Apache Commit History in Neo4J Representation

http://sequoia.cs.byu.edu/msr2013data/

Alexander C. MacLean, Charles D. Knutson
Dept. of Computer Science
Brigham Young University
2204 TMCB, Provo, UT 84602
amaclean@byu.net, knutson@cs.byu.edu

***Abstract*—Building non-trivial software is a social endeavor. Therefore, understanding the social network of developers is key to the study of software development organizations. We present a graph representation of the commit behavior of developers within the Apache Software Foundation for 2010 and 2011. Relationships between developers in the network represent collaborative commit behavior. Several similarity and summary metrics have been pre-calculated.**

I. Introduction

Understanding the social interactions at play within a development organization allows for deeper and richer analysis of software development activities. To this end, we constructed graphs that represent the contributors to open source software projects, along with data describing their contributions.

In previous studies authors have constructed similar graphs wherein developers (represented by nodes) are connected to each other if they have modified the same file [5], [12]. In these studies edges are unweighted. In our graphs, edges are weighted using various metrics to reflect the nature and magnitude of the contributions of individual developers. Weighted edges enable more accurate reconstruction of the relationships that exist in the actual organization [8].

II. Definitions

In order to reduce confusion when discussing derived metrics (metrics calculated from other metrics), we define several symbols.

Definition 1: λ_a is the set of files modified by author a within the context of the data set.

Definition 2: $\lambda_{a,i}$ is the set of files modified by author a in a given revision, i.

Definition 3: σ is a function that returns a vector containing measurements for an author's contributions to each file within the data set (for example, $\sigma(\lambda_a)$ would return the vector for author a). σ is a placeholder for any of the file-level metrics described in Section IV-B that return a vector (see Table I).

III. Data

The data presented here represents the commit behavior of Apache Software Foundation (ASF) developers over a two year period (2010 through 2011). It is stored in a Neo4J graph database that relates developers to their respective commits, and commits to the files that they modify (see Figure 3). Many metrics describing the data set have been pre-calculated and stored within the graph (see Section V). For example, the graphs contains edges relating developers based upon their similarity (see Figure 1).

file	measurement
1	23
2	0
3	197
4	2
⋮	⋮

TABLE I
This table illustrates a vector representation of commit behavior for a single developer. Each file in the dataset appears in the vector, whether or not the developer has modified the file. The order of files in the vector is the same for all developers so that they may be compared directly.

Fig. 1. Developer-to-developer connections. Colors indicate top project communities.

Fig. 2. Developer Tenure

IV. Methodology

All data is mined from the ASF Subversion repository via the repository dumps made available at http://svn.us.apache.org/dump/. We then import these dumps into a local instance of Subversion. From the local repository we analyze sliding three month time windows to create graph representations of developer behavior. Thus, our data set contains 22, three-month time window graphs: January through March, 2010, February through April 2010, March through May, 2010, etc.

We chose three month time windows to capture the working behavior of developers. The median tenure for a developer on the Apache HTTP Server Project—a very successful, long-running project—is 3.7 years [9], while the median tenure for a developer in the entire foundation is 1.5 years. Developer tenure is highly skewed towards short tenures (see Figure 2). A quarter of developers spend less than three months actively contributing to an ASF project. Conversely, over the course of several years of development effort, a long-tenured developer may collaborate with many different groups of other developers. Choosing relatively short time windows (three months) allows us to capture the day-to-day collaboration that exists between developers regardless of their overall tenure.

Time window graphs are created in the following steps: 1) graph creation, 2) top project calculation, 3) similarity metric calculation, 4) author statistics calculation, and 5) community finding.

A. Graph Creation

Each graph is created by iterating over all of the revisions in the repository and excluding those that are outside the time window[1]. Then, for each revision included, creation proceeds as follows:

1) If a node has not been created for the author of the revision, create the author node and assign the author metadata.
2) Create a revision node and assign the revision metadata.
3) Link the author node to the revision node with an *AUTHOR_ON_REVISION* relationship.
4) Link the revision node to the file node with an *ADD*, *DELETE*, *MODIFY*, or *REPLACE* relationship.
5) Use the UNIX diff utility to calculate the number of lines added to and deleted from the file, and assign these numbers to the *ADD*, *DELETE*, *MODIFY*, or *REPLACE* relationship.

