Interlinking Developer Identities Within and Across Open Source Projects: The Linked Data Approach

Aftab Iqbal
Digital Enterprise Research Institute,
National University of Ireland
Galway, Ireland
Email: aftab.iqbal@deri.org

Michael Hausenblas
Digital Enterprise Research Institute,
National University of Ireland
Galway, Ireland
Enail: michael.hausenblas@deri.org

***Abstract*—Software developers use various software repositories in order to interact with each other or to solve related problems. These repositories provide a rich source of information for a wide range of tasks. However, one issue to overcome in order to make this information useful is the identification and interlinking of multiple identities of developers. In this paper, we propose a Linked Data-based methodology to interlink and integrate multiple identities of a developer found in different software repositories of a project as well as across repositories of multiple projects. By providing such interlinking will enable us to keep track of a developer's activity not only within a single project but also across multiple projects. The methodology will be presented in general and applied to 5 Apache projects as a case study. Further, we show that the few methods suggested so far are not always appropriate to overcome the developer identification problem.**

***Keywords*-Linked Data; Software Engineering; FLOSS Repositories; Data Integration; Developer Identities**

I. Introduction and Motivation

In *Software Engineering*, many tools with underlying repositories have been introduced to support the collaboration and coordination in distributed software development. Research has shown that these software repositories contain rich amount of information about software projects. By mining the information contained in these software repositories, practitioners can depend less on their experience and more on the historical data [11]. However, software repositories are commonly used only as record-keeping repositories and rarely for design decision processes [10]. Examples of software repositories are [12]: source control repositories, bug repositories, archived communication etc.

Developers[1] use these repositories to interact with each other or to solve software-related problems. By extracting rich information from these repositories, one can guide decision processes in modern software development. For example, source-code and bugs are quite often discussed on bug repositories and project mailing lists. Data in these software repositories could be analyzed to extract bug and source-code related discussions, which could be linked to the actual bug description and source-code. This could allow keeping track of developers discussion related to a bug or source-code in different software repositories.

Developers are required to adopt an identity for each software repository they want to use. For example, they are required to adopt an email address in order to send an email to the project mailing list, adopt an ID[2] to push commits to the source control repository, adopt an ID to report bugs or post comments on a bug in bug repository etc. Often developers adopt different ID(s) for each software repository and sometimes multiple ID(s) for the same repository [21], while interacting with these software repositories in different context. The software repositories are designed to help developers collaborate and coordinate but lacks the feature to manage and merge multiple ID(s) of a developer specially in the case of mailing list and bug tracking repositories. Hence, a developer can register multiple ID(s) and use them to collaborate and coordinate with other developers. For example, Figure 1 shows a subset of the social network graph derived from the communication occurred among developers on the mailing list of an *Apache* project.

In the figure, we colored two nodes (i.e., red and blue) showing that both email ID(s) belong to a developer `X`[3]. It is quite clear that the developer `X` has used both email ID(s) to communicate with other developers on the mailing list. Looking into the social structure of graphs (based on both ID(s)), we see that there are few ID(s) which are common in both graphs but most of the ID(s) are part of different social graphs of developer `X`. In an ideal scenario, either both social graphs need to be merged into one graph or at-least there should exist some sort of information conveying that both ID(s) belong to developer `X`. Without any information provided about the different ID(s) used by a developer, a researcher who carries out social network analysis research [9], [19], [18] might consider both ID(s) as two different developers because using distinct ID(s) makes

[1]We will use the term "developer" to represent the core developers, contributors, bug reporters and users of an open source project.

[2]We will use the term "ID" to represent different types of identities which developers use to interact with software repositories.

[3]We shortened the labels of each node in order to keep the privacy of developers.

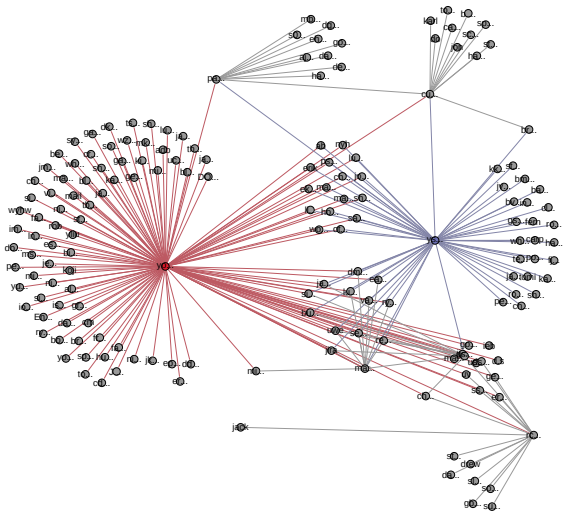

Figure 1. Different Identities of a Developer.

developers appear as different entities. Furthermore, analysis of developer's activity [7] within a project might be difficult if the developer is using different and multiple ID(s) for each software repository.

