Project Entity Matching across FLOSS Repositories

Megan Conklin
Elon University Campus Box 2126, Elon, NC, 27244 USA
mconklin@elon.edu
WWW home page: http://facstaff.elon.edu/mconklin

Abstract. Much of the data about free, libre, and open source (FLOSS) software development comes from studies of code repositories used for managing projects. This paper presents a method for integrating data about open source projects by way of matching projects (entities) and deleting duplicates across multiple code repositories. After a review of the relevant literature, a few of the methods are chosen and applied to the FLOSS domain, including a simple scoring system for confidence in pairwise project matches. Finally, the paper describes limitations of this approach and recommendations for future work.

1 Introduction

Free, libre or open source software (FLOSS) development teams often use centralized code repositories to help manage their project code, to provide a place for users to find the product, and to organize the development team. Although many FLOSS projects host their own code repository and tools, many projects use the tools hosted at a third-party web site (such as Sourceforge[1], ObjectWeb[2], or Rubyforge[3]). These code forges provide basic project/team management tools, as well as hosted space for the source code downloads, a version control system, bug tracking software, and email mailing lists. There are also directories of FLOSS software (such as Freshmeat[4] and the Free Software Foundation[5] directory) that try to gather into one convenient place material about projects interesting to a particular community.

Much software development research has been focused on gathering metrics from code repositories. Many aspects of the repository-based software development

[1] http://www.sf.net
[2] http://forge.objectweb.org
[3] http://www.rubyforge.org
[4] http://www.freshmeat.net
[5] http://directory.fsf.org

Please use the following format when citing this chapter:

Conklin, M., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 45–57.

process have been studied in depth, and repository data collection is important for these studies (see [2] for background). The FLOSSmole project [5] was created to consolidate metadata and analyses from some of these repositories and directories into a centralized collaboratory for use by researchers in industry and academia. As of this writing, FLOSSmole includes data and analyses from Sourceforge, Freshmeat, Rubyforge, ObjectWeb, and the Free Software Foundation (FSF) directory of free software. One of the challenges mentioned in [2] in creating this kind of collaboratory is in integrating the data from these various sources. When integrating project data from multiple sources, we must first identify which project pairs are matches; in other words, we want to find out which projects are listed on multiple forges. For example, is the *octopus* project on ObjectWeb the same as the *octopus* project on Sourceforge or the project also called *octopus* on Freshmeat? If we can determine a heuristic for determining whether a project pair is a match, then can we automate the matching process?

The focus of this paper is entity matching (and duplicate identification) as applied to the domain of FLOSS projects. Section 2 outlines some terminology from the study of data integration problems and gives a background of entity matching algorithms. Section 3 describes the FLOSS domain in terms of entities and duplicates. Section 4 gives an example of applying some of the algorithms for entity matching to this domain. Section 5 outlines limitations of this work and gives recommendations for future study.

2 Entity Matching Background

The act of integrating multiple data sets and finding the resulting duplicate records ("matches") is nearly as old as database processing itself. In practice and in the literature, this set of processes is known by many names: merge/purge, object identification, object matching, object consolidation, record linkage, entity matching, entity resolution, reference reconciliation, deduplication, duplicate identification, and name disambiguation. The terms *entity matching* and *duplicate identification* will be used throughout this paper.

Within the larger activity of data integration, the act of matching entities or identifying duplicates is not to be confused with the act of *schema reconciliation*. Schema reconciliation refers to the act of matching up columns or views in different data sources, and using data or metadata to make the match. For a trivial example, suppose a field in Table A is called *url* but it is called *home_page* in Table B. To resolve these schemas, the analyst could create a global schema or view that encapsulates both underlying schemas. This task can be done manually, or can be automated through various machine learning techniques [1,4,8]. Schema reconciliation and entity matching are related, but not identical, tasks of data

integration. Most often the schema reconciliation will happen first, followed by the "merge" task, and finally by the eventual "purge" of duplicate data.

2.1 Agree/Disagree and Frequency-Based Matching

The simplest and oldest form of entity matching is the simple agree/disagree method: take two data sets A and B and compare them pairwise for matches based on one or more attributes. The pairs will either agree or disagree on zero or more of the attributes, and thus a weight for the match can be determined.

