between each pair of attributes: authors' name, first name, last name, user name, email and the first author's first name to the 2nd author's last name (we refer to this as the inverse first name). In addition to the string similarities based on these fields, we also include the term frequency adjustment score, as is commonly done in record matching literature. The highly frequent values tend to carry less discriminative power than infrequent email addresses or names. Finally, we include three fingerprint metrics — files touched, time-zone, and commit log text. The resulting data is used as an input to the next phase, i.e. the active learning process.

5.4.2 ***PHASE 2: Active Learning***

Supervised classification requires ground truth data and manual classification is time consuming. It is also error-prone - we found some differences between two raters and further errors discovered in the Active Learning phase where the learner indicated manual classification error that was verified by both raters. Since manual classification of all possible pairs is impossible (it would require roughly 100 people each working 9000 hours to hand label the Openstack collection we have now), it is necessary to identify a small subset of instances so that the classifier would produce accurate results on the remainder of the data. Selecting a random sample of pairs to compare for manual labeling is not likely to work either: in our case, the chance that 2000 randomly selected pairs from 256M author pairs would belong to the same person is close to zero (assuming, on average, two developer IDs per person, 2000 pairs would represent 10^{-5} fraction of the entire sample). *Active Learning* [38] is an idea that can be use to minimize manual classification effort. In a nutshell, a variation in predictions of a preliminary classifier fitted on different subsets of the classified data are manually resolved. It helps ensure that the pairs that are most likely to be confused by the learner are added to the training data. The expensive manually classified training data, therefore, is only collected where the initial classifier is not consistent. As found in other work [38], we also discovered that this approach achieves very high accuracy with relatively few manually classified pairs.

Active Learning Design Details

In order to seed the training set for the active learning procedure, we have to resolve the problem of how to select a set of initial pairs for manual labeling. If we randomly select authors, the fraction of matches obtained will be very low as described above. To increase the proportion of matches in this seed data, we identify non-homonym developer IDs where there were at least two distinct emails for the same name (first and last) or where there were at least two distinct names for the same email. We then sample 2,825 pairs from this set, so that either for each pair either name or email is the same. This number was obtained by calculating the number of pairs we can manually validate in a one week sprint working 2 hours per day. These 2,825 pairs were manually labeled as a match (1) or a non-match (0). We found 2,016 matches and 809 non-matches in this set. We also calculated the string similarity and behavioral similarity for each pair in this set. We then randomly partitioned this data into ten parts and fit three classifiers on nine parts of the data (each of these 9 parts had overlapping and different observations). Typical active learning approach would then use these classifiers to predict matches on the data outside these pairs. However, since manual labels may be prone to errors, we add an additional step of using these three models to predict matches on the data that has already been manually labeled. Each classifier was used to predict outcomes for the all 2,825 pairs. As expected, the three classifiers trained on different training subsets yielded slightly different predictions. There were 2,345 pairs where all three learners did not agree (i.e at least one learner had a prediction different from the other two). This is the confusion region of the learner, as shown in Table 3. The predictions from the learners contained some instances where the manual labels were incorrectly assigned. We made appropriate correction in the manually classified data and the resulting set was used as the training data for the next iteration of the active learner.

We used all 16 attributes (name, email, first name, last name, user name, inverse first name, name frequency adjustment, email frequency adjustment, last name frequency adjustment, first name frequency adjustment, user name frequency adjustment, files touched, time-zone, and text similarity) in the initial Random Forest model. We obtained the importance of each predictor in this initial model and dropped the attributes with low importance from all subsequent models.

5.4.3 ***PHASE 3: Classification***

Once the labeled data set is created, we use it to train random forest models and perform a 10-fold cross validation with results shown in Table 4. The classifier-predicted pairwise matches are completed via transitive closure to obtain the final predictor of identity matches [5]. The result of the transitive closure is a set of connected components with each cluster representing a single author. Once the clusters are obtained, we consider all clusters containing 10 or

[5] We found that more accurate predictors can be obtained by training the learner only on the matched pairs, since the transitive closure typically results in some pairs that are extremely dissimilar, leading the learner to learn from such pairs and, subsequently, produce many more false positives

Table 4 Confusion Matrix of 10-fold cross validation of the Random Forest Model: 1 represents a match while 0 represents a non-match

	0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
0	549,609	4	548,179	3	549,469	2	551,136	5	550,108	5
1	0	992	2	1,110	0	1,082	0	1,039	3	1,014
	0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
0	549,204	1	549,402	1	547,958	4	548,730	4	549,569	2
1	2	1,075	1	1,033	0	1,021	0	1,084	0	1,010

more elements to investigate if multiple developers may have been grouped into a single component. The resulting 20 clusters - 44 elements in the largest and 10 elements in the smallest cluster among these, were then manually inspected and grouped. This manual effort included the assessment of name, user name and email similarity, projects they worked on, as well as looking up individual's profiles online where names/emails were not sufficient to assign them to a cluster with adequate confidence. An example of cluster reassignment is given in Table 5 where we dis-aggregated a single large cluster of 11 IDs to 3 smaller clusters. The first column is the author ID, the second is the cluster number the ID was assigned to by the algorithm, the third column is the manually assigned cluster number after dis-aggregation. We noticed that, the largest cluster of size of 44 in fact was based on homonym 'root'. Therefore, we dis-aggregated the entire cluster to form 44 single-element clusters. The output of this phase is a cleaned data set in which we have corrected synonym errors via machine learning and fixed some of the homonym errors by inspecting the largest clusters. We use the corrected set as a reference or 'golden' data set representing developer identities in the further analysis of OpenStack data.