B. Similarity Metrics

We calculate the Pearson Product Moment Correlation Coefficient, Euclidean Distance, Cosine Similarity, and Extended Jaccard Similarity between developers. For a complete description of the similarity metrics calculated on the the graph, we refer the reader to our previous work [8].

C. Top Project

We calculate the top project for a developer based upon file metrics. A developer's top project is the project for which his or her commits cause the greatest amount of change, as measured by a particular file metric.

D. Author Statistics

Based upon revision information, we calculate the aggregate statistics found in Table II for each author.

E. Community Finding

We group developers into communities using Blondel's modularity metric [1]. Weights between authors are determined using combinations of file and similarity metrics (see Section V-J). For a complete description of this process, see our previous work [8].

V. Schema

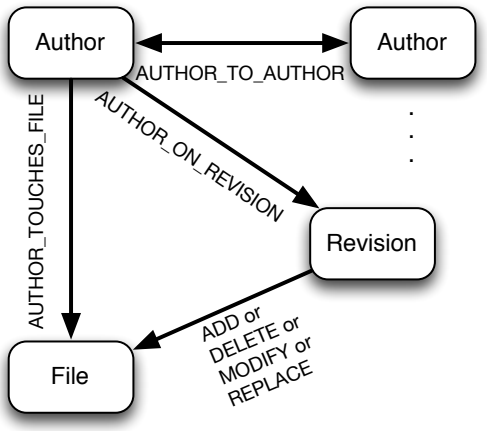

Fig. 3. Data Model

[1]Note: Subversion does not guarantee that revisions are stored in chronological order.

Attribute	Description
name	The user id of the author within the SVN repository.
join-date	The date of the author's first commit to the ASF. This metric is only meaningful if the author joined the project within the time period contained within the the data set.
leave-date	The date of the author's final commit to the ASF. As with *join-date* this date is only meaningful if the author leaves the project within the time period of the data set.
lines-added	The total number of lines the author added to files within the data set.
lines-removed	The total number of lines the author deleted from files within the data set.
lines-modified	The total number of lines the author modified in files within the data set.
number-of-files-touched	The total number of files the author modified in some manner.
average-files-per-commit	The average number of files the author changes per revision.
number-of-file-changes	The total number of times an author has changed any file within the data set (the sum of the number of files changed per revision).
commit-count	The number of revisions an author committed within the data set.
unique-file-count	The number of files an author modified within the data set that were not modified by any other authors.
percent-files-unique	The percentage of the files the author modified that were unique.

TABLE II
SUMMARY *author* NODE ATTRIBUTES.

The data stored in the Neo4J database is comprised of three types of nodes—*author*, *revision*, and *file*—where the type is identified by the attribute *node-type*. The nodes are connected by seven types of relationships: *AUTHOR_ON_REVISION*, *ADD*, *DELETE*, *MODIFY*, *REPLACE*, *AUTHOR_TOUCHES_FILE*, and *AUTHOR_TO_AUTHOR*. In the following sections we describe each of the three node types and the seven relationship types along with the attributes that they store. Certain derived metrics are calculated for multiple base metrics. These metrics are named such that for each base metric an attribute of the form *metric-name-[base-metric]* is created, where *[base-metric]* is replaced by the name of the base metric.

A. Author Node

An *author* node (*node-type* = *author*) represents a unique contributor to the ASF repository. The node contains summary information about the overall behavior of the author (see Table II), categorical information indicating community membership (see Sections V-A1, V-A3, and V-A4), and summary information about categorical data (see Section V-A2). All summary statistics relate only to the time window represented by the graph.

1) top-project-[file-metric]: Top project attributes represent the top project for an author for a given *file-metric* as described in Section IV-C.