To improve agree/disagree entity matching, early research relied on frequencies of values to determine the probability of a match (see [10] and [7] for brief explanations of this work). Frequency matching asserts an important premise: that two rare values are more easily and accurately matched than two common values. The example given in [10] is for two records listing the name Zbigniew Zabrinsky, two records listing the name James Smith, and any two records with first and last names. The two records for Zbigniew Zabrinsky are likely to be more easily and accurately matched than James Smith due to the rarity of the field values.

2.2 Disjoint Sets

In [4], the authors consider the problem of how to match 'person' records using disjoint attributes and a 'typical person' profile. For instance, the example given in the paper is that the two records {Mike Smith, age 9} and {Mike Smith, salary $200,000} are not likely to be the same person based on a profile indicating that a typical person with an annual salary of $200k is older than 9 years. The authors compare their system to a traditional agree/disagree system of matching, and show that disjoint attributes can be effective if paired with shared attributes.

2.3 Confidences

Numerous authors, including recently [6], consider how to merge records when a confidence measure has been added to the results of a prior merge process. In their description, confidences (also called weights or scores) usually measure either (a) the level of user-defined "belief" in the data, or (b) the amount of "accuracy" the user thinks is present in that particular merged record. In their paper, the authors ask: what is the best (most efficient, least work) way to match and merge records, given a confidence measure on each record? These authors do not discuss in [6] how to actually calculate a confidence value, but this is one of our concerns in Section 4.

3 Entity Matching Methods for FLOSS Data

This portion of the paper describes the way each of these entity matching methods can be applied to integrate disparate sets of projects from the FLOSSmole project.

First, by way of introduction to the FLOSSmole data, Table 1 shows a partial list of the project attributes available for each of the repositories/forges in FLOSSmole at the time of this writing. These project attributes are the most likely candidates for the job of matching projects. (There are dozens of other attributes about each project in FLOSSmole, such as *registration date* or *project status* or *number of downloads*, but these are not likely to be helpful in matching projects across repositories.)

Attribute	**Forge** *Sourceforge, Freshmeat, Rubyforge, Objectweb, Free Software Foundation*				
	SF	FM	RF	OW	FSF
Short Name (unixname)	X	X	X	X	X
Long Name	X	X	X	X	X
Description	X	X	X	X	X
URL	X	X	X	X	X
License Type(s)	X	X	X	X	X
Programming Language(s)	X	X	X	X	X
Operating System(s)	X	X	X	X	
Topic(s)	X	X	X	X	
Intended Audience(s)	X	X	X	X	
User Interface(s)	X	X			X
Environment(s)			X	X	
Developer(s)	X	X	X	X	X

Table 1. Project metadata: relevant attributes for matching projects, (FLOSSmole, Dec, 2006)

Most of these attributes shown in Table 1 are self-explanatory. However, some confusion can arise when differentiating between the *short name* and the *long name* for a project. The short name is usually an internal-to-the-repository name that is given to the project at the time of its creation. Some repositories use this as a sort of primary key for the project in its database. The long name of a project is the more descriptive name for a project. It can change over time, it can include spaces and special formatting characters, and it typically more descriptive than the short name. Values for all of the attributes shown in the list in Table 1 are chosen by the project administrators, and except for short name and long name, they can all be NULL. License type, operating system, topic, audience, interface, and environment can have multiple values.

The next three sections describe a few of the obvious choices for attributes from this list that can be used to establish matches between projects. One choice from Table 1 that may initially look promising is "List of Developers". Since this attribute is actually a list of developers who work on each project, what better way to differentiate or match two projects? (If the list of developers is the same for the two projects, then the two are a match.) The problem with this is that developers are entities themselves, and matching developers between repositories requires an entirely separate list of attributes (developer name, developer email, developer skills, role on project, etc). Section 5 discusses broadening project entity matching to include developers, but the remainder of this paper will exclude developers as entities and will retain the focus on project matching only.