5.5 Results

We evaluate the models using the standard measure of correctness - precision and recall - using the true positive (*tp*), true negative (*tn*), false positive (*fp*) and false negative (*fn*) outcomes produced by the models. An outcome is true positive when the model predicts a match correctly. This means the model predicted a match between an ID pair, when the ID pair is truly a match. An outcome is true negative when the model predicts a non-match correctly. This means the model predicted a non-match between an ID pair, when the ID pair is truly a non-match. An outcome is false positive when the model predicts a match incorrectly. This means the model predicted a match between an ID pair, when the ID pair is truly a non-match. Finally, an outcome is false negative when the model predicts a non-match incorrectly. This means the model predicted a non-match between an ID pair, when the ID pair is truly a match. We obtained an average precision of 99.9% and an average recall of 99.7% from the 10-fold cross validation of the random forest model shown in Section 4.

Table 5 Cluster Cleanup through Manual Disaggregation

Author Identity	**Cluster#**	**New Cluster#**
AD <adidenko@mirantis.com>	22	1
Aleksandr Didenko <adidenko@mirantis.com>	22	1
Alexander Didenko <adidenko@mirantis.com>	22	1
Sergey Vasilenko <stalker@makeworld.ru>	22	2
Sergey Vasilenko <sv854h@att.com>	22	2
Sergey Vasilenko <sv@makeworld.ru>	22	2
Sergey Vasilenko <svasilenko@mirantis.com>	22	2
Sergey Vasilenko <xenolog@users.noreply.github.com>	22	2
Vasyl Saienko <vsaienko@mirantis.com>	22	3
vsaienko <vsaienko@cz5578.bud.mirantis.net>	22	3
vsaienko <vsaienko@mirantis.com>	22	3

$$Precision = \frac{tp}{tp + fp}, Recall = \frac{tp}{tp + fn}$$

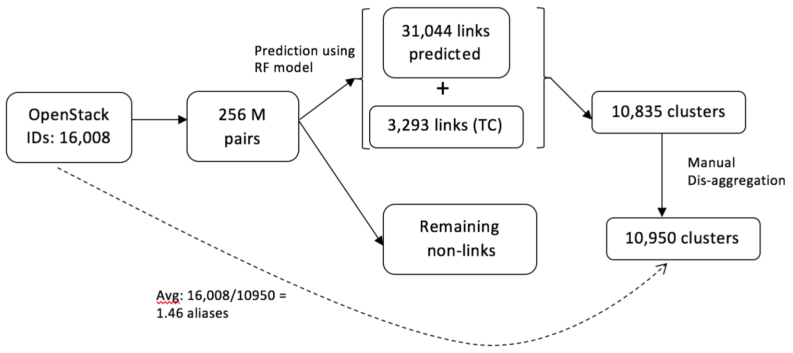
Since record matching is a slightly different problem from traditional classification, previous work done on identity matching introduces two additional error metrics: splitting and lumping [40] as they are easier to interpret than standard precision and recall for the domain of identity matching. Lumping occurs when multiple developer IDs are identified to belong to a single developer. The number of lumped records is defined as the number of records that the disambiguation algorithm incorrectly mapped to the largest pool of IDs belonging to a given developer. Splitting (i.e. 1 - Recall) occurs when an ID belonging to a single developer is incorrectly split into IDs representing several developers. The number of split records is defined as the number of developer IDs that the disambiguation algorithm fails to map to the largest pool of IDs belonging to a given developer.

Table 6 Largest cluster Corresponding to Single Entity with Highest Aliases After Disaggregation

AuthorID	**AuthorID**
Greg Holt <gholt@rackspace.com>	tlohg <z-github@brim.net>
Greg Holt <greg@brim.net>	tlohg <gholt@rackspace.com>
Greg Holt <gregory.holt@gmail.com>	gholt <z-launchpad@brim.net>
Greg Holt <gregory˙holt@icloud.com>	gholt <z-github@brim.net>
Greg Holt <z-github@brim.net>	gholt <gregory.holt+launchpad.net@gmail.com>
Gregory Holt <gholt@racklabs.com>	gholt <gholt@rackspace.com>
gholt <devnull@brim.net>	gholt <gholt@brim.net>

These two metrics only focus on the largest pool of IDs belonging to a single developer and ignores the other clusters of IDs corresponding to the same unique developer. To address that the work in [44] modifies these measures to evaluate all pairwise comparison of author records made by the disambiguation

Fig. 4 Results from disambiguation from OpenStack developers



algorithm. We follow the latter approach and create a confusion matrix of the pairwise links both from the golden data set and from the links created by the classifier:

$$Splitting = \frac{fn}{tp + fn}, Lumping = \frac{fp}{tp + fn}$$

The cross-validation shows that 0.3% of the cases were split and 0.1% of the cases were lumped. We use one of these models to predict links or non-links for our entire dataset of over 256M pairs of records. The classifier found 31,044 links and we generated an additional 3,293 links through transitive closure. Therefore, we have 34,337 pairs linked after running the disambiguation algorithm. Using this, we constructed a network that had 10,835 clusters that were later manually inspected and disaggregated using the procedure described in subsection 5.4.3. Finally, we were left with 10,950 clusters, each representing an author, with 14 elements in the largest cluster, corresponding to the highest number of aliases by a single individual as shown in Table 6. The results extracted in each phase is illustrated in Figure 4.

The results in Table 7 report the prediction performance where the prediction model was fit with and without the behavioral metrics. The table allows us to answer RQ2 (do behavioral metrics increase accuracy). We obtained seven times higher precision error that increases from 0.1% to 0.7% and recall error increases 1.7 times from 0.3% to 0.5% when behavioral metrics are dropped from the models.