2) top-project-[file-metric1]#[file-metric2]: Aggregate statistics for a particular *file-metric1* are indicated by *file-metric2*, where the value represents the sum of *file-metric2* values for all activity within the top project for this author. For example, for *top-project-lines-added*, the attribute *top-project-lines-added#lines-removed* would contain the sum of the *lines-removed* file metric for all files contained within the author's *top-project* (where *top-project* has been calculated based upon *lines-added*).

3) community-[similarity-metric]-[file-metric]: This numeric value indicates the community into which an author is grouped by the *similarity-metric* when calculated using the *file-metric* (for example, *community-cos-similarity-lines-removed*).

4) community-commit-[similarity-metric]-[aggregate]-[file-metric]: This numeric value indicates the community into which an author is grouped when the *similarity-metric* is calculated using the *file-metric* as with *community-[similarity-metric]-[file-metric]*. However, in this case, the similarities are calculated in a pair-wise comparison of commits between two developers. The edge value is either the *MEAN*, *MEDIAN*, *SUM*, or *MAX* of those comparisons, indicated by *aggregate*.

B. Revision Node

A Revision Node (*node-type* = *revision*) represents a single revision made by an author.

1) revision-number: The *revision-number* attribute indicates the unique revision number from the SVN repository. Note that revision numbers are not necessarily chronological in SVN.

2) date: The *date* attribute contains the UNIX timestamp indicating the time and date the revision was committed into the repository.

C. File Node

A File Node (*node-type* = *file*) represents a single fully qualified file path within the repository. File nodes are immutable and always represent the same path, regardless of restructuring within the repository. They contain a *path* attribute.

D. AUTHOR_ON_REVISION Relationship

An *AUTHOR_ON_REVISION* relationship connects an author to a revision that the author committed. It is directed from the author to the revision.

E. ADD Relationship

An *ADD* relationship connects a revision node to a file node and indicates that the file was created in the specified revision. It contains a *lines-added* attribute.

F. DELETE Relationship

A *DELETE* relationship connects a revision node to a file node and indicates that the file was deleted in the specified revision. It contains a *lines-removed* attribute.

G. MODIFY Relationship

A *MODIFY* relationship connects a revision node to a file node and indicates that the file was modified in the specified revision. It contains both a *lines-added* and a *lines-removed* attribute.

H. REPLACE Relationship

A *REPLACE* relationship connects a revision node to a file node and indicates that the file was replaced by a new file in the specified revision. It contains both a *lines-added* and a *lines-removed* attribute.

I. AUTHOR_TOUCHES_FILE Relationship

The *AUTHOR_TOUCHES_FILE* relationship connects an author to a file that he/she has added, deleted, modified, or replaced in some revision within the data set. It contains a *count* attribute that indicates the number of revisions in which the author has affected the file in some manner.

J. AUTHOR_TO_AUTHOR Relationship

When two authors are connected to a common file via an *AUTHOR_TOUCHES_FILE* relationship, they are also connected through an *AUTHOR_TO_AUTHOR* relationship. This relationship stores information that describes the similarity between two authors as calculated by three metric types: 1) *co-commit-count*, 2) vector-based similarity metrics, and 3) vector-based similarity metrics at the commit level.

1) co-commit-count: The size of the intersection between the sets of files that two authors a and b have modified ($\|\lambda_a \bigcap \lambda_b\|$) is the *co-commit-count*.

2) [similarity-metric]-[file-metric]: The similarity between two authors as defined by $m(\sigma(\lambda_a), \sigma(\lambda_b))$ where m is a similarity metric as described in Section IV-B.

3) commit-[similarity-metric]-[aggregate]-[file-metric]: For each pairwise comparison of commits between two developers, this metrics performs the same calculation as *[similarity-metric]-[file-metric]*. It then returns a value using a function, ϕ, such as maximum, minimum, mean, or median: $\phi_{i,j}(\sigma(\lambda_{a,i}), \sigma(\lambda_{b,j}))$.

VI. Limitations

A. Monolithic Commits

In the SVN revision history for the ASF (and in other open source repositories) we frequently see large commits by a single developer on the order of many hundreds of projects and thousands of lines of code [10], [11], [13]. These monolithic commits may cause a developer to be much more highly connected to other developers in one metric, while not connected to anyone in another metric.