In the specific case of open source software development, developers are often involved in multiple projects. One of the many reasons is that they see their contribution of time and efforts as a gift they provide to the project community. Other reasons could be a developer's ability to acquire skills and experience by participating in multiple open source projects. Hence the development activity of a developer is distributed not only within different software repositories of a project but also across different projects. It is worth mentioning that there exists an implicit connection between developer's activity within different software repositories of a project and also across different projects. Therefore, we require methods and techniques to correctly identify and relate the different ID(s) of developers not only within a project but also across different projects.

The contribution of this paper is manifolds: we propose a simple yet effective Linked Data-based approach to identify and interlink multiple ID(s) of a developer within different software repositories of a project as well as across software repositories of different projects. We show how interlinking helps in integrating developer's development activity which is distributed within different software repositories of a project due to the usage of multiple ID(s). Further, we show how the interlinking enables us to keep track of developer's activity across different projects. Further, we compare the matching results of our approach with other existing developer identification approaches.

The paper is structured as follows: in Section II we review related work and compare it to our approach. Research questions are outlined in section III. In Section IV, we present our approach to identify and interlink multiple ID(s) of a developer. We report our evaluation on 5 *Apache* projects in Section V followed by the discussion on the results and research questions in Section VI. Finally, we conclude in Section VII and outline future steps.

II. Related Work

To the best of our knowledge, there are only a few published works on identifying and relating the different ID(s) that developers use to interact with software repositories of a project. In [6], Bird et al. proposed an approach to produce a list of <name,email> pairs by parsing the emails and clustering them. The clustering algorithm to measure the similarity between every pair of ID(s) is based on string similarity between names, between emails, between names and emails, etc. Two ID(s) with a similarity measure lying below a pre-defined threshold value are placed into the same cluster. The authors use different approaches to compute the similarity measures between every pair of ID(s), some of which we tested on our data set to validate the effectiveness of these approaches (cf. Section V). We also use the string similarity approach in order to interlink the ID(s) of a developer but our scope is broader in a sense that we are applying the heuristics not only within a single project or a repository but also across repositories of multiple projects.

In [21], Robles et al. discusses the problem of developer identification in general, but the work lacks in details about the heuristics they propose to identify and match the different ID(s) of developers. This makes it difficult to validate their approaches for solving this problem. The authors propose a technique to build one identity from another by extracting the "real life" name from email addresses, such as *surname@domain.com, name.surname@domain.com* etc. This is an approach based on pre-defined name schemes. Further, they try to match user names obtained from CVS to email addresses (excluding the domain after "@"). Their approach also relies on string similarity algorithm.

In [8], Megan et al. presents a methodology to integrate data about projects by way of matching projects (entities) based on certain heuristics. The heuristics include a simple scoring system for confidence in pairwise project matches. The author propose to interlink similar projects which are listed on multiple code forges. On the contrary, in this paper we focused on identification and interlinking multiple ID(s) of a developer.

In general, the problem is related to duplicate detection. The duplicate detection frameworks provide several techniques to effectively solve matching different entities. In this regard, Kopcke et al. [17] analyzed and compared 11

different duplicate detection frameworks. While research in this area mostly refers to identifying duplicates in the same data set, the techniques might be mapped to the case of matching over different data sets. However, they are tailor-made for identifying different ID(s) of the same developer inside one repository. Naumann et al.[20] provides a nice overview of this research direction.

In the Semantic Web domain, Volz et al. [23] proposed an interlinking framework also known as SILK framework which generates link between two RDF data sets based on some string similarity measures specified by the user. Their interlinking framework supports different string similarity algorithms to compare if two different RDF resources are similar or not. In a next step, we will assess to what extent we can use the SILK framework to interlink developer identities in the near future.

III. Research Questions

In the following, we list down few research questions which will be addressed later (Section VI) based on interlinking multiple ID(s) of a developer:

1) **RQ-1:** *What is the added benefits of interlinking multiple ID(s) of a developer?*
   We will investigate if developers are using multiple ID(s) within a software repository and if interlinking them will provide any added benefits. We will validate this on the mailing list data set by computing the distinct nodes which are not shared among different social graphs of a developer using multiple ID(s). Further, we will compute the number of bugs reported by a developer using multiple ID(s). This would show us a clear picture on the amount of information which was not previously connected.
2) **RQ-2:** *What is the ratio of developer's existence in multiple projects?*
   We will investigate the frequency of developer's participation in multiple projects. We will evaluate if developer's participation follow Pareto's law i.e., many developers participate in few projects and only few developers participate in many projects.
3) **RQ-3:** *What are the contributions made by developers in multiple projects?*
   As mentioned in Section I, developers often contribute to multiple open source projects. Therefore, we will evaluate to what extent our interlinking approach helps us in tracing developer's activity across multiple projects. In particular, we will investigate if developers are highly active in pushing source control commits, reporting bugs and participating in discussions across multiple projects.