We would like to stress that the tolerance for identity errors is extremely low. For example in [52], only 6% identity errors lead to about 20% error in counting the number of developers and about 52% error in calculating the time of stay of developers within a repository. As observed in [52], developers who are more active are much more likely to have identity errors. As a result, even small errors may get massively magnified. For example, in absolute terms, matching 256M pairs for the OpenStack data (of about 16,000 authors), 0.1% error would mean 256,000 mis-classified links that may also be magnified by

Table 7 Comparison of the Results with and without behavioral fingerprinting

Metric	Without Behavioral Fingerprint (%)	With Behavioral Fingerprint (%)
Precision	99.3	99.9
Recall	99.5	99.7
Splitting	0.4	0.3
Lumping	0.6	0.1

a transitive closure of the the matched pairs. Therefore, one mis-classification can easily cascade into much bigger problems. For research that relies on such data, this poses a difficult problem. While the empirical study we conducted did not reach the ideal level of zero errors, we expect that the proposed algorithm, after being trained with sufficient amount of data, could potentially reach this highly desirable threshold. A measure of the importance of the features used in the classification shows that two out of the three behavioral fingerprints were the second and third most important variables. The most important feature was the name of the user. We look at the mean decrease in accuracy when each of the features are dropped from the classification process, and removing the behavioral fingerprints had the largest impact after removing the name feature. Mean decrease in accuracy is impacted by the features in the following order (high to low): name, time-zone, files-modified, email, first-name frequency adjustment, inverse first name, last-name frequency adjustment, username, text-similarity, last-name and firstname.

An empirical study quantifying the impact is discussed in Section 7. In summary, the answer to RQ2 is positive: the behavioral fingerprints increase the accuracy of synonym resolution.

6 EVALUATION

In this section we try to answer questions related to the accuracy of the manually labeled training data and to compare our approach to two alternatives from the commercial and research domains. It is important to note that we are evaluating our algorithm trained on a small amount of training data and, as with other machine learning techniques, we expect it to have higher accuracy with more training data that would be added in the future.

6.1 Accuracy of the training data

The absence of ground truth requires us to investigate the accuracy of the training data. Two independent human raters (authors who are PhD students in Computer Science) were presented with the spreadsheet containing 1,060 pairs of OpenStack author IDs and marked it using the following protocol. Each rater was instructed to inspect each pair of author IDs (full name and email) listed in the spreadsheet and supplemented by author's affiliations (see Section 6.2, the dates of their first and last commits in the OpenStack projects, and their behavioral similarity scores. Each rater was instructed to mark author pair as a match (1) if the two identities are almost certainly from the same person, a non-match (0) if they are certainly not from the same person, and provide a number in between zero and one reflecting the raters subjective probability that they are representing the same person. Each rater was instructed to use the above mentioned information (listed next to the pair in the spreadsheet) to make their decision and were instructed to search for developers on Github or Google if they did not feel confident about their decision. For cases where both raters marked either zero or one we found 1,011 instances of agreement and 17 cases of disagreement between the two raters. By thresholding the 32 cases that had probability value greater than zero and less than one to the nearest whole, we obtained 1,042 instances of agreement (98.3%) and 18 cases of disagreement (1.69%).

We also compare the ten-fold cross-validation predictions described above with the second rater (whose input was not used for training). The numbers of disagreements between the second rater and the predictions over ten folds ranged from 11 (1.03%) to 18 (1.69%) (a mean of 15.18). The second rater had, therefore, similar or better agreement with the prediction than with the first rater.

We thus have established the degree to which the two raters agree on the decision, but not necessarily that either of the raters was correct. To validate rater's opinions we, therefore, administered a survey to a randomly selected set of authors. The survey provided respondents with a set of commits with distinct author strings. All commits, however, were predicted to have been done by the respondent and each respondent was asked to indicate which of the commits were the ones made by them. From a randomly selected 400 developers sixty-nine emails bounced due to the delivery problems. After 20 days we obtained 45 valid responses, resulting in a response rate of 13%. No respondents indicated that commits predicted to be theirs were not submitted by them, for an error rate of 0 out of 45. This allows us to obtain the bound on the magnitude of error. For example, if the algorithm has the error rate of 5%, then we would have less than one in ten chances to observe 0 out of 45 observation to have errors[6].

After establishing high accuracy of the training data we proceed to compare our approach to an approach that was implemented by professional commercial effort.

6.2 Comparison with a commercial effort

Openstack is developed by a group of companies, resulting in an individual and collective interest in auditing the development contribution of each firm working on Openstack. This task was outsourced to Bitergia[7], which is a company dedicated to performing software analytics. We collected the disambiguation

[6] Assuming independence of observations and using binomial distribution.

[7] https://bitergia.com/

data on OpenStack authors produced by Bitergia. The data was in a form of a relational (mysql) database that had a tuple with each commit sha1 and developer ID and another table that mapped developer ID (internal to that database) to developer name (as found in a commit). The Bitergia data had only 10,344 unique author IDs that were mapped to 8,840 authors (internal database IDs). We first restricted the set of commits in our dataset to the set of commits that were in the Bitergia database and selected the relevant subset of authors (10,344 unique developer IDs) from our data for comparison to ensure that we are doing the comparison on exactly the same set of authors. Bitergia algorithm misses 17,587 matches predicted by our algorithm and introduced six matches that our algorithm does not predict. In fact, it only detected 1,504 matches of over 22K matches (under 7%) predicted by our algorithm. Bitergia matching predicted 8,840 distinct authors or 41% more than our algorithm which estimated 6,271 distinct authors. As shown in Table 8, it has almost 50 times higher splitting error than manual classification, though it almost never lumps two distinct authors. We, therefore, conclude that the prediction done by the commercial effort has substantially lower accuracy.

