Developer identification methods for integrated data from various sources

Gregorio Robles, Jesus M. Gonzalez-Barahona
{grex, jgb}@gsyc.escet.urjc.es
Grupo de Sistemas y Comunicaciones
Universidad Rey Juan Carlos
Madrid, Spain

ABSTRACT

Studying a software project by mining data from a single repository has been a very active research field in software engineering during the last years. However, few efforts have been devoted to perform studies by integrating data from various repositories, with different kinds of information, which would, for instance, track the different activities of developers. One of the main problems of these multi-repository studies is the different identities that developers use when they interact with different tools in different contexts. This makes them appear as different entities when data is mined from different repositories (and in some cases, even from a single one). In this paper we propose an approach, based on the application of heuristics, to identify the many identities of developers in such cases, and a data structure for allowing both the anonymized distribution of information, and the tracking of identities for verification purposes. The methodology will be presented in general, and applied to the GNOME project as a case example. Privacy issues and partial merging with new data sources will also be considered and discussed.

1. INTRODUCTION

Most research in the area of mining software repositories has been performed on a single source of data. The reason for this is that tools are usually targeted towards accessing a specific kind of data, which can be retrieved and analyzed uniformly. Data mining for control versioning systems [11], bug-tracking systems, mailing lists and other sources is currently state of the art. The focus of these studies is more on the analysis than in the data extraction process, which can be automated, as has already been discussed [2, 9].

However, there is a wide interest in considering data from several sources and integrating them into a single database, getting richer evidence from the observed matter [5]. The data gathered following this approach can be used for studying several kinds of artifacts relevant to the software development process, such as source code files or, as we will discuss in this paper, developers.

As an example of the usefulness of this approximation, let's consider collaboration in libre software[1] projects, which is an active research field. Libre software is produced in part (in many cases a large part) by volunteers, which makes it difficult to predict the future evolution. However, it has at least in some cases produced high-quality software, used by millions of persons around the world. It has been shown that this collaboration follows a Pareto law for commits [11], source code contributions [4], bug reports [8] or mailing list posts [6]; i.e. a small amount of developers of around 20% is responsible for a huge amount of the produced artifacts (around 80%). But although this research on different sources coincide in results, there is still no evidence of coherence. In other words, although it is known that the Pareto distribution appears in several data sources for a given project, are the most active actors for each of those sources (mailing lists, code repositories, bug report systems, etc.) the same ones?

In the specific case of merging information about developers from different repositories, the main difficulty is caused by the many identities that they use from repository to repository, and even for the same one, making tracking difficult. That is the reason why we need methods and tools that can find the different identities of a given developer. These methods, and the data they produce, should be designed to be sharable among research groups, not only for validation purposes but also for enabling the merging of partial data obtained by different teams from different sources.

In general, any study considering individuals in libre software projects, even when using a single data source, is sensible to identity variety. Before performing any analysis on the data set, it is necessary to merge the identities corresponding to the same person. This is for instance the case in the promising case of clustering [3] and social network analysis [7], which are trying to get insight in the structure of libre software projects.

The structure of this paper is as follows. The next section deals with the kinds of identities which are usually found in software-related repositories. The third section is devoted to the extraction of data, its structure and verification. Section

MSR'05, May 17, 2005, St. Louis, Missouri, USA.
Copyright 2005 ACM 1-59593-123-6/05/0005 ...$5.00.

[1] Through this paper we will use the term "libre software" to refer to any code that conforms either to the definition of "free software" (according to the Free Software Foundation) or "open source software" (according to the Open Source Initiative).

four deals with heuristics for matching identities. Handling data about developers raises some privacy concerns, which are discussed in the fifth section, including some suggestions and solutions for sharing data without violating anonymity. We finish the paper with a section on conclusions and further work. We also include two appendixes, one with some results of applying the methodology to some GNOME repositories, and the other to post-matches analysis.

2. IDENTITIES IN SOFTWARE REPOSITORIES

Libre software developers, or more broadly, participants in the creation of libre software (from now on actors) usually interact with one or more Internet-based systems related to the software production and maintenance, some of which are depicted in Figure 1. These systems usually require every actor to adopt an identity to interact with them. This identity is usually different for every system, and in some cases a given author can have more than one identity for the same system, sometimes successive in time, sometimes even contemporary.

