On the Need of Graph Support for Developer Identification in Software Repositories

[position paper]

Aftab Iqbal and **Marcel Karnstedt**
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway, Ireland
firstname.lastname@deri.org

Abstract

Software repositories from open-source projects 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 accurate identification of developers. This is a particular challenge, as developers usually use different IDs in different repositories of one project, but usually there is no kind of dictionary or similar available to map the different IDs to real-world persons. Often, they even use different IDs in the same repository. We show that the few methods suggested so far are not always appropriate to overcome this problem. Further, we highlight related techniques from other areas and discuss how they can be applied in this context. We particularly focus on the idea of applying graph-based methods and argue for the benefits we expect from that.

1 Motivation

In *Software Engineering*, many tools with underlying repositories have been introduced to support the collaboration of 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 [Hassan, 2008]. However, software repositories are commonly used only as record-keeping repositories and rarely for design decision processes [Diehl *et al.*, 2009]. The Mining Software Repositories (MSR) field analyzes the rich information available in these software repositories to discover interesting facts about the software projects [Diehl *et al.*, 2009]. Examples of software repositories are [Hassan *et al.*, 2005] :

1. **source control repositories** store changes to the source code as development progresses;
2. **bug repositories** keep track of the software defects;
3. **archived communications** between project developers record rationale for decisions throughout the life of a project.

Software developers use these repositories to interact with each other or to solve software-related problems. For example, source-code and bugs are quite often discussed on bug tracking systems or project mailing lists. By extracting rich information from these repositories, one can guide decision processes in modern software development. For example, data in a source control repository could be analyzed to extract the authorship information, which could be linked to additional authorship information extracted from author tags of source-code files. This could allow to keep track of which source files were committed by a developer in different periods of time. With "Linked Data Driven Software Development" (LD2SD) [Iqbal *et al.*, 2009], we have introduced a Linked Data-based methodology to relate data across software repositories explicitly and unambiguously. The so created interlinked data sets can be used for querying and browsing the related information that exists in these software repositories.

An excerpt of an exemplary RDF representation of Java source-code using our LD2SD approach is shown in listing 1. Further, an example of RDF representation of a SVN Commit is shown in listing 2.

```
@prefix baetle:
     <http://baetle.googlecode.com/svn/ns/#> .
@prefix ld2sd: <http://ld2sd.deri.org/LD2SD/ns#> .
@prefix : <http://ld2sd.deri.org/data/Java/> .
 :connect a baetle:JavaClass;
baetle:author
     <http://ld2sd.deri.org/data/author/Developer_A>
     ;
ld2sd:imports "java.io.IOException" ,
       "javax.servlet.ServletException" ;
ld2sd:hasMethod :connect#getConnection .
```

Listing 1: An exemplary Java RDFication.

```
@prefix baetle:
     <http://baetle.googlecode.com/svn/ns/#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix : <http://ld2sd.deri.org/data/Svn2RDF/> .
 :275 a baetle:Committing ;
baetle:modified
     <http://svn.deri.org/trunk/org/link/connect.java>
     ;
baetle:author
     <http://ld2sd.deri.org/data/author/Dev_A> .
<http://svn.deri.org/trunk/org/link/connect.java> a
baetle:JavaSource ;
owl:sameAs <http://ld2sd.deri.org/data/Java/connect>
     .
```

Listing 2: An exemplary Subversion RDFication.

The listings indicate one major problem in the context of mining these repositories: often developers use different identities for each software repository and sometimes multiple identities for the same repository, while interacting with these software repositories in different context. Using distinct identities for different repositories makes developers appear as different entities. Hence we need methods and techniques to correctly link the different identities of software developers. This is a main requirement for, among others, being able to keep track of the developers' activities in different software repositories.

There exist only few works discussing this particular issue (see Section 3). However, these works either lack in details or propose simplified solutions. In a series of experiments, we found that the so far proposed methods are not sufficient for all cases. We reflect on these results in Section 4. Thus, we argue that we need more sophisticated identification methods. Specifically, we propose to use graph-based mechanisms. This is based on the observation that all the repository data can be represented as graphs. Moreover, these graphs are related and similar to each other, as they are based on related and similar activities from the same set of developers. Finally, we conclude and propose to use these graph-based and related approaches in Section 5.