IV. Our Approach

In this section, we describe our approach for identifying and interlinking different ID(s) of a developer found in different software repositories of a project as well as across repositories of multiple projects. We considered mailing list and bug repository data sources in this paper because developers often use multiple ID(s) on these software repositories as opposed to source control repository where access and accounts are controlled centrally.

With "Linked Data Driven Software Development" (LD2SD) [15], we have introduced a Linked Data-based methodology to relate and integrate data across software repositories explicitly and unambiguously. We propose to use Semantic Web technologies to represent data from different software repositories. As such, we propose to use RDF [16] (Resource Description Framework) as the core, target data model. Once modeled in RDF, the data can be easily integrated, indexed and queried using the SPARQL query[4] standard and associated tools. Finally, the integrated data can be published on the Web using Linked Data principles[5] allowing third parties to discover and subsequently crawl the knowledge, and also allowing to interlink with background or other related information available remotely on the Web. We refer the readers to [13] for details on how these standards would be used. Instead here we focus on the identification and interlinking of developer ID(s) in software repositories.

As mentioned by Bird et al., most emails contain header of the form [6]:

```
From: "aftab iqbal" <ai@example.com>
```

For each email, we represent header information in RDF using FOAF[6] ontology as shown in Listing 1 (`lines 10-12`)[7]. We used email alias to build developer URI (i.e., a resource in RDF) because email alias can be used to distinguish between different developers. Further, the name and email address are defined as property values of that resource.

While converting mailing list data to RDF using our custom written scripts, we found certain developers who have used either full or part of their names in the email header. For example:

```
From: "muhammad aftab iqbal" <ai@example.com>
```

The above two email headers reveal multiple name(s) of the sender (i.e., `aftab iqbal` and `muhammad aftab iqbal`). The developer URI will remain the same in the case of above two exemplary email headers (i.e., `http://srvgal85.deri.ie/linkedfloss/ai`) but the property values will be changed (name in that case)

[4] http://www.w3.org/TR/rdf-sparql-query/
[5] http://www.w3.org/DesignIssues/LinkedData.html
[6] http://www.foaf-project.org/
[7] The URIs used in the listings are for illustration purpose only and are not dereferenceable.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sioct: <http://rdfs.org/sioc/types#> .
@prefix email:
     <http://simile.mit.edu/2005/06/ontologies/email#> .
@prefix :
     <http://srvgal85.deri.ie/linkedfloss/email2rdf/> .
 :10029029 a sioct:MailMessage ;
email:from <http://srvgal85.deri.ie/linkedfloss/ai> ;
email:subject "compilation problem" ;
email:body "while compiling the latest source i am ..."
     .

<http://srvgal85.deri.ie/linkedfloss/ai> a foaf:Person ;
foaf:name "aftab iqbal" ;
foaf:mbox <mailto:ai@example.com> .
...
```

Listing 1. An Exemplary Email RDFication.

as each email header belongs to a different email. Therefore, multiple name(s) can be easily extracted to produce a list of <name,id> pairs for a particular developer. Each bug usually represent developer information of the form:

```
<assignee username="aftab.iqbal">aftab iqbal</assignee>
<reporter username="jim">jim</reporter>
```

We represent developer information for each bug in RDF using FOAF ontology as shown in Listing 2 (`lines 10-12`).

```
@prefix baetle: <http://baetle.googlecode.com/svn/ns/#>
     .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix :
     <http://srvgal85.deri.ie/linkedfloss/bug2rdf/> .
 :41508 a baetle:Issue;
baetle:assigned_to
     <http://srvgal85.deri.ie/linkedfloss/aftab.iqbal> ;
baetle:reporter
     <http://srvgal85.deri.ie/linkedfloss/jim> ;
baetle:summary "compilation issue in build 5.1" ;
baetle:hasState "Open" .

<http://srvgal85.deri.ie/linkedfloss/aftab.iqbal> a
     foaf:Person ;
foaf:name "aftab iqbal" ;
foaf:accountName "aftab.iqbal" .
...
```

Listing 2. An Exemplary Bug RDFication.

After transforming the data into RDF, we loaded the RDF data sets into SPARQL endpoint[8]. The next step is to identify and interlink multiple ID(s) of a developer. From the Listings 1 and 2, we are able to conclude that the developer (`aftab iqbal`) has used different ID(s) (i.e., `ai` and `aftab.iqbal`) while interacting with others on the mailing list and bug tracking repository. We can interlink these two RDF fragments as shown in Listing 3 using an `owl:sameAs` property indicating that the developer URIs actually refer to the same entity.

[8]http://linkedfloss.srvgal85.deri.ie/sparql

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
<http://srvgal85.deri.ie/linkedfloss/ai> owl:sameAs
     <http://srvgal85.deri.ie/linkedfloss/aftab.iqbal> .
```

Listing 3. An Interling Example.

By explicitly interconnecting ID(s) of a developer, we will be able to trace developer's activity on the mailing list as well as bug tracking repository. However, manually identifying and interlinking ID(s) of a developer is a time consuming task especially when dealing with large amount of data. Hence, one is required to devise an automatic approach for the identification and interlinking of developer ID(s). Therefore, in the following we will explain our simple yet effective developer matching approach.