3.1 Matching by URLs

The diagram shown in Figure 1 depicts each forge/directory in FLOSSmole and how many of its projects list *another forge* as the actual hosting home page. For example, in the diagram, the topmost arrow shows 11 projects on the FSF that actually have Rubyforge listed as the home page. The arrow notation is used to show a direction of the relationship (e.g. 10,044 Freshmeat projects show a home page on Sourceforge, but only 4 Sourceforge projects list a Freshmeat home page). Pairs of forges with no URLs in common are not shown. (No Rubyforge projects list ObjectWeb URLs, and vice versa. Also, as is befitting its status as a directory and not a repository, the FSF directory is not listed as the home page of *any* projects from the other repositories, so these empty relationships are not shown in the diagram.)

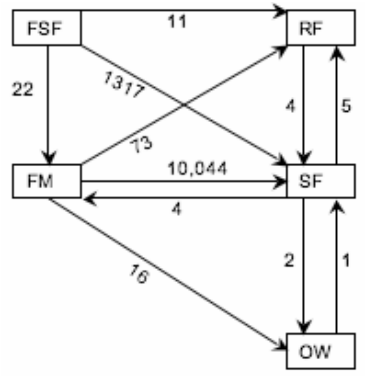

Fig. 1. Number of projects at each repository that list a home page at another repository

3.2 Matching by Project Names

Figure 2 shows the number of short project names shared in common between each pair of projects. For instance, *starfish* is a project listed on both Sourceforge and Rubyforge. On Rubyforge, it is described as a "tool to make programming ridiculously easy", but on Sourceforge the *starfish* project is described as a password management application. There are 470 projects with shared names on Rubyforge and Sourceforge. A similar problem exists between the project names on Sourceforge and ObjectWeb. For example, the project called *octopus* exists on both these forges and appears to be a completely different application: on Sourceforge this is an Eclipse plug-in, but on ObjectWeb *octopus* is an ETL data warehousing tool. Of the 125 applications (total) listed on ObjectWeb, 41 have names that are shared with a Sourceforge project. The Sourceforge project may (as in the case of *lemonldap*) or may not (as in the case of *octopus*) be the same project. On Freshmeat, there also is a project called *octopus*, but this one is a financial trading application.

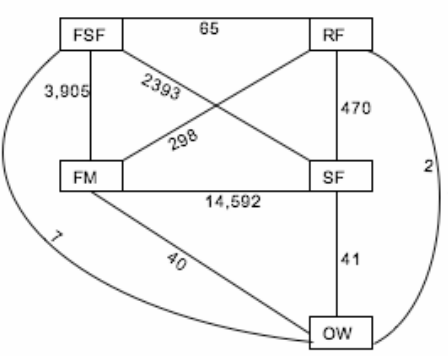

Fig. 2. Number of projects at each repository that share an identical short project name

Most forges require projects to have a unique name (sometimes called the "unixname") within that forge. For example, once a project called *starfish* has been added to Sourceforge, another one cannot be added with the same short unixname. However, multiple projects can have the same "display name"; Sourceforge projects *starfish* and *xstarfish* both have the display name of "starfish". On Sourceforge, 44,112 (39%) of projects have unixnames that are different from their display names (December 2006 FLOSSmole data). Note that the FSF directory has only a requirement for case-sensitive uniqueness in project names. The FSF lists project pages for both ANT (telephony application) and ant (build tool). There are 54 such (ambiguously) named projects listed on FSF.

3.3 Matching by Other Attributes

It may be possible to determine the accuracy of each matched pair further by attempting to match the project owner or developer names, emails, or usernames as in [11]. Or, it may be possible to find a matched pair through the textual description of the project, or through the project license type, the programming language(s), operating system(s), or other metadata about the project. Each of these possible match fields requires that the project administrator has accurately filled in the metadata for his/her project. If the administrator never bothered to fill in the programming language for the project on one or both of the sites where the project is listed, then it will not be possible to disambiguate by finding a match on this item.

Project Attribute	Projects listing at least one	Projects listing none
Programming Language	82,969 (73%)	29,946 (27%)
License Type	84,102 (74%)	28,813 (26%)
Operating System	78,334 (69%)	34,581 (31%)

Table 2. Numbers of Sourceforge projects with and withoug certain attribute data, (FLOSSmole, Dec, 2006)

Table 2 summarizes a few of the most common attribute statistics for Sourceforge projects. It is also interesting to discover that of those 74% of projects that list a license type, over half use the GPL.