2 Identities in Software Repositories

In order to interact with the many software repositories that are part of an open-source project, developers usually require to adopt an identity for each repository. Often, developers use multiple identities for the same repository [Robles and Gonzalez-Barahona, 2005]. Different types of identities that developers use in software repositories discussed by Robles et al. are (also summarized in Table 1):

1. In a source-code file, developers appear with many different identities, such as real life names, email addresses, SVN identifiers and sometimes the combination of real life name and email addresses;
2. Developers usually use multiple email addresses to send mails to the project mailing list. Sometimes the email headers contain the <*name, email*> pair, which helps to link the email address to the developer;
3. To commit source-code on the source control repository, developers use a separate account on the versioning system;
4. Bug tracking systems require an account associated with an email address.

Data Source	**Identities**
Source Code	Name Surname
Source Code	username@domain.com
Source Code	Name[username@domain.com]
Source Code	$subversionID
Mailing List	username@domain.com
Mailing List	Name Surname
Versioning System	$subversionID
Bug Tracking	username@domain.com

Table 1: Identities found in different software repositories [Robles and Gonzalez-Barahona, 2005].

In open-source projects, project outsiders (i.e., contributors) also submit source-code patches to the project mailing list. They cannot contribute code directly to the source control repository, because they are not invited to be part of the core group of developers of that project [Hassan, 2008]. Such contributors quite often mention their names in the javadoc[1] of the source-code files while modifying them to fix a particular bug or adding a new feature request to the project. When core developers finally commit the changes of contributors to the source-control repository, they very often mention the name of the contributor who provided the patch in the summary of the commit (e.g., "*Patch provided by DeveloperABC ...*"). Such information is very useful for analysis if extracted properly and interlinked to the correct developer name.

However, the fact that the sets of users differ between the repositories increases the difficulties of mapping IDs between them. This is a further challenge beside the fact that we have no 1:1 mapping, neither in one repository nor over different repositories. Actually, we often encounter an n:m mapping, e.g., several email addresses might belong to the same developer and multiple developers might use the same email address. An example of the later is an address like `developers@apache.org`. While this might be obvious in this case, it poses a significant challenge for automated approaches for developer identification.

3 Related Work

To the best of our knowledge, there are only a few published works on identifying and relating the different identities that developers use to interact with different tools in the field of software engineering. In [Bird *et al.*, 2006], Bird et al. proposed an approach to produce a list of <*name,email*> identifiers by parsing the emails and clustering them. The clustering algorithm to measure the similarity between every pair of IDs is based on string similarity between names, between emails, between names and emails, etc. Two IDs 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 IDs, some of which we tested on our dataset to validate the effectiveness of these approaches (cf. Section 4.1 and Section 4.2). We found that their approach failed to provide satisfying results in our case. We use this as a motivation to argue for sophisticated approaches, such as graph-based methods.

[Robles and Gonzalez-Barahona, 2005] 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 identities 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 *nsurname@domain.com, name.surname@domain.com* etc. This is an approach based on pre-defined name schemes, a variant that we also evaluate for our case in Section 4.3. Similarly, this method did not provide satisfying results for the data set we investigated. Further, they try to match user names obtained from CVS to email addresses (excluding the domain after "@"). This approach relies on string similarity again.

In general, the problem is related to duplicate detection. 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 IDs of the same developer inside one repository. [Naumann and Herschel, 2010] provides a nice overview of this research direction. Interestingly, the authors also reflect on graph-based duplicate detection. The general idea is to use structural measures of graphs in order to identify different nodes that are similar to each other, wrt. these structural features. Two very similar nodes are likely referring to the same person. Bringing these approaches to the case of multiple graphs forms a basis for several works on network de-anonymisation [Narayanan and Shmatikov, 2009; Wondracek *et al.*, 2010]. The idea is similar: use the structural information from one (partially) known graph and use it to identify similar nodes in another graph, one of which we have only (maybe limited) structural information. Again, those nodes that are very similar are likely referring to the same person. This requires input graphs that are somehow related to each other. Further, the more adversary information we have, the higher accuracy we can achieve. We explore this approach further for our case, where we have several different input graphs, in Section 5.

[1] `http://java.sun.com/j2se/javadoc/writingdoccomments/`

4 Preliminary Results

In this section, we present results gained on data from a representative open-source project: *Apache Tomcat*[2]. We executed some experiments to evaluate the methods described in [Bird *et al.*, 2006] and [Robles and Gonzalez-Barahona, 2005]. In the following, we refer to *email addresses* by using only the term *email*. We gathered data by parsing the emails from *Apache Tomcat* mailing lists starting from 2005 to date. For every email containing a written name *and* the according email, we extracted the *from* and *to* field from the email header to produce a list of <*name,email*> pairs. We parsed 49,927 emails from the mailing list and produced 1261 distinct <*name,email*> pairs. We use these results as a ground proof to validate the different techniques that could be utilized to interlink the different identities of a developer. The reason for using data from a set where we actually know the exact matching is simple: it is the only way to provide some ground truth. It is not possible to extract similar information for other types of IDs. The only, clearly impractical, way would be to ask all involved persons for the different IDs they use. However, we strongly believe that the gained results are significant for these other types of IDs as well. Moreover, the methods we tested were proposed for matching email addresses.