A. Preliminaries

Let **`Authors`** be a list of distinct developer URIs found in the data set:

$$\textbf{Authors} = \{A, B, C, D, \cdots\} \tag{1}$$

where

$$\mathbf{A} = http://srvgal85.deri.ie/linkedfloss/ai \tag{2}$$

and

$$\mathbf{B} = http://srvgal85.deri.ie/linkedfloss/aftab.iqbal \tag{3}$$

Let $\acute{\mathbf{A}}$ and $\acute{\mathbf{B}}$ be the distinct <name,id> pairs for **A** and **B** respectively:

$$\acute{\mathbf{A}} = \begin{pmatrix} < aftab\ iqbal, ai@example.com > \\ < muhammad\ aftab\ iqbal, ai@example.com > \end{pmatrix} \tag{4}$$

$$\acute{\mathbf{B}} = \left(< aftab\ iqbal, aftab.iqbal@example.com >\right) \tag{5}$$

Let **`L`** be a set which contains a collection of distinct developer URI(s) along with their associated <name, id> pairs:

$$\mathbf{L} = \begin{pmatrix} (\mathbf{A}, \acute{\mathbf{A}}) \\ (\mathbf{B}, \acute{\mathbf{B}}) \\ (\mathbf{C}, \acute{\mathbf{C}}) \\ \vdots \end{pmatrix} \tag{6}$$

B. Interlinking id(s) within a repository

We compared each developer URI with every other developer URI in set **`L`** by comparing their respective <name,id> (where `id` means excluding everything after "@") pairs. If the match is found then we established an `owl:sameAs` link between both developer URIs as shown in Listing 3. The pseudocode of our approach is shown in IV.1.

Algorithm IV.1: ID_LINKING_IN_A_REPOSITORY()

```
lstURI ← get distinct developer URIs
for i ← 0 to lstURI.size()
  do { lst ← get id(s) and name(s) associated with lstURI[i]
       hshMap.add(devURI[i], lst)
comment: match each developer URI with every other developer URI in
comment: the list based on the <name,id> pairs
for i ← 0 to hshMap.size()
  do { id1 ← hshMap.getKey(i)
       lst1 ← hshMap.getValue(id1)
       for j ← i + 1 to hshMap.size()
         do { id2 ← hshMap.getKey(j)
              lst2 ← hshMap.getValue(id2)
              if matched(lst1, lst2)
                then {generateSameAsLink(id1, id2)
```

C. Interlinking id(s) across projects

Let $\mathbf{Prj}_A$ and $\mathbf{Prj}_B$ (cf. (7) and (8)) be the sets which contains the collection of distinct developer URI(s) along with their associated <name, email> pairs extracted from the repositories of `project A` and `project B` such that:

$$\mathbf{Prj}_A = \begin{pmatrix} (\mathbf{A}, \acute{\mathbf{A}}) \\ (\mathbf{B}, \acute{\mathbf{B}}) \\ (\mathbf{C}, \acute{\mathbf{C}}) \\ \vdots \end{pmatrix} \tag{7}$$

$$\mathbf{Prj}_B = \begin{pmatrix} (\mathbf{A}, \acute{\mathbf{A}}) \\ (\mathbf{B}, \acute{\mathbf{B}}) \\ (\mathbf{C}, \acute{\mathbf{C}}) \\ \vdots \end{pmatrix} \tag{8}$$

For each developer URI, we queried the `owl:sameAs` links (if any) in order to build a comprehensive list of <name,email> pairs. Later, we compared each developer URI in set $\mathbf{Prj}_A$ with every developer URI in set $\mathbf{Prj}_B$ by comparing their respective <name,email> pairs. If the match is found then we established an `owl:sameAs` link between both developer URIs (cf. Listing 3). The pseudocode of our approach is shown in IV.2.

Algorithm IV.2: ID_LINKING_ACROSS_PROJECTS()

```
prj_A ← get distinct developer URIs from project A
prj_B ← get distinct developer URIs from project B
for i ← 0 to prj_A.size()
  do { lstID ← getSameAsLinks(prj_A[i])
       for j ← 0 to lstID.size()
         do { lst ← get email(s) and name(s) associated with lstID[j]
              lstPairs.add(lst)
       hshMap_A.add(prj_A[i], lstPairs)
for i ← 0 to prj_B.size()
  do { lstID ← getSameAsLinks(prj_B[i])
       for j ← 0 to lstID.size()
         do { lst ← get email(s) and name(s) associated with lstID[j]
              lstPairs.add(lst)
       hshMap_B.add(prj_B[i], lstPairs)
comment: match each developer URI of project A with every developer URI
comment: in project B based on the <name,email> pairs
for i ← 0 to hshMap_A.size()
  do { id1 ← hshMap_A.getKey(i)
       lst1 ← hshMap_A.getValue(id1)
       for j ← 0 to hshMap_B.size()
         do { id2 ← hshMap_B.getKey(j)
              lst2 ← hshMap_B.getValue(id2)
              if matched(lst1, lst2)
                then {generateSameAsLink(id1, id2)
```

The outcome of our above described matching approach(es) is a set of RDF files in `N-TRIPLES`[9] format which contains the developer URIs having `owl:sameAs` links between them (e.g., see Listing 3). The RDF files were later loaded into the SPARQL endpoint so that it can be used to query development activity of a developer within and across projects.