3.4 Advanced Methods

In our attempt to match FLOSSmole projects by URL, name, or any other combination of attributes, we are still performing basic agree/disagree entity matching. Our brief review of the database literature on entity matching indicates that these methods do work for some cases, but can be optimized and improved.

3.4.1 Frequency-Based Matching

The first improvement made to the agree/disagree entity matching was to consider how to apply a form of frequency matching on the name field. Recall that [10] explains that rare names (Zbigniew Zabrinsky) are more easily matched than common names (James Smith). "Rare" and "common" are determined by an already-existing set of names and their general frequency rankings in the population. In the case of FLOSS projects, there is no such ranking for software project names, but a corollary might be that projects with dictionary words for names (e.g. the *octopus* and *starfish* examples) are more likely to be non-matches than projects with unusual, non-dictionary names (e.g. *sqlite-ruby* or *lemonldap*). Because there is also a difference between the unique *unixname* and the non-unique *display name* for each project, we ask: which of these fields should be used to consider the frequency match? In Section 4, we answer with "both", but we score the matches differently.

3.4.2 Disjoint Sets

The next improvement was to use the notion of a disjoint set, as in 2.2. by listing which attribute values would likely *never* coexist. Initial ideas included the following possible disjoint sets: {*op_sys*=linux, *prog_lang*=asp}, {*date_registered*<2001, *prog_lang*=C#}. Not only are these rules fairly weak insofar as there are plenty of examples of projects that would violate them for various reasons, but unlike the age/salary information in the example case in 2.2, the number of records in FLOSSmole which match these disjoint sets is likely to be quite small. We conclude that in the FLOSS domain, it is more likely to be the case that duplicates can be found through simpler methods than disjoint sets. This is due to three factors: the low number of valid disjoint set rules we would be able to construct, the difficulty of applying disjoint set rules to our data when so many of the pairs are missing metadata on which these disjoint sets would be based (recall section 3.3), and the low number of duplicates that *would not* be identified by other, simpler methods.

3.4.3 Confidences or Scoring

One final serious consideration in advanced methods of entity matching and duplicate identification was the use of confidences to describe numerically the analyst's degree of belief in the accuracy of the merge/match. How should scoring be done? We worked backwards from our initial assumption that the end goal of this exercise is be able to point from one project to another based on likelihood that they are a match. Thus, we planned to consider each pair in turn, then apply a confidence/match score (based on the heuristics used) to the record to indicate how good the match was. The scoring and results are given in the next section.

4 Application

To apply entity matching methods to project data in FLOSSmole, we assume a set of heuristics and associated weights for calculating whether the items in a pair are a match (Table 3). Match modifiers were determined through trial and error, and based on an intuitive sense of which matching criteria were important.

	Match Modifier
Home Page URLs match	+3.00
Short names match	+2.00
--if yes, is short name in the dictionary (i.e. is it common?)	-1.00
--if not, does Partial Name match?	+0.50
-- if partial name matches, is partial name in dictionary?	-0.25
Textual descriptions tokens match, per token match	+0.10
Long (display) names match	+0.50
Programming language matches, per token match	+0.50
License matches, per token match	+0.50
Other project metadata matches, per token match	+0.50

Table 3. Scoring table for matching pairs of projects

A short example of a table designed to hold the FLOSSmole pairs with their matching scores across multiple repositories might look like Table 4. Higher scores mean the pair is more likely to be a match, but it will be up to an individual analyst to decide where to "draw the line" for what score indicates a match. The highest score is around 8; the lowest score is 0. As shown in the table, the highest score could be higher if more attributes were added. Attributes included are programming language, operating system, and license because these are the fields whose values were most available and easiest to standardize over a variety of repositories. (Compare with attributes like "environment", "interface", or "topic" that are hard to standardize.)