6.3 Comparison with a research study

Next, we compare our method to a recent research method[8] that was applied on data from 23,493 projects [43] from GHTorrent to study social diversity in software teams. From this point forward, we refer to that method as "Recent". Method Recent starts by creating a record containing elements of the name and email address as shown on the left of Figure 1. It then forms candidate pools of identities linked by matching name parts, then uses a heuristic to accept or reject each pool based on counts of different similarity "clues". The authors then iteratively adapted this automatic identity matching by manually examining the pools of matched emails and adjusting the heuristic. To ensure that the heuristics in Recent were applied in a way consistent with their prior use, we asked the first author of the original paper [43] to run it on our datasets and adjust it analogously to how he had adjusted for his own studies[9]. We first compare the results of Recent to our approach on the entire set of 16K OpenStack authors and then on a larger dataset described below. Table 8 provides comparisons where the labels generated using method in the left column to be the ground truth. The "Comparison" column compares all methods to the first model from the 10 fold cross-validation which is considered to be the "assumed ground truth. The error is not zero for row ALFAA vs ALFAA, because it provides an average error over nine remaining models from the 10 fold cross-validation. We see that there is a good agreement between the raters: splitting error of 0.0139 and lumping error of 0.0139. The average ten-fold cross-validation error of ALFAA on the Rater 1 labeled data is 4.63

[8] https://github.com/bvasiles/ght_unmasking_aliases

[9] The author got much better results than we could obtained using their published code without modifications.

Table 8 Comparison of ALFAA against others

Set	Assumed Ground Truth	Comparison	Precision	Recall	Split	Lump
Training	R1	R2	0.9861	0.9861	0.0139	0.0139
Set	ALFAA	ALFAA	0.9990	0.9970	0.0030	0.001
	ALFAA	R2	0.9936	0.9823	0.0177	0.0063
Full	ALFAA	Bitergia	0.9991	0.4688	0.5312	0.0004
OpenStack	ALFAA	Recent	0.9480	0.8891	0.1109	0.0487

(0.0139/0.003) times and 13.9 (0.0139/0.001) times lower, though. The agreement between ALFAA and Rater 2 is similar to that between Rater 1 and Rater 2, suggesting that ALFAA has captured the heuristics implicit in the training set. Commercial effort has a much higher split error but it almost never lumps distinct individuals together. The last comparison of ALFAA vs Recent shows that ALFAA is 7.97 times more accurate than Recent with respect to splitting (0.1109/0.0139), and 3.5 times with respect to lumping (0.0487/0.0139). This is possible due to the large training sample that allows the Random Forest algorithm to devise a much more accurate decisions on matching than could be possible through a manual design of a heuristic.

We expect that this model will be used by other researchers and some of them may create additional training data. In such cases, as the training set would increase, we expect the accuracy of the supervised learning to continue to increase, hopefully approaching extremely low error rates.

6.3.1 ***Manual validation of results from Recent and ALFAA***

We found that ALFAA produced 1,876,595 matches that were not identified by Recent. Recent produced 363,818 matches that were not matched by ALFAA. In order to understand and validate the differences we design an experiment to compare the results of Recent to ALFAA generated from a set of 16,008 Openstack authors. As the size of the full data set is too large for manual validation (over 256M pairs), we sample a small section of the data for manual inspection. The sampling is done using the following procedure.

– Remove all self-matched observations from consideration i.e. where ID1 and ID2 are identical
– Remove all observations that have positive outcomes i.e. match, for both ALFAA and Recent
– Set1: Randomly sample 500 observations where ALFAA yields a positive outcome and Recent does not
– Set2: Randomly sample 500 observations where Recent yields a positive outcome and ALFAA does not
– Combine Set1 and Set2 and randomly extract 500 observations from the combined set without replacement. Let's call this Sample1. The remaining observations will be in Sample2. This step ensures that no bias is introduced when using human raters while generating ground truth.

We created random samples of 1000 pairs of IDs until it had more than 50% cases where email of ID1 was not identical to email of ID2 because recent automatically matched IDs that had the same email. We sampled 500 observations from each of these two sets and found that 418 observations in Set1 and 312 in Set2 did not have identical emails. We used two raters (all PhD students in Computer Science,) to label each observation as a match (1), a non_match(0), and uncertain (0.5) keeping the following in mind besides the individual rater's judgment.

- If first names and last names match and neither are homonyms label as 1.
- If names are somewhat similar (case difference, partial match, abbreviated etc.) and emails indicate personal and work (deduced from email domains) label as 1 Example: jonathanramirez <jonathanramirezmeza@gmail.com >,Jonathan Ramirez <jonathan@Jonathan-BMP.iptvṁicrosoftċom>
- If name in ID1 matches username in ID2 label as 1 Example: Elliot Fehr <elliot@weebly.com>, Elliot <elliotfehr@gmail.com>
- If a name has high empirical frequency (appears frequently in the dataset) label as 0.5
- If a name has high cultural frequency (known to be common in a nationality or culture) label as 0.5
- If one or both of the ID(s) is/are homonym(s) label as 0.5 Example: coffeemug <coffeemug@dell-desktop.example.com>, Slava Akhmechet<coffeemug @Elyas-MacBook-Pro.local>
- If there are multiple names present in the IDs, then label 0.5 Example: Jonathan Berkhahn and Raina Masand <rmasand@pivotal.io>, Ruth Byers and Raina Masand <rmasand@pivotal.io>

We selected all instances that were labeled as a match (1) or a non-match (0) by both raters. In other words, we dropped all instances that were labeled a 0.5 by any one rater. We extracted 534 such observations out of the 1000 labeled instances. There were 47 cases of disagreement between the two raters with a Cohen's Kappa of 0.722 which we conclude is satisfactory as Cohen's Kappa between 0.6 and 0.8 is considered to be substantial agreement according to Landis and Koch [19]. We found 184 pairs from results produced by ALFAA and 350 pairs from results produced by Recent in this set.