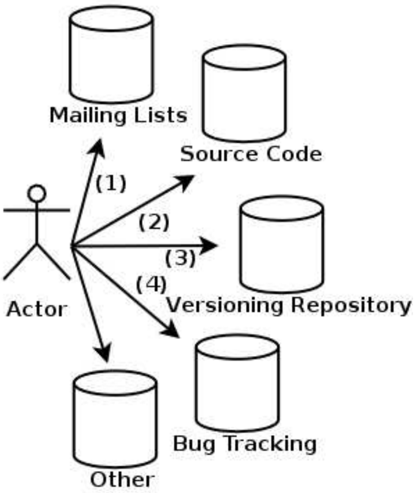

Figure 1: Different systems with which an actor may interact.

Some kinds of identities are the following (summarized in Table 1):

- An actor may post on mailing lists with one or more e-mail addresses (some times linked to a real life name).
- In a source file, an actor can appear with many identities: real life names (such as in copyright notices), e-mail addresses, RCS-type identifiers (such as those automatically maintained by CVS), etc.
- The interaction with the versioning repository occurs through an account in the server machine, which appears in the logs of the system.
- Bug tracking systems require usually to have an account with an associated e-mail address.

Other sources may include entries in weblogs, forums, blogs, etc. Although they are not considered in this study, the approach proposed could easily include them.

Type	Data Source	Primary Identities
(1)	Mailing lists	username@example.com
(1)	Mailing lists	Name Surname
(2)	Source Code	(c) Name Surname
(2)	Source Code	(c) username@example.com
(2)	Source Code	$id: username$
(3)	Versioning System	username
(4)	Bug Tracking	username@example.com

Table 1: Identities that can be found for each data source.

Given the various identities linking an actor to his actions on a repository, our goal is to determine all which correspond to the same real person. Basically we can classify these identities in two types: primary and secondary.

- Primary are mandatory. For instance, actors need an e-mail address to post a message to a mailing list. Mailing lists, versioning system and bug tracking system require to have at least a mandatory identity in order to participate (although in some exceptional cases this can be done anonymously). Source code does not have primary identities, except in some special projects where the copyright notice or some other authorship information is mandatory.
- Secondary are redundant. For instance, actors may provide their real-life name in the e-mails they send, but this is not required. Secondary identities usually appear together with primary identities, and may help in the identification process of actors.

Note that the relationships between actors and repositories have not to be unique: an actor could have one or more different identities in any repository. Even in cases such as CVS repositories, a given actor may change the username of his account, and of course the same actor could have different usernames in different CVS repositories.

3. DATA FETCHING, STRUCTURE AND VERIFICATION

Figure 2 shows a glimpse of the data structures used to learn the identities that correspond to the same person in several data sources.

All the identities are introduced into the database in the Identities table. This table is filled by directly extracting identities (using heuristics to locate them) from software-related repositories. Besides the identity itself, this table stores identifiers for the repository (data source) where it was found, which could be of value not only in the latter matching process, but also for validation and track-back purposes. The kind of identity (login, email address, “real name”) is also stored, to ease the automatic processing. Hashes of identities are added to provide a mechanism which can be used to deal with privacy issues, as will be described in a later subsection.

When extracting identities, sometimes relationships among them can be inferred. For instance, a real name can be next to an e-mail address in a From field in a message. Those relationships are captured as entries in the Matches table, which will be the center of the matching (identification of identities of the same person) process. The 'evidence' field in this table provides insight about every identified match. As the process we are performing is mostly automatic, the value of 'evidence' will contain the name of the heuristic that has been used. This will include automatic heuristics, but also human inspection and verification. Sometimes, the information is not enough to ensure that the match is true for sure, and that is the reason why a field showing the estimated probability has been added. Fields that have been verified by humans with absolute certainty will be assigned a probability of 1.