Pair ID	Project Name	Source A	Project Name	Source B	Score
1	phpmyadmin	SF	8001 (phpmyadmin)	FM	6.9
2	octopus	SF	octopus	OW	1.0
3	octopus-ge	SF	octopus	OW	2.6
4	16120 (octopus)	FM	octopus	OW	1.5
5	13902 (ant)	FM	152 (ant)	FSF	4.1
6	sqlite-ruby	SF	sqlite-ruby	RF	6.9

Table 4. Scoring table for matching pairs of projects

Pair 1 shows Sourceforge project *phpmyadmin* matching an identically-named Freshmeat project. These projects share a short name (with low frequency count when compared to a dictionary word: +2), long name (+.5), URL (+3), and license type (+.5). Several key tokens are the same in each description (+.9: MySQL, PHP, Web, administration, alter, drop, database, delete, SQL).

Pair 2 shows Sourceforge project *octopus* with ObjectWeb project *octopus*. The short project names match (+2), but urls are different. The long project names are also different ('Octopus' and 'Enhydra Octopus'). Additionally, because the Sourceforge project *octopus* does not list any project metadata, it can't be matched very well with the ObjectWeb project of the same name using these additional attributes. Finally, these two entities share the dictionary name 'octopus' (-1).

Pair 3 shows the project *octopus-ge* on Sourceforge and project *octopus* on ObjectWeb. These projects share a beginning partial string match, octopus* (+1) but it is a dictionary word (-.5). They share one programming languages (+.5), a license type (+.5), and one operating system (+.5). The textual description of the projects increases the score, since both use the strings 'Enhydra Octopus', 'extraction', 'transformation', 'load*', 'ETL', and 'XML' (+.6). However, a closer read of the textual description field by a human being reveals that the Sourceforge project is actually a graphical editor for the ObjectWeb project. They are related projects, but not the same project. The combination of no ULR score and low scores for the textual matches has (accurately) kept this project from a high score.

Pair 4 shows the attempted match between that same *octopus* project at ObjectWeb but now paired with the *octopus* project at Freshmeat. The projects have the same short name (+2 for similarity, -1 for dictionary), but different URLs, totally different textual descriptions, and share only the license type in common (GPL, +.5). Indeed, manual checking of this result shows that these two projects are not related.

Pair 5 shows the Freshmeat project *ant* matching with the Free Software Foundation project *ant* as follows: short name (+2), url (+3). However, the display names for this project are different ('ant' on FSF and 'apache ant' on FM). In addition, the common dictionary name 'ant' lowers the score somewhat (-1). Note that while there is only one significant matching token in the textual description (the word

"Java", +.1), the entire first sentence of the two projects is identical. This indicates a strong need to refactor the scoring algorithm for textual descriptions.

Pair 6 shows that SQLite-ruby project listed nearly identical information on both Sourceforge and Rubyforge. They share: the project home page (+3), short name (+2), the display name (+.5), one programming language (+.5), the operating system (+.5), and 4 significant text tokens (+.4), yielding a total score of 6.9.

5 Limitations, Recommendations, and Future Work

Based on the application shown in Section 4, entity matching is an interesting exercise, but is certainly problematic. One of the most obvious problems is the scoring modifiers given in Table 2; there is a distinct possibility that a pair of projects could achieve a score of 4.0 by having a partial non-dictionary name match (+.5), five attributes in common (+2.5), and a handful of well-chosen tokens in the textual description (+.5), and yet these projects could be completely unrelated. Yet, it is not enough to simply require a score higher than 5.0 for a match; according to the table, the *ant* project pair on Freshmeat and FSF also received a score of 4.1, and it *is* a legitimate match.

This leads to a discussion of how to set scoring thresholds. Perhaps there could be a "yes" category for projects scoring above a certain value, a "no" category for projects scoring below a certain value, and a middle category for questionable scores. These questionable scores may take human intervention to resolve. It will be necessary to constantly tweak the scoring system and thresholds so that there are not too many false positives, false negatives, or values needing human intervention.

There are also numerous ways to improve the definitions of token matches within textual descriptions. For instance, in the case of *ant*, there were very few singularly meaningful tokens in the textual descriptions, but the description as a whole matched perfectly. The use of dictionary word definitions for frequency matching may need to be refactored also. The *ant* match lost points because of this. Also, should non-dictionary strings that are also common in software development ("lib", "db", "php") be added to the dictionary? Partial matches were also problematic. How should the word be broken: by leading strings, ending strings, or middle-of-word strings? Also, if a project name matches by 14 letters, should that get a higher score than a pair that only matches by three letters? Is it possible that those three letters could be highly significant?