Rater 1 validation results for ALFAA: Out of the 178 pairs that were matched by ALFAA, rater 1 found 7 instances where ALFAA wrongly matched a pair of IDs and 177 instances where ALFAA correctly matched a pair of IDs. This means that ALFAA achieved 3.8% False Positive rate 96.2 True Positive rate from Rater 1.

Rater 1 validation results for Recent: Out of the 350 pairs that were matched by Recent, rater 1 found 80 instances where Recent wrongly matched a pair of IDs and 270 instances where Recent correctly matched a pair of IDs. This means that Recent achieved 22.8% False Positive rate 77.14% True Positive rate from Rater 1.

Rater 2 validation results for ALFAA: Out of the 178 pairs that were matched by ALFAA, Rater 2 found 28 instances where ALFAA wrongly matched a pair of IDs and 156 instances where ALFAA correctly matched a pair of IDs. This means that ALFAA achieved 15.2% False Positive rate 84.8% True Positive rate from Rater 2.

Rater 2 validation results for Recent: Out of the 350 pairs that were matched by Recent, rater 2 found 94 instances where Recent wrongly matched a pair of IDs and 256 instances where Recent correctly matched a pair of IDs. This means that Recent achieved 26.8% False Positive rate 73.14% True Positive rate from Rater 2.

In conclusion, ALFAA and Recent achieved an average false positive rate of 9.5% and 24.85% respectively. This shows that Recent is 2.61 times more prone to lumping error than ALFAA. To be able to report the splitting error, we would need to conduct a similar experiment by sampling from the set containing pairs that were labeled as a non-match (0) by ALFAA and the set containing pairs that were labeled as a non-match (0) by Recent, which is both time and labor intensive, and therefore, we keep it beyond the scope of this paper.

6.4 Evaluation on a large set of identities

To evaluate the feasibility of ALFAA on large scale we created a list of 1.8 million identities from commits to repositories in Github, Gitlab and Bioconductor for packages in 18 software ecosystems. The repositories were obtained from libraries.io data [32] for the Atom, Cargo, CocoaPods, CPAN, CRAN, Go, Hackage, Hex, Maven, NPM, NuGet, Packagist, Pypi, and Rubygems ecosystems; extracted from repository websites for Bioconductor[10], LuaRocks[11] and Stackage[12], and from Github searches for Eclipse plugins.

The application of Recent algorithm mapped the 1,809,495 author IDs to 1,411,531 entities as the algorithm was originally configured (1.28 aliases per entry), or 1,052,183 distinct entities after the heuristic was adjusted by its author, identifying an average of 1.72 aliases per entry. Upon applying our own algorithm to this dataset, we mapped the set to 988,905 — identifying an average of 1.83 aliases per entity. It is important to note that we did not incorporate any additional training beyond the original set of manually marked pairs and we expect the accuracy to increase further with an expanded training dataset.

Notably, to apply ALFAA for 1.8M IDs, we need 3.2×10^{12} string comparisons for each field (first name, last name, etc) and the same number of comparisons for each behavioral fingerprint. The full set of engineering decisions needed to accomplish the computation and prediction is beyond the scope of this paper, but the outline was as follows. To compare strings we used an allocation of 1.5 million core hours for Titan supercomputer at Oak

[10] https://www.bioconductor.org
[11] https://luarocks.org
[12] https://www.stackage.org/lts-10.5

Ridge Leadership Computing Facility (OLCF) [13]. The entire calculation was done in just over two hours after optimizing the implementation in pbdR [33] on 4096 16-core nodes. The approach can, therefore, scale to the entire set of over 23M author IDs in over 1B public commits. To compare behavioral fingerprints we exploited network properties (authors touch only a small number of all files) to reduce the number of comparisons by several orders of magnitude. Finally, it took us approximately two weeks to train Doc2Vec model on approximately 9M developer identities and 0.5B commits using Dell server with 800G RAM and 32 cores.

7 MEASURING IMPACT ON DEVELOPER COLLABORATION NETWORK

In this section of our work, we discuss RQ4, the impact of identity errors in a real world scenario of constructing a developer collaboration network. More specifically, we measure the impact of disaggregation (or split) errors by comparing the raw network to its corrected version. To create the collaboration network (a common network used in software engineering collaboration tools [8]), we start from a bipartite network of OpenStack with two types of nodes: nodes representing each author ID and nodes representing each file, we refer to as G. The edges connecting an author node and a file node represent the files modified by the author. This bipartite network is then collapsed to a regular author collaboration network by creating links between authors that modified at least one file in common. We then replace multiple links between the authors with a single link and remove authors' self links as well. The new network, which has 16,007 author nodes, depicts developer collaboration, we refer to as G′. We apply our disambiguation algorithm on G′and aggregate author nodes that belong to the same developer and produce a corrected network which we refer to as G′′. The network and its transformations are illustrated in Figure 5.