V. Evaluation

Before we discuss the results of our matching approach(es), we describe the *Apache* projects selected for evaluation. We gathered data from software repositories of 5 *Apache* projects (c.f Table I). The reason of choosing *Apache* projects is that the repositories of these projects are on the Web and are available to download (i.e. mailing list archives, bugs etc.). We selected data from the beginning of each *Apache* project to date as listed in Table I.

Apache Projects	Date Range
Apache Ant [1]	2000 - 2012
Apache Hadoop [2]	2006 - 2012
Apache Logging [3]	2001 - 2012
Apache Lucene [4]	2001 - 2012
Apache Maven [5]	2003 - 2012

Table I
APACHE PROJECTS DATA RANGE.

A. Interlinking id(s) within a repository

We applied our interlinking approach (cf. algorithm IV.1) on the mailing list and bug tracking repositories separately in order to identify and interlink multiple ID(s) of a developer within a repository. During the matching phase, we found certain ID(s) which were more generic and multiple developers in each project were associated to those ID(s). We excluded those ID(s) during the matching phase. Example of those ID(s) are: `jakarta`, `jakarta-ant`, `lists`, `ant-dev`, `dev`, `ant`, `apache`, `general`, `hadoop`, `nutch-dev`, `log4j`, `log4j-dev`, `java-dev`, `lucene` etc. The outcome of our interlinking approach is listed in Table II and Table III where the ``links'' column tells the total number of `owl:sameAs` links established between ID(s) and the ``id(s) found'' column tells the distinct ID(s) found with a match.

Apache Projects	No. of developers	id(s) found	links
Apache Ant	2,469	453	320
Apache Hadoop	1,260	189	118
Apache Logging	988	105	76
Apache Lucene	1,506	208	158
Apache Maven	1,886	340	246

Table II
MATCHING EMAIL ID(S) WITHIN A PROJECT.

[9]http://www.w3.org/2001/sw/RDFCore/ntriples/

Apache Projects	No. of developers	id(s) found	links
Apache Ant	3,927	143	76
Apache Hadoop	1,219	14	7
Apache Logging	1,131	42	21
Apache Lucene	981	27	15
Apache Maven	1,577	41	22

Table III
MATCHING BUG REPOSITORY ID(S) WITHIN A PROJECT.

From Table II and Table III, it is quite obvious that the usage of multiple ID(s) is mostly common on the mailing list than the bug tracking repository except the special case of *Apache ANT* project. However, it is still important to identify and interlink multiple ID(s) of a developer in every software repository. The same interlinking approach can be used to match developer ID(s) across different repositories (i.e., matching mailing list and bug tracking repository developer ID(s)) of a project.

B. Interlinking id(s) across projects

In order to identify and interlink ID(s) of a developer across different projects, we applied our interlinking approach (cf. algorithm IV.2) to the bug tracking and mailing list repositories of the *Apache* projects separately. We considered two ID(s) to be similar if either the <name,email> pairs of both ID(s) matched or if only the `name` or `email address` of the ID(s) matched. During the evaluation, we found match for certain developers who have used single word name (e.g., chris, brian, J etc.). As it is difficult to differentiate if the developers are same (based on a single word name), we only considered a match if the developer name consists of at-least two words (i.e., name surname). The results of our cross project matching approach based on mailing list and bug tracking repositories are shown in Table IV and Table V.

The results shows that matching based on either `name` or `email address` generates more links comparing to the <name,email> approach. Also, there are good number of developers whose ID(s) are found in multiple projects. The most prominent among them are the developers of *Apache ANT* and *Apache Maven* project. The reasons behind it could be the technical dependencies between both projects (e.g., using project components) or the interest of developers working on similar kind/category of projects.

C. Comparison of developer identification approaches

In this section, we apply few methods suggested so far (see section II) on our data set in order to validate the effectiveness of these methods comparing to our approach. We applied the methods only on the mailing list data set because we found the usage of multiple ID(s) more on the mailing list comparing to the bug repository data set. In order to identify multiple ID(s) of a developer, we computed similarities between developer names and email addresses or between two email addresses using Levenshtein edit distance [22] algorithm, as suggested by Bird et. al in [6]. The match(es) are those developers with the lowest edit distance. Clearly, there is no sense in allowing arbitrary large distance values so we tested the methods by setting three different threshold values:

1) the maximal length of both strings
2) the minimal length of both strings
3) a fixed threshold value of 4

In order to identify the similarity between names and emails (i.e., Name-Email approach), we matched email addresses (excluding everything after "@") and developer names by doing a pairwise comparison. Besides names and emails, it is very likely that the emails are textually similar to each other if they belong to the same developer. Thus, we also computed the pairwise similarities between email addresses (i.e., Email-Email approach) and determined match(es) for one email by choosing those with the lowest edit distance. Later, we validate the match(es) produced by each method and discard the wrong match(es). The results of both methods comparing to our method is shown in Figure 2.