Next, what about multi-way matches? We have given little attention to the problem (as presented in [6]) of how to merge multiple confidence scores after they've been created. Consider a project such as *sqlite-ruby* that appears on Sourceforge, Rubyforge, Freshmeat, and the FSF directory. What is the appropriate

way to integrate its multiple scores? *Sqlite-ruby* is likely to have high scores on all 6 pair combinations, so a simple average might work, but what about a project like *ant* whose scores may vary more?

Section 3 mentioned the possibility of matching projects based on the lists of developers on each project. Before doing this, it would be necessary to use similar entity matching methods to actually match *developer* entities as well. As is so well-described in [9], matching developers also leads to a few additional complexities: "real" emails are most often not available for public lists of developers on code repositories, name matching with developers could be even more complex than matching on names for projects because of similarities in names and spellings, and of course, developer privacy is always a concern when integrating disparate personal data. It is instructive that the authors in [9] do also rely on heuristics to make their matches, and that they limit their matches to a single group of actors in the GNOME project, albeit over numerous data sources within that project (mailing lists, CVS repositories, etc.)

One final recommendation for future work is to remember some of the work being done on sites like Krugle[6], Swik[7], and the Galactic Project Registry[8] to standardize the notion of a project name. Krugle is a source code search engine that actually uses some FLOSSmole data to populate its list of projects. Swik is a wiki of information about individual open source projects; it gets some of its initial information from FLOSSmole as well. The Galactic Project Registry is attempting to put together a plan for being "the One True Known Up-To-Date Source" for project names and DOAP (description of a project) information on each project. Each of these projects probably would benefit from this work in entity matching and duplicate identification across repositories, and perhaps they can contribute to the conversation about the best way to achieve this goal.

6 Acknowledgements

The author would like to acknowledge Charles Irons and Wayne Conley for their important work on the *Other Souths* project at Elon University. Their work implementing *Other Souths* directly influenced the methods designed for this paper.

7 References

1. Batini, C., Lenzerini, M., Navathe, S. (1986). A comparative analysis of methodologies for database schema integration. *ACM Comp. Surveys*, 18:4. 323-364.

[6] http://www.krugle.com
[7] http://www.swik.net
[8] http://gpr.wikiwall.org

2. Conklin, M. (2005). Beyond low-hanging fruit: Seeking the next generation of FLOSS data mining. In *Proc. 2nd Intl. Conf. on Open Source Sys*. Como, Italy. 47-56.

3. Doan, A., Domingos, P., Halevy, A. (2001). Reconciling schemas of disparate data sources: A machine learning approach. In *Proc. of the ACM SIGMOD*. Santa Barbara, CA, USA. 509-520.

4. Doan, A., Lu, Y., Lee, Y., Han, J. (2003). Object matching for information integration: A profiler-based approach. In *Proc. of the IJCAI Workshop on Information Integration on the Web*. Acapulco, Mexico. 53-58.

5. Howison, J., Conklin, M., Crowston, K. (2005). OSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. In *Proc. of the 1st Intl. Conf. on Open Source Sys*. Genova, Italy. 54-59.

6. Menestrina, D., Benejelloun, O., Garcia-Molina, H. (2006). Generic entity resolution with data confidences. In *Proc. of 1st Int. VLDB Workshop on Clean Databases*. Seoul, Korea.

7. On, B-W., Lee, D., Kang, J., Mitra, P. (2005). Comparative study of name disambiguation problem using a scalable blocking-based framework. In *Proc. of the 5th ACM/IEEE-CS Joint Conf. on Digital Libraries*. Denver, CO, USA. 344-353.

8. Rahm, E. and Bernstein, P. (2001). A survey of approaches to automatic schema matching. *VLDB Journal*, 10. 334-350.

9. Robles, G. and Gonzalez-Barahona, J. (2005). Developer identification methods for integrated data from various sources. In *Proc. of the Mining Software Repositories Workshop (MSR2005)*. 1-5.

10. Winkler, W. (1999). The State of Record Linkage and Current Research Problems. Technical Report, Statistical Research Division, US Bureau of the Census.