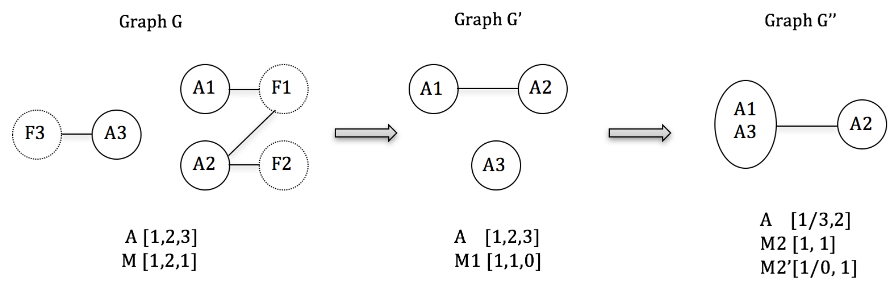


Fig. 5 Correcting Disaggregation Errors in a Developer Network

[13] https://www.olcf.ornl.gov/

To evaluate the impact of correction from G′to G″, we follow prior work investigating the impact of measurement error on social network measures [45]. We look at four node-level measurements of network error, i.e. degree centrality [14], clustering coefficient [46], network constraint [6] and eigenvector centrality [5]. For each node-level measure we compute a vector M. For example, in Figure 2, vector M has the degree centrality of G, G′ and G″. Similar to the approach discussed in [45] we compute two vectors M2 and M2′ for graph G″using each node level measure and compute Spearman's rho between these two vectors. We obtain Spearman's rho for degree centrality to be 0.8619, clustering coefficient to be 0.8685, network constraint to be 0.8406 and eigenvalue centrality to be 0.8690. The correlations below 0.95 for any of these measures are considered to indicate major disruptions to the social network [45]. In our case all of these measures are well below 0.95. We can also look at the quantiles of these measures: for example one quarter of developers in the corrected network have 210 or fewer peers, but in the uncorrected network that figure is 113 peers. The eigen-centrality has an even larger discrepancy: for one quarter of developers it is below 0.024 for the corrected and 0.007 (or more than four times lower) for the uncorrected network. Therefore, to answer RQ4, we find that developer identity errors have a high impact on collaboration networks.

8 LIMITATIONS

Our findings have a number of limitations. First, the proposed behavioral fingerprints may not be optimal for authorship assignment and other types of fingerprints may lead to a more accurate disambiguation. The results we got, however, show fairly high accuracy, suggesting that the specific choices we made appear to be reasonable as it is not clear how much more the accuracy can be increased. However, exploring other behavioral fingerprints should be considered in future work.

Second, our training and validation data rely on manual labeling by raters and not actual ground truth. To correct for the bias of individual raters, we used several raters to manually label the data and found that there was a high agreement among raters. To get to the actual ground truth, we contacted a small sample developers via a survey asking them to identify if the commits done by alternative IDs were in fact produced by them. We found low errors, but the response rate to the survey was fairly low. Therefore, it may be possible that the accuracy of the matches for the non-respondents of the survey may differ from what we observe with the respondents.

There may be a sampling bias as well, i.e., the errors for OpenStack identities may different from errors in the other projects and ecosystems. We, therefore applied our method and compared with a recent state-of-the-art method on a much large sample of identities from 18 distinct software ecosystems and found that it performs well there. We conducted additional manual labeling of the data on the mismatches to determine which of the methods was more accurate.

It is possible that identity errors may not impact research results much. We, therefore, looked into the question if the identity errors actually lead to errors in commonly constructed social networks. We found that even relatively small rate of identity errors to have a substantial impact on social network errors. While it would be ideal, to gauge the impact of ALFAA versus Recent in this context, we feel that this question is better addressed in the future work with additional analysis.

We mention in Section 4 that identity errors are can be classified broadly under two parts – Synonym and Homonym errors. The synonym resolution requires matching between pairs of authors, where the ground truth can be somehow determined based on the similarity of names, emails or other sources. It is harder to correct homonym IDs - for example it is impossible for a human rater to determine whether "anonymous <anonymous@gmail.com>" is the same as "Greg Holt <gregory.holt@gmail.com>" solely based on these two strings. Therefore, a modified approach is required for resolving homonyms. The proper identity of author needs to be determined for each commit separately (as multiple commits with the same homonym belong to distinct individuals). Using behavioral fingerprints of each commit (that has a homonym as an author ID) can help identify the most likely author. However, such experiment needs a careful design and ground truth has to be obtained at the resolution of each individual commit.

Finally, it is not clear if the proposed method would generalize to other domains or scale well. For example, would it work with data generated from tools other than Git? It is difficult to answer such questions without further studies, but given that the supervised learning approach was already being used in other domains suggest that the answer may be positive.

9 CONCLUSIONS

Through this work we have proposed a new approach (ALFAA) for correcting identity errors in software engineering context and apply it on OpenStack ecosystem. We find it to be several times more accurate than a commercial effort and a recent research method. More importantly, ALFAA does not rely on hand-crafted heuristics, but can, in contrast, increase its accuracy by simply incorporating additional training data. In fact, it is designed to work with the minimum amount of manual validation effort through the active learning approach.

To answer RQ1, we examined a very large collection of commits and found that the identity errors were substantially different from the types of errors that are common in domains such as administrative records (drivers licenses, population census), publication networks, or patent databases. Using Open Card-Sort approach we found that there are two primary types of errors (synonym and homonym errors) and further six sub-types within synonym errors and five within homonyms. These sub-types may be superimposed in some instances. While the data appears to have fewer phonetic spelling errors, it does contain similar typos. Additional errors involve template names or usage of names that imply desire for anonymity as well as missing data. Furthermore,the fraction of records with error appears to be much higher than in the other domains.

To answer RQ2 we summarize additional code commit information as behavioral fingerprints or vector embeddings of the very high-dimensional space represented by files modified, the times of these modifications, and the word embeddings of the commit messages. Such behavioral fingerprints can provide information needed to disambiguate common instances of homonyms due to tool templates or desire for anonymity.