To check the effectiveness of each approach, we first used the approach to find a matching developer name for each email. Then, we computed two versions of precision P and recall R values for each approach using the following equations:

$$P_1 = \frac{\#\text{emails with at least one correct match}}{\#\text{matched emails}} \quad (1)$$

$$R_1 = \frac{\#\text{emails with at least one correct match}}{\#\text{total emails}} \quad (2)$$

$$P_2 = \frac{\#\text{emails with exactly one correct match}}{\#\text{matched emails}} \quad (3)$$

$$R_2 = \frac{\#\text{emails with exactly one correct match}}{\#\text{total emails}} \quad (4)$$

By determining these different types we gain some interesting additional knowledge. For P_1 and R_1 we count a hit if we found at least one correct match, i.e., even if we found more incorrect matches. This provides an "optimistic" quality assessment, as the correct match(es) are found. Anyhow, in reality one would still have to filter out the wrong matches. Thus, we determine a "pessimistic" quality assessment in P_2 and R_2. If there exist correct *and* wrong matches for a given email this is *not* regarded as a hit.

In order to compute the similarities between developer names and email addresses or between two email addresses (see Section 4.1 and Section 4.2), we used the Levenshtein edit distance [Ukkonen, 1985] algorithm, as suggested in [Bird *et al.*, 2006]. The match(es) for an email are those developer names with the lowest distance. Clearly, there is no sense in allowing arbitray large distances, as this would mean that one string can be transformed into a completely other one. Thus, we tested 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

[2]`http://tomcat.apache.org/`

4.1 Similarity between Names and Emails

In the first test we matched email addresses (excluding the domain after "@") and developer names by doing a pairwise comparison. Based on the above described approaches, we computed the precision and recall values, which are shown in Table 2.

Threshold	P_1 **in %**	R_1 **in %**
Maximum length	67.1	57.1
Minimum length	57.4	46.3
Fix Threshold	54.4	30.3
Threshold	P_2 **in %**	R_2 **in %**
Maximum length	24.3	20.1
Minimum length	18.4	15.3
Fix Threshold	48.4	27.2

Table 2: Names-email similarity results.

The results for P_1 and R_1 are actually quite good, where the maximal-length threshold seems to be the best choice. However, the accuracy is not high enough. Moreover, the values of P_2 and R_2 show that these methods produce a lot of false matches. As the fixed threshold seems to be the most "stable" in that context, we can learn that high thresholds tend to match very different strings – and thus create more false positives. In general, all methods are not very well suited for practice on a data set like ours.

4.2 Similarity between Emails

Besides names and emails, it is very likely that email addresses are textually similar to each other if they belong to the same developer. Thus, in the second test, we computed the pairwise similarities between email addresses (excluding the domain after "@") and determined match(es) for one email by choosing those with the lowest distance. This was also suggested in [Bird *et al.*, 2006]. The resulting precision and recall values are shown in Table 3.

Threshold	P_1 **in %**	R_1 **in %**
Maximum length	3.2	3.2
Minimum length	3.1	3.0
Fix Threshold	7.2	2.4
Threshold	P_2 **in %**	R_2 **in %**
Maximum length	2.0	2.0
Minimum length	1.5	1.5
Fix Threshold	3.5	1.3

Table 3: Email similarity results.

Clearly, this idea does not work at all for our data. The low values show that assuming similarity between emails in order to find matches is an absolutely inappropriate solution for our case.

4.3 Matching based on Name Schemes

More likely, email addresses are built from the "real life" name of a developer. As suggested by [Robles and Gonzalez-Barahona, 2005], in the third test we tried to identify matching developer names for emails (excluding the domain after "@") by checking the relation between them based on different name schemes. For example, `aftab.iqbal@deri.org` matches to `Aftab Iqbal` based on the name scheme *name.surname*. Based on our observations on different open-source projects we selected different name schemes, which developers quite often used to build their email addresses : *name.s* (e.g., *aftabi*), *n.surname* (e.g., *aiqbal*), *n.s* (e.g., *ai*), *na.su* (e.g., *afiq*), *name.surname* (e.g., *aftabiqbal*), *e.surname* (e.g., *biqbal*) and *name* (e.g., *aftab*). Note that the dot in a name scheme is only for illustration purposes – in the actual matching we do not use dots and ignore them in emails if present. Precision and recall as computed for each of the name scheme are shown in Table 4.