B. Programming Language Verbosity

Programming languages vary greatly in the number of lines of code required to produce the same results [2], [3], [4], [6], [7]. For example, consider the difference in terseness between C and Perl. Some of our metrics rely upon the number of lines of code added or removed, therefore they may be biased by differences in verbosity.

C. Co-Commit Requirement

For two authors, a and b, to be connected by an *AUTHOR_TO_AUTHOR* relationship, they must have made changes to some common set of files ($|\lambda_{a,i} \cap \lambda_{b,i}| > 0$). While this requirement makes sense in many cases, is easy to determine, and produces acceptable results, it does not capture every instance of developer interaction. For example, author a may rely heavily upon an API that is maintained by author b. Although author a never makes changes to the source files that define the API (resulting in $|\lambda_{a,i} \cap \lambda_{b,i}| = 0$), authors a and b could collaborate regularly to determine the requirements and specification for future API modifications.

References

[1] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, 2008.

[2] Daniel P. Delorey, Charles D. Knutson, and Scott Chun. Do Programming Languages Affect Productivity? A Case Study Using Data from Open Source Projects. In *1st International Workshop on Emerging Trends in FLOSS Research and Development*, May 2007.

[3] Daniel P. Delorey, Charles D. Knutson, and Christophe Giraud-Carrier. Programming Language Trends in Open Source Development: An Evaluation Using Data from All Production Phase SourceForge Projects. In *2nd International Workshop on Public Data about Software Development*, June 2007.

[4] Daniel P. Delorey, Charles D. Knutson, and Alex MacLean. Studying Production Phase SourceForge Projects: A Case Study Using cvs2mysql and SFRA+. In *Second International Workshop on Public Data about Software Development*, June 2007.

[5] Shih-Kun Huang and Kang min Liu. Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–5. ACM, 2005.

[6] Jonathan L. Krein, Alexander C. MacLean, Daniel P. Delorey, Charles D. Knutson, and Dennis L. Eggett. Language Entropy: A Metric for Characterization of Author Programming Language Distribution. *4th Workshop on Public Data about Software Development*, 2009.

[7] Jonathan L. Krein, Alexander C. MacLean, Daniel P. Delorey, Charles D. Knutson, and Dennis L. Eggett. Impact of Programming Language Fragmentation on Developer Productivity: a SourceForge Empirical Study. In *International Journal of Open Source Software and Processes*, volume 2, pages 41–61, June 2010.

[8] Alexander C. MacLean, Jonathan L. Krein, Landon J. Pratt, Charles D. Knutson, and Dennis L. Eggett. Modeling Latent Open Source Developer Communities: Evaluation of Relational Metrics. *Under Review*.

[9] Alexander C. MacLean, Landon J. Pratt, and Charles D. Knutson. Knowledge Homogeneity and Specialization in the Apache HTTP Server Project. In *Proceedings of the 7th International Conference on Open Source Systems (OSS 2011)*, pages 106–122, October 2011.

[10] Alexander C. MacLean, Landon J. Pratt, Jonathan L. Krein, and Charles D. Knutson. Threats to Validity in Analysis of Language Fragmentation on SourceForge Data. In *Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research*, May 2010.

[11] Alexander C. MacLean, Landon J. Pratt, Jonathan L. Krein, and Charles D. Knutson. Trends That Affect Temporal Analysis Using SourceForge Data. In *Proceedings of the 5th International Workshop on Public Data about Software Development*, June 2010.

[12] Andrew Meneely, Laurie Williams, Will Snipes, and Jason Osborne. Predicting Failures with Developer Networks and Social Network Analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 13–23, 2008.

[13] Landon J. Pratt, Alexander C. MacLean, and Charles D. Knutson. Cliff Walls: An Analysis of Monolithic Commits Using Latent Dirichlet Allocation. *Proceedings of the 7th International Conference on Open Source Systems (OSS 2011)*, 2011.