With the information stored in Identities and Matches, the identification process may begin. Unique actors are identified with information in Matches, filling the Identifications table, and choosing unique person identifiers. Other information in the Persons table can be filled directly with data from the repositories or from other sources.

4. MATCHING IDENTITIES IN MORE DETAIL

We will usually have many identities for every actor. For instance, we can have name(s), username(s) and e-mail address(es). Every actor considered will have at least one of them, although possibly he may be identified with several, as is shown in Figure 3.

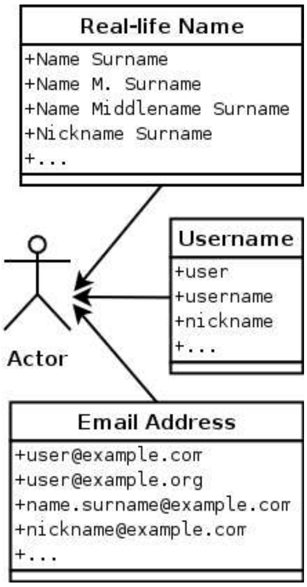

Figure 3: An actor with three different kinds of identities

Our problem is how to match all the identities that correspond to the same actor. In other words, we want to fill the Matches table with as much information as possible (and as accurate as possible). As already mentioned this is done using heuristics. Let's expose some of them with some detail:

- In many cases it is common to find a secondary identity associated to a primary one. This happens often in mailing lists, source code authorship assignments and bug tracking system. In all these cases, the primary identity (usually an e-mail address) may have a 'real life' name associated to it. Consider, for instance, *Example User <username@example.com>*, which implies that *Example User* and *<username@example.com>* correspond to the same actor. GPG key rings can also be a useful source of matches. A GPG key contain a list of e-mail addresses that a given person may use for encryption and authentication purposes. GPG is very popular in the libre software community and there exist GPG servers that store GPG keys with all these information.

- Sometimes an identity can be built from another one. For instance, the 'real life' name can be extracted in some cases from the e-mail username. Many e-mail addresses follow a given structure, such as *nsurname@example.com*, *name.surname@example.com* or *name_surname@example.com*. We can easily infer in those cases the 'real life' name of the actor.

- In many cases one identity is a part of some other. For instance, it is common that the username obtained from CVS is the same as the username part of the e-mail address. This can be matched automatically, and later verified by other means. This is one of the more error-prone heuristics, and is of course not useful for very popular usernames like 'joe'. But despite these facts, it has proven to be very useful.

- Some projects or repositories maintain specific information that can be used for matching (for instance, because a list of contributors is maintained). As an example, the KDE project maintains a file which lists, for every person with write access to the CVS, his 'real life' name, his username for CVS and an e-mail address. Other similar case are developers registered in the SourceForge.net platform, who have a personal page where they may include their 'real life' name.

Of course this is not an exhaustive list, and combinations of the described heuristics can be used. For instance, a mixed approach could benefit from the data in Changelog files [1] for finding identity matches.

Usually, the fraction of false positives for matches can be minimized by taking into account the project from which the data was obtained. If we have a 'joe' entry as username for the CVS repository in an specific project, and in that same project we find somebody whose e-mail address is joe@example.com (and no other e-mail address that could be suspicious of being from a 'joe') then there is a high probability that both are identities of the same actor.

In any case, the fraction of false positives will never be zero for large quantities of identities. Therefore, some heuristics are specifically designed for cleaning the Matches table (eliminating those entries which are not correct, despite being found by an heuristic) and verification, including human verification. In some cases, the help from an expert that knows about the membership of a project, for instance, should be of great help.