NameScheme	P_1 in %	R_1 in %
name.s	83.0	2.0
n.surname	96.0	9.2
n.s	33.3	1.0
na.su	50.0	0.2
name.surname	99.4	14.0
e.surname	100.0	0.6
name	72.0	3.5

NameScheme	P_2 in %	R_2 in %
name.s	65.5	2.0
n.surname	94.0	9.0
n.s	5.0	0.1
na.su	0.0	0.0
name.surname	99.4	14.0
e.surname	100.0	0.6
name	72.0	3.5

Table 4: Namescheme - email results.

To our surprise, most of the schemes result in very high precision values, for both types of quality measures. This means that if we find a match with these techniques, it is very likely a correct match. However, the in contrast very low recall values show that this method is capable of identifying only a handful of all matches. Thus, on its own, it is also not suited for our use case.

Summarizing, we can state that all three tested approaches do not prove to be suited for our case. As we have some good results in parts, it seems to be promising to combine these techniques with some more advanced approaches. By inspecting the precision and recall values, we can conclude that combining the methods among themselves is not promising. Thus, in the following section, we discuss graph-based methods to complement them.

5 Graph Support: A New Way of Developer Identification

As briefly mentioned in Section 3, there are several approaches that tackle a problem similar to the one discussed here on the basis of graphs. The main principle is that if we have two related graphs, we can use the structural features of nodes from one graph to reflect on the nodes from the other graph. This is particularly efficient in the context of social network graphs [Narayanan and Shmatikov, 2009; Wondracek *et al.*, 2010], i.e., graphs built from social relations between persons.

The approach suggested by us builds on the same principle. Out of the different repositories, we can construct several graphs: developer-to-file relations, developer-to-developer relations (e.g., working on the same files), email-to-email conversations, bug-to-developer relations, etc. While not all of these graphs are from social interactions, they are clearly all related to each other. Using some base knowledge, for instance gained from the name scheme methods described above, we can try to identify pairs of matching nodes. In order to map SVN IDs to developer names we could for instance build "file fingerprints" (similar to the group fingerprints in [Wondracek *et al.*, 2010]) and compare them. There are plenty of different options. The main task is to identify the kind of relations that connect two or more graphs.

Unfortunately, we are currently restricted in the graphs we can use. This is mainly due to the fact that we have ground truth only for emails and developer names. The only graphs we can extract from the *Apache Tomcat* project containing both types of IDs are email-to-email conversations and developer-to-developer relations (based on common files). However, we found that both graphs are similar in certain structural features, such as centrality values and degree distribution. This is quite intuitive, as developers that handle more classes (central position, higher degree) can be assumed to communicate more by email. In the very near future we plan to develop according methods for developer identification based on this observation. Later, we plan to extend these approaches to all types of graphs and IDs we find in the open-source software repositories. Moreover, we believe it is promising to extend the raw graphs we find by graphs derivated from other knowledge. For example, we will analyse the actual content of email conversations. Using according NLP techniques, we will very likely find more relations between developers and other mail users.

With confidence, we could show that existing methods are not suited for all cases. With the experiences from duplicate detection and de-anonymization, we strongly believe to have identified the right way for sophisticated developer identification. We are looking forward to explore this very interesting and relevant field of research in more detail in very near future.

Acknowledgements

This material is based upon works jointly supported by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and under Grant No. 08/SRC/I1407 (Clique: Graph & Network Analysis Cluster)

References

[Bird *et al.*, 2006] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *MSR '06: Int. Workshop on Mining Software Repositories*, pages 137–143, 2006.

[Diehl *et al.*, 2009] Stephan Diehl, Harald C. Gall, and Ahmed E. Hassan. Guest editor's introduction: Special issue on mining software repositories. *Empirical Softw. Eng.*, 14(3):257–261, 2009.

[Hassan *et al.*, 2005] Ahmed E. Hassan, Audris Mockus, Richard C. Holt, and Philip M. Johnson. Guest editor's introduction: Special issue on mining software repositories. *IEEE Trans. Softw. Eng.*, 31(6):426–428, 2005.

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

[Iqbal *et al.*, 2009] 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.

[Narayanan and Shmatikov, 2009] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *SP '09: IEEE Symposium on Security and Privacy*, pages 173–187, 2009.

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

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

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

[Wondracek *et al.*, 2010] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. A practical attack to de-anonymize social network users. Technical report, 2010. `http://www.iseclab.org/papers/sonda-TR.pdf`.