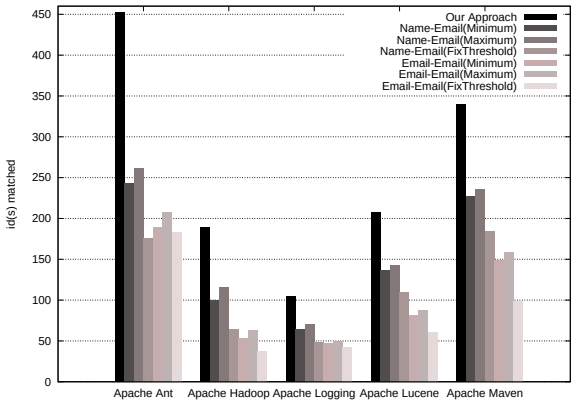

Figure 2. Comparison of Different Matching Approaches.

The figure shows that our approach identifies more ID(s) comparing to other existing approaches. Although combination of existing approaches with our approach might produce even better matching results but this is not in the current scope of this paper.

VI. DISCUSSION

In this section, we discuss the research questions (cf. Section III) based on the results we achieved through interlinking multiple ID(s) of a developer.

ProjectA (no. of developers)	ProjectB (no. of developers)	Mailing List			
		Name&Email		Name/Email	
		id(s) found	links	id(s) found	links
Apache Ant (2469)	*Apache Hadoop (1260)*	24	12	59	43
Apache Ant (2469)	*Apache Logging (988)*	153	77	232	151
Apache Ant (2469)	*Apache Lucene (1506)*	75	38	129	92
Apache Ant (2469)	*Apache Maven (1886)*	198	100	365	254
Apache Hadoop (1260)	*Apache Logging (988)*	2	1	6	3
Apache Hadoop (1260)	*Apache Lucene (1506)*	86	43	121	82
Apache Hadoop (1260)	*Apache Maven (1886)*	34	17	48	29
Apache Logging (988)	*Apache Lucene (1506)*	40	20	68	41
Apache Logging (988)	*Apache Maven (1886)*	96	51	156	103
Apache Lucene (1506)	*Apache Maven (1886)*	92	46	134	81

Table IV
MATCHING EMAIL ID(S) ACROSS PROJECTS.

ProjectA (no. of developers)	ProjectB (no. of developers)	Bug Tracking System			
		Name&Email		Name/Email	
		id(s) found	links	id(s) found	links
Apache Ant (3927)	*Apache Hadoop (1219)*	20	10	68	35
Apache Ant (3927)	*Apache Logging (1131)*	242	121	363	197
Apache Ant (3927)	*Apache Lucene (981)*	24	12	61	34
Apache Ant (3927)	*Apache Maven (1577)*	74	37	261	143
Apache Hadoop (1219)	*Apache Logging (1131)*	2	1	19	10
Apache Hadoop (1219)	*Apache Lucene (981)*	44	22	45	23
Apache Hadoop (1219)	*Apache Maven (1577)*	20	10	20	10
Apache Logging (1131)	*Apache Lucene (981)*	4	2	16	8
Apache Logging (1131)	*Apache Maven (1577)*	32	16	98	52
Apache Lucene (981)	*Apache Maven (1577)*	16	8	27	14

Table V
MATCHING BUG REPOSITORY ID(S) ACROSS PROJECTS.

RQ-1: *What is the added benefits of interlinking multiple ID(s) of a developer?*

In order to evaluate the added benefits of interlinking multiple ID(s) of a developer, we calculated the difference in the social graphs constructed from the mailing list data set and the number of bugs reported by a developer whose multiple ID(s) were not interlinked. We wanted to find out how much of the developer's development activity is not explicitly interconnected due to the usage of multiple ID(s) within a software repository.

We calculated the difference between social graphs of a developer who has multiple ID(s) for each *Apache* project. The social graphs was constructed by analyzing the reply structure of the email threads. The difference between two social graphs belonging to the same developer was calculated by computing the number of nodes (i.e. id) which are part of one social graph of a developer and is missing in the other social graph of that particular developer. This would reveal distinct nodes which are part of different social graphs of a developer and lacks a common connection (as motivated in Figure 1). We sorted the results and plotted it in the form of a chart as shown in Figure 3.