But even after cleaning and verification, some matches will be false, and some will be missing, which can cause problems. However, since we are interested in using the

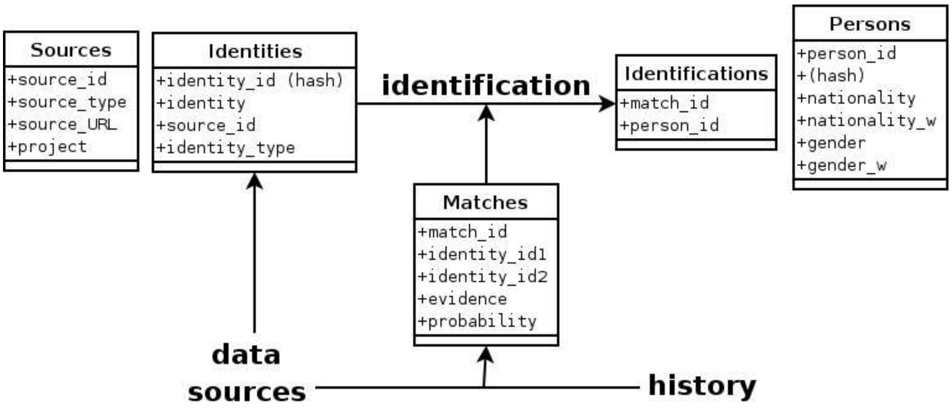

Figure 2: Main tables involved in the matching process and identification of unique actors

collected data for statistical purposes, this is not a big issue provided the error rate is small enough.

5. PRIVACY ISSUES

Privacy is of course an important concern when dealing with sensible data like this. Although all the information used is public, and it hardly contains any private data, the quantity and detail of the information available for any single developer after processing may cause privacy problems. Therefore, we have devised a data schema which allows both for the careful control of who has access to linking data to identified real persons, and for the distribution of information preserving anonymity. In the latter case, the information can be distributed in such a way that real persons are not directly identifiable, but new data sets can be, however, combined with the distributed one. This will for sure allow for a safe exchange of information between research groups.

For this purpose, the hashes of identities serve as a firewall. They are easy to compute from real identities, but are not useful for recovering them when only the hashes are available. Therefore, the Matches, Identifications and Persons tables can be distributed without compromising the real identities of developers as a whole. However, new data sets can be combined. Assuming a research group has a similar schema, with some identities found, the corresponding hash can be calculated for any of them and it may be looked up in the Matches table. Of course this will not be useful in many cases for finding new matches, but it would always allow to link an identity (and the data associated with it) to an actor in the Persons table. Therefore, any development data distributed using hash identities instead of developer names can be safely shared (but see below).

Although hashes will make it impossible to track real persons from the distributed data, it is still possible to look for certain persons in the data set. By hashing the usual identifiers of those persons, they can be found in the Matches table, and their identity is thus discovered. That is the reason why although distributing hashes to other research groups under reasonable ethical agreements is acceptable, probably it is not to do the same for anyone.

To avoid this problem, our schema has still a second level of privacy firewall: the person identifier in the Persons table. This identifier is given in such a way that it cannot be used in any way to infer the identities of an actor without having access to the Identifications table. Therefore it is enough to key all development data with this person identifiers, and distributing only the Persons table in addition to that data to ensure the full privacy of the involved developers.

Of course, even in this latter case somebody could go to the software repositories used to obtain the data, and try to match the results with the distributed information. But this is an unavoidable problem: a third party can always milk the same repositories, and obtain exactly the same data, including real identities. In fact, this is the basis of the reproducibility of the studies.

6. CONCLUSIONS AND FURTHER WORK

Actors in libre software projects may use many different identities when interacting with different systems related to the development (and even with just a single one). When studying repositories related to libre software development it is very important to find those corresponding to the same person, so that actions can be assigned to the corresponding actor.

In this paper we have presented a design for dealing with this problem, and a methodology, based on heuristics, to identify as accurately as possible the different identities of the involved actors. For that, we use information stored in the repositories, and rely on some properties of the identifiers. This information can also be used to infer some personal information, such as the gender or the nationality (as is shown in appendix).

We have also discussed how privacy issues can be dealt with in our schema, including the distribution of anonymized information about the development, and have presented some results of performing the described study on some repositories of the GNOME project (in appendix).

We are currently testing our approach with larger data sets from several projects at once, and also starting to use it for sharing development data with other research groups. In the future, we are planning to include the functionality described in our GlueTheos tool [10], and to use it widely to obtain estimations of the number of people involved in libre software development, and their activities. We expect to use this data in combination with data from surveys and other sources to get a more complete view of the libre software development landscape.