To answer RQ3 we compared of our disambiguation approach with a commercial effort and with a recent research method. We found that our approach yielded several times lower errors, suggesting that it does represent a real improvement over the state of practice. Finally, to answer RQ4, we assessed the impact of measurement errors on the resulting networks. We found that use of uncorrected data would lead to major differences in resulting networks, thus raising questions about the validity of results for research studies that rely on such networks.

Our replication scripts and data are shared in a public repository[14]. We hope that with additional training data contributed, the models we share would become more accurate and that the proposed method and associated tool will make it easier to conduct research and to build tools that rely on accurate identification of developer identities and, therefore, lead to future innovations built on developer networks.

Acknowledgements

This research material is based on work supported by the National Science Foundation (NSF) grants IIS-1633437 and IIS-1901102. We would like to thank our collaborators at the Open Source Supply Chains and Avoidance of Risk (OSCAR) team at the University of Tennessee and from the Institute for Software Research (ISR) at the Carnegie Mellon University for their valuable feedback on this work.

References

1. Badashian, A.S., Esteki, A., Gholipour, A., Hindle, A., Stroulia, E.: Involvement, contribution and influence in github and stack overflow. In: Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, pp. 19–33. IBM Corp. (2014)
2. Baltes, S., Diehl, S.: Usage and attribution of stack overflow code snippets in github projects. Empirical Software Engineering pp. 1–37 (2018)

[14] https://github.com/ssc-oscar/ALFAA-Replication

3. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06, pp. 137–143. ACM, New York, NY, USA (2006). DOI 10.1145/1137983.1138016. URL http://doi.acm.org/10.1145/1137983.1138016
4. Bird, C., Rigby, P.C., Barr, E.T., Hamilton, D.J., German, D.M., Devanbu, P.: The promises and perils of mining git. In: 2009 6th IEEE International Working Conference on Mining Software Repositories, pp. 1–10. IEEE (2009)
5. Bonacich, P.: Power and centrality: A family of measures. American Journal of Sociology **92**(5), 1170–1182 (1987). DOI 10.1086/228631. URL https://doi.org/10.1086/228631
6. Burt, R.S.: Structural Holes. Harvard University Press (1992)
7. Cataldo, M., Herbsleb, J.D., Carley, K.M.: Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pp. 2–11. ACM (2008)
8. Cataldo, M., Wagstrom, P.A., Herbsleb, J.D., Carley, K.M.: Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, pp. 353–362. ACM (2006)
9. Christen, P.: A comparison of personal name matching: Techniques and practical issues. In: Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06), pp. 290–294 (2006). DOI 10.1109/ICDMW.2006.2
10. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string metrics for matching names and records. In: KDD WORKSHOP ON DATA CLEANING AND OBJECT CONSOLIDATION (2003)
11. Czerwonka, J., Nagappan, N., Schulte, W., Murphy, B.: Codemine: Building a software development data analytics platform at microsoft. IEEE software **30**(4), 64–71 (2013)
12. Edberg, D.T., Bowman, B.J.: User-developed applications: An empirical study of application quality and developer productivity. Journal of Management Information Systems **13**(1), 167–185 (1996)
13. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. Journal of the American Statistical Association **64**(328), 1183–1210 (1969). DOI 10.1080/01621459.1969.10501049. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1969.10501049
14. Freeman, L.C.: Centrality in social networks conceptual clarification. Social Networks **1**(3), 215 – 239 (1978). DOI https://doi.org/10.1016/0378-8733(78)90021-7. URL http://www.sciencedirect.com/science/article/pii/0378873378900217
15. German, D., Mockus, A.: Automating the measurement of open source projects. In: Proceedings of the 3rd workshop on open source software engineering, pp. 63–67. University College Cork Cork Ireland (2003)
16. German, D.M.: Mining cvs repositories, the softchange experience. In: 1st international workshop on mining software repositories, pp. 17–21. Citeseer (2004)
17. Gharehyazie, M., Posnett, D., Vasilescu, B., Filkov, V.: Developer initiation and social interactions in oss: A case study of the apache software foundation. Empirical Software Engineering **20**(5), 1318–1353 (2015). DOI 10.1007/s10664-014-9332-x. URL https://doi.org/10.1007/s10664-014-9332-x
18. Goeminne, M., Mens, T.: A comparison of identity merge algorithms for software repositories. Science of Computer Programming **78**(8), 971 – 986 (2013). DOI https://doi.org/10.1016/j.scico.2011.11.004. URL http://www.sciencedirect.com/science/article/pii/S0167642311002048
19. Hallgren, K.A.: Computing inter-rater reliability for observational data: an overview and tutorial. Tutorials in quantitative methods for psychology **8**(1), 23 (2012)
20. Jergensen, C., Sarma, A., Wagstrom, P.: 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, pp. 70–80. ACM (2011)
21. Kouters, E., Vasilescu, B., Serebrenik, A., van den Brand, M.G.J.: Whos who in gnome: using lsa to merge software repository identities. In: 28th IEEE International Conference on Software Maintenance (ICSM). IEEE (2012)
22. Lawrence, S., Giles, C.L., Bollacker, K.: Digital libraries and autonomous citation indexing. Computer **32**(6), 67–71 (1999). DOI 10.1109/2.769447