The charts plotted shows a potential difference in the social graphs of a developer who has multiple ID(s). For example, in *Apache Maven* project (cf. Figure 3), we found a developer whose multiple social graphs had a difference of 613 nodes. This means that there are 613 nodes which

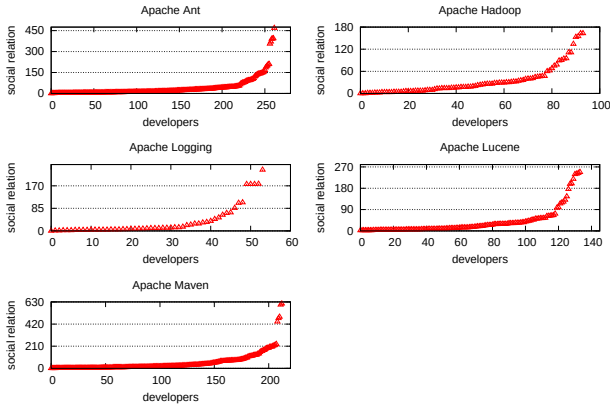

Figure 3. Social Relation Difference.

are not shared among his multiple social graphs. We further computed the number of bugs reported by a developer using multiple ID(s) and plotted the results in the form of chart as shown in Figure 4. We stacked up the bugs reported by a developer using multiple ID(s) to show the total number of bugs he/she reported.

The results depicted in Figure 3 and Figure 4 shows that establishing `owl:sameAs` links between multiple ID(s) of a developer helps in better integration of data sources from

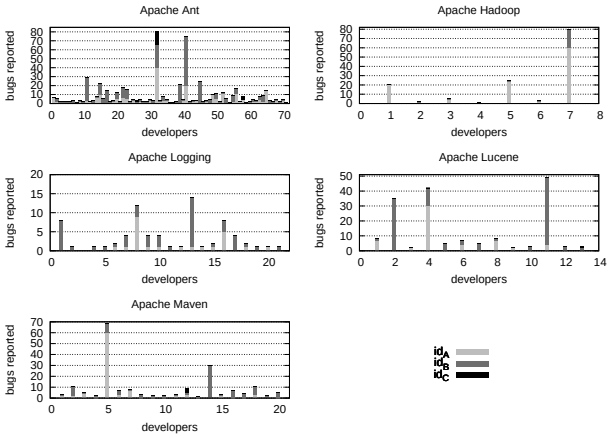

Figure 4. Bugs Reported Difference.

the research (i.e., carrying out social network analysis) as well as the software development point of view (i.e., tracing developer activity in a repository). However, based on the small number of *Apache* projects we selected for this study, the bugs reporting activity using multiple ID(s) are not significant comparing to the mailing list's social graph differences but we believe that one might find significant differences if it is applied to a large number of open source projects.

RQ-2: *What is the ratio of developer's existence in multiple projects?*

Based on the evaluation data from 5 *Apache* projects, we queried `owl:sameAs` links of each developer across different *Apache* projects under consideration. Later, we counted the number of projects for which a developer has an `owl:sameAs` link and plotted the results as shown in Figure 5.

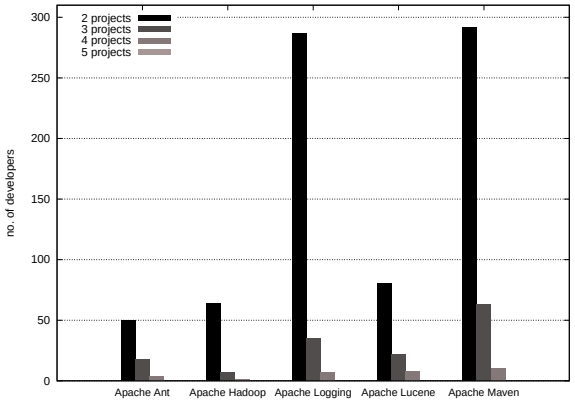

Figure 5. Developer's Existence in Multiple Apache Projects.

The chart shows that developer's participation in multiple projects follow Pareto's law because of the large number of developer's participation in 2 *Apache* projects in contrast to participation in 3,4 or 5 *Apache* projects. For each *Apache* project, we found a small subset of developers participating in at-least 4 *Apache* projects. It shows us a great tendency of developer's participation in multiple projects. It further opens up new research challenges in order to access, evaluate and analyze developer's behavior while contributing to multiple projects in a given period of time.

RQ-3: *What are the contributions made by developers in multiple projects?*

In order to trace developer's activity across multiple projects, we queried and retrieved `owl:sameAs` links of each developer. Later, we computed the development activity of only those developers who participated in multiple projects. In Figure 6, we have summarized the development activity of few developers who contributed to or participated in at-least 2 *Apache* projects.

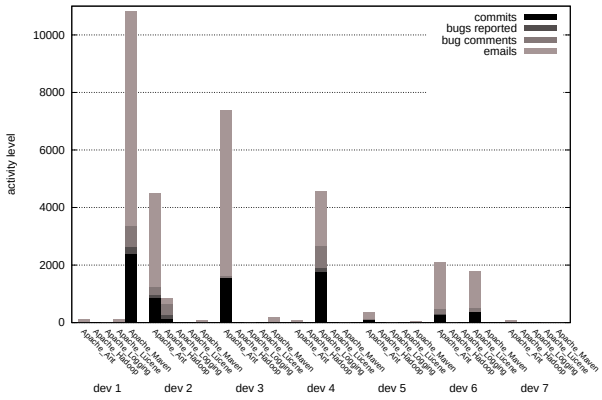

Figure 6. Developer's Contribution in Different Apache Projects.