7. ACKNOWLEDGEMENTS

This work has been funded in part by the European Commission, under the CALIBRE CA, IST program, contract number 004337, by the Universidad Rey Juan Carlos under project PPR-2004-42 and by the Spanish CICyT under project TIN2004-07296.

8. REFERENCES

[1] A. Capiluppi, P. Lago, and M. Morisio. Evidences in the evolution of os projects through changelog analyses. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, 2003.

[2] D. German and A. Mockus. Automating the measurement of open source projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, USA, 2003.

[3] R. A. Ghosh. Clustering and dependencies in free/open source software development: Methodology and tools. *First Monday*, 8(4), Apr. 2003.

[4] R. A. Ghosh and V. V. Prakash. The orbiten free software survey. *First Monday*, 7(5), May 2002.

[5] J. M. Gonzalez-Barahona and G. Robles. Getting the global picture. In *Proceedings of the Oxford Workshop on Libre Software 2004*, Oxford, UK, June 2004.

[6] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: Gnome. *Information Systems Journal*, 12(1):27–42, 2002.

[7] L. Lopez, J. M. Gonzalez-Barahona, and G. Robles. Applying social network analysis to the information in cvs repositories. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburg, UK, 2004.

[8] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of Open Source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002.

[9] G. Robles, J. M. Gonzalez-Barahona, J. Centeno, V. Matellan, and L. Rodero. Studying the evolution of libre software projects using publicly available data. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pages 111–115, Portland, USA, 2003.

[10] G. Robles, J. M. Gonzalez-Barahona, and R. A. Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburg, Scotland, UK, 2004.

[11] G. Robles, S. Koch, and J. M. Gonzalez-Barahona. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*, Edinburg, Scotland, UK, 2004.

APPENDIX

A. A CASE STUDY: GNOME

To debug and complete our methodology, we have applied it to the data from several real libre software repositories. One of the most complete studies we have performed to date has been on the GNOME project, retrieving data from mailing lists, bug tracking system (including bug reports and comments) and from the CVS repository. Next, we offer some results from this study:

- 464,953 messages from 36,399 distinct e-mail addresses have been fetched and analyzed.
- 123,739 bug reports, from 41,835 reporters, and 382,271 comments from 10,257 posters have been retrieved from the bug tracking system.
- Around 2,000,000 commits, made by 1,067 different commiters have been found in the CVS repository.
- From these data, 108,170 distinct identities have been identified.
- For those distinct identities, 47,262 matches have been found, of which 40,003 were distinct (therefore, our Matches table contains that number of entries).
- Using the information in the Matches table, we have been able of finding 34,648 unique persons.

This process has been statistically verified by selecting a sample of identities, looking by hand for matches and comparing the results to the corresponding entries in the Matches table. Currently we are completing the Persons table, and performing gender and nationality analysis.

B. AUTOMATIC (POST-IDENTIFICATION) ANALYSIS

The reader has probably noted that the Persons table in Figure 2 includes some fields with personal information. We have devised some heuristics to infer some of them from data in the repositories, usually from the structure of identities. For instance, nationality can be guessed by several means:

- Analyzing the top level domain (TLD) of the various e-mail addresses found in the identities could be a first possibility. The algorithm in this case consists of listing all e-mail addresses, extracting the TLD from them, rejecting those TLD that cannot be directly assigned to a country (.com, .net, .org, etc.) or those who are from "fake" countries (.nu, etc.), and finally looking at the remaining TLDs and count how often they occur. The TLD that is more frequent gives a hint about the nationality of the person. Of course this heuristic is specially bad for US-based actors (since they are not likely to use the US TLD), and for those using .org or .com addresses, quite common in libre software projects.
- Another approach is to us whois data for the second level domain in e-mail address, considering that the whois contact information (which includes a physical mail address) is valid as an estimator of the country of the actor. Of course, this is not always the case.

Other case example of information which can be obtained from identities is the gender. Usually we can infer the gender from the name of the person. However, in some cases it depends on the nationality, since some names may be assigned to males in one country and to females in another. This is for instance the case for Andrea, which in Italy is a male name while in Germany, Spain and other countries is usually for females.