23. Le, Q., Mikolov, T.: Distributed representation of sentences and documents. In: Proceedings of the 31 st International Conference on Machine Learning, vol. 32. JMLR, Beijing,China (2014). URL https://cs.stanford.edu/ quocle/paragraph_vector.pdf
24. Ma, Y., Bogart, C., Amreen, S., Zaretzki, R., Mockus, A.: World of code: An infrastructure for mining the universe of open source vcs data. In: Proceedings of the 2019 International Conference on Mining Software Repositories (2019)
25. Martinez-Romo, J., Robles, G., Gonzalez-Barahona, J.M., Ortuño-Perez, M.: Using social network analysis techniques to study collaboration between a floss community and a company. In: B. Russo, E. Damiani, S. Hissam, B. Lundell, G. Succi (eds.) Open Source Development, Communities and Quality, pp. 171–186. Springer US, Boston, MA (2008)
26. Mockus, A.: Amassing and indexing a large sample of version control systems: towards the census of public source code history. In: 6th IEEE Working Conference on Mining Software Repositories. IEEE (2009). URL papers/amassing.pdf
27. Mockus, A.: Succession: Measuring transfer of code and developer productivity. In: Proceedings of the 31st International Conference on Software Engineering, pp. 67–77. IEEE Computer Society (2009)
28. Mockus, A.: Succession: Measuring transfer of code and developer productivity. In: 2009 International Conference on Software Engineering. ACM Press, Vancouver, CA (2009). URL papers/succession.pdf
29. Mockus, A.: Engineering big data solutions. In: ICSE'14 FOSE, pp. 85–99 (2014). URL http://dl.acm.org/authorize?N14216
30. Mockus, A., Herbsleb, J.D.: Expertise browser: a quantitative approach to identifying expertise. In: Proceedings of the 24th international conference on software engineering, pp. 503–512. ACM (2002)
31. Nagappan, N., Murphy, B., Basili, V.: The influence of organizational structure on software quality. In: 2008 ACM/IEEE 30th International Conference on Software Engineering, pp. 521–530. IEEE (2008)
32. Nesbitt, A., Nickolls, B.: Libraries.io Open Source Repository and Dependency Metadata (2017). DOI 10.5281/zenodo.808273. URL https://doi.org/10.5281/zenodo.808273
33. Ostrouchov, G., Chen, W.C., Schmidt, D., Patel, P.: Programming with big data in r. URL http://r-pbd. org (2012)
34. Petersen, K., Wohlin, C.: Measuring the flow in lean software development. Software: Practice and experience **41**(9), 975–996 (2011)
35. Pinzger, M., Nagappan, N., Murphy, B.: Can developer-module networks predict failures? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, SIGSOFT '08/FSE-16, pp. 2–12. ACM, New York, NY, USA (2008). DOI 10.1145/1453101.1453105. URL http://doi.acm.org/10.1145/1453101.1453105
36. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pp. 45–50. ELRA, Valletta, Malta (2010)
37. Robles, G., Gonzalez-Barahona, J.M.: Developer identification methods for integrated data from various sources. In: Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR '05, pp. 1–5. ACM, New York, NY, USA (2005). DOI 10.1145/1082983.1083162. URL http://doi.acm.org/10.1145/1082983.1083162
38. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '02, pp. 269–278. ACM, New York, NY, USA (2002). DOI 10.1145/775047.775087. URL http://doi.acm.org/10.1145/775047.775087
39. Sariyar, M., Borg, A.: The recordlinkage package: Detecting errors in data. The R Journal **2**(1), 61–67 (2010). URL https://journal.r-project.org/archive/2010-2/RJournal_2010-_Sariyar+Borg.pdf
40. Smalheiser, N.R., Torvik, V.I.: Author name disambiguation. Annual Review of Information Science and Technology **43**(1), 1–43 (2011). DOI 10.1002/aris.2009.1440430113. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/aris.2009.1440430113
41. Spencer, D., Warfel, T.: Card sorting: a definitive guide. Boxes and Arrows p. 2 (2004)

42. Thung, F., Bissyande, T.F., Lo, D., Jiang, L.: Network structure of social coding in github. In: 2013 17th European Conference on Software Maintenance and Reengineering, pp. 323–326. IEEE (2013)
43. Vasilescu, B., Serebrenik, A., Filkov, V.: A data set for social diversity studies of github teams. In: Proceedings of the 12th Working Conference on Mining Software Repositories, pp. 514–517. ACM (2015). URL https://dl.acm.org/citation.cfm?id=2820601
44. Ventura, S.L., Nugent, R., Fuchs, E.R.: Seeing the non-starts: (some) sources of bias in past disambiguation approaches and a new public tool leveraging labeled records. Elsevier (2015)
45. Wang, D.J., Shi, X., McFarland, D.A., Leskovec, J.: Measurement error in network data: A re-classification. Social Networks **34**(4), 396–409 (2012)
46. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**, 440–442 (1998). DOI 10.1038/30918
47. Wiese, I.S., d. Silva, J.T., Steinmacher, I., Treude, C., Gerosa, M.A.: Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 345–355 (2016). DOI 10.1109/ICSME.2016.13
48. Winkler, W.E.: Overview of record linkage and current research directions. Tech. rep., BUREAU OF THE CENSUS (2006)
49. Wolf, T., Schrter, A., Damian, D., Panjer, L.D., Nguyen, T.H.D.: Mining task-based social networks to explore collaboration in software teams. IEEE Software **26**(1), 58–66 (2009). DOI 10.1109/MS.2009.16
50. Xiong, Y., Meng, Z., Shen, B., Yin, W.: Mining developer behavior across github and stackoverflow. In: The 29th International Conference on Software Engineering and Knowledge Engineering, pp. 578–583 (2017). DOI 10.18293/SEKE2017-062
51. Zhou, M., Mockus, A., Ma, X., Zhang, L., Mei, H.: Inflow and retention in oss communities with commercial involvement: A case study of three hybrid projects. ACM Transactions on Software Engineering and Methodology (TOSEM) **25**(2), 13 (2016)
52. Zhu, J., Wei, J.: An empirical study of multiple names and email addresses in oss version control repositories. In: Proceedings of 16th International Conference on Mining Software Repositories (MSR). IEEE/ACM (2019)