The figure shows that only few developers have source control commit rights on multiple projects while most developers have commit rights on a single project. Further, the results shows that developers are mostly active in exchanging emails and participating in bugs related discussions comparing to bugs reporting while contributing to multiple projects. There is a possibility that the developers contributed to the source-code of multiple projects by submitting source-code patches to the mailing list or bug tracking repository. We opt out this particular study as it requires interlinking source-code authorship to the developers which is not in the current scope of this paper.

Based on the results, we believe that if the same methodology is applied to a large number of open source projects then we will see a significant amount of developer's activity across multiple projects.

VII. Conclusion

We have motivated and proposed a simple yet effective approach to identify and interlink developer ID(s) within different software repositories of a project as well as across different projects. We have shown that explicitly establishing the interconnection (i.e., Linked Data approach) between multiple ID(s) of a developer helps in integrating, querying and tracking developer activities in software repositories of a single project as well as across multiple projects.

We plan to combine existing developer identification methods with our methodology which might help in improving the interlinking approach, yielding higher quality and quantity links for developer ID(s). We further plan to integrate other potential developer-related open source repositories (i.e., analytical services like Ohloh, code forges etc.) as discussed elsewhere [14]. We also plan to include more *Apache* projects and establish cross-project interlinking among developer ID(s). Based on that, we will study the social behavior of developers who had contributed to different projects in a given period of time. We believe that this will give new insights into the social and technical aspects of open source software developers.

Acknowledgment

The work presented in this paper has been funded by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References

[1] http://ant.apache.org/.

[2] http://hadoop.apache.org/.

[3] http://logging.apache.org/.

[4] http://lucene.apache.org/.

[5] http://maven.apache.org/.

[6] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *MSR '06: Int. Workshop on Mining Software Repositories*, pages 137–143, 2006.

[7] S. Christley and G. Madey. Analysis of activity in the open source software development community. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, HICSS '07, Washington, DC, USA, 2007. IEEE Computer Society.

[8] M. Conklin. Project entity matching across floss repositories. In *proceeedings of 3rd International Conference on Open Source Systems*, 2007.

[9] S. de Sousa, M. Balieiro, J. dos R. Costa, and C. de Souza. Multiple social networks analysis of floss projects using sargas. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, 2009.

[10] S. Diehl, H. C. Gall, and A. E. Hassan. Guest editor's introduction: Special issue on mining software repositories. *Empirical Softw. Eng.*, 14(3):257–261, 2009.

[11] A. E. Hassan. The Road Ahead for Mining Software Repositories. In *Future of Software Maintenance (FoSM) at Int. Conf. on Software Maintenance(ICSM)*, 2008.

[12] A. E. Hassan, A. Mockus, R. C. Holt, and P. M. Johnson. Guest editor's introduction: Special issue on mining software repositories. *IEEE Trans. Softw. Eng.*, 31(6):426–428, 2005.

[13] T. Heath and C. Bizer. Linked data: Evolving the web into a global data space (1st edition). *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, 2011.

[14] A. Iqbal and M. Hausenblas. Integrating developer-related information across open source repositories. In *IEEE 13th International Conference on Information Reuse and Integration (IRI), 2012*, 2012.

[15] A. Iqbal, O. Ureche, M. Hausenblas, and G. Tummarello. LD2SD: Linked Data Driven Software Development. In *Int. Conf. on Software Engineering and Knowledge Engineering (SEKE 09)*, 2009.

[16] G. Klyne, J. J. Carroll, and B. McBride. Resource Description Framework (RDF): Concepts and Abstract Syntax). W3C Recommendation 10 February 2004, RDF Core Working Group, 2004.

[17] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data Knowl. Eng.*, 69(2):197–210, Feb. 2010.

[18] G. Madey, V. Freeh, and R. Tynan. The open source software development phenomenon: An analysis based on social network theory. In *In Americas conf. on Information Systems (AMCIS2002)*, pages 1806–1813, 2002.

[19] A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 13–23, New York, NY, USA, 2008. ACM.

[20] F. Naumann and M. Herschel. An introduction to duplicate detection. *Synthesis Lectures on Data Management*, 2(1):1–87, 2010.

[21] G. Robles and J. M. Gonzalez-Barahona. Developer identification methods for integrated data from various sources. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.

[22] E. Ukkonen. Algorithms for approximate string matching. *Inf. Control*, 64(1-3):100–118, 1985.

[23] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk a link discovery framework for the web of data. In *2nd Workshop about Linked Data on the Web (LDOW2009)*, Madrid, Spain, 2009.