

Analysis of Activity in the Open Source Software Development Community

Scott Christley and Greg Madey
 Dept. of Computer Science and Engineering
 University of Notre Dame
 Notre Dame, IN 44656
 Email: {schristl, gmadey}@nd.edu

Abstract—Open Source Software is computer software for which the source code is publicly open for inspection, modification, and redistribution. While research of a few, large, successful projects have provided insights into the nature and practices of the open source software community; it still leaves open the question about the thousands of other open source projects which are neither large or highly successful. In this paper, we describe a data set of SourceForge.net, the world's largest open source software development site, which is available for research purposes; we discuss various data mining techniques that can be applied to the data and the type of research questions that can be answered. We apply a few of these techniques and provide analysis of the results.

I. INTRODUCTION

Open Source Software is computer software for which the source code is publicly open for inspection, modification, and redistribution. Individuals involved in open source software projects exhibit high levels of motivation to participate; though, the type of motivation varies among groups of individuals [1]; still the number of projects and individuals in the community continues to grow. Individuals can participate in open source software projects in a number of different ways, such as contributing source code, testing and installing the software, reporting and fixing bugs, writing documentation, and posting forum messages. Proponents of the open source software community have suggested that the community employs different software development processes from proprietary software development, and these different processes are part of the reason for the success of open source software. However some descriptions of the open source software development processes are based upon of a few, large, already successful projects like Apache, Mozilla [2] and NetBeans.org [3]. Another approach is to use data mining algorithms and analyze a large number of open source software projects to infer properties of the projects and users.

In this paper, we discuss a large data set of open source software projects from the SourceForge.net website and the Concurrent Version System (CVS) repositories. We describe how we extracted information regarding the activities performed by users on those projects; we discuss a variety of analyses that can be performed with the data and the type of research questions that can be answered by such analyses. Finally, we implement some of the data mining techniques, apply them to the data set and discuss the results.

II. DATA SET

SourceForge.net [4] is the world's largest open source software development web site, with the largest repository of Open Source code and applications available on the Internet. Owned and operated by OSTG, Inc. ("OSTG"), SourceForge.net provides free services to open source developers. The SourceForge.net web site is database driven and the supporting database includes historic and status statistics on over 100,000 projects and over 1 million registered users' activities at the project management web site. OSTG has shared certain SourceForge.net data with the University of Notre Dame for the sole purpose of supporting academic and scholarly research on the free/open source software phenomenon. There are over 100 relations (tables) in the data dump provided to Notre Dame. Some of the data have been removed for security and privacy reasons. SourceForge.net cleanses the data of personal information and strips out all OSTG specific and site functionality specific information.

As part of a previous study [5], twenty-one types of activity event data was extracted from the SourceForge.net data dump; this activity can be considered support or management operations: submission or assignment of artifacts like bugs, feature requests and patches, posting of forum messages, modification of project settings, file releases, management of projects tasks, documents and jobs. The collected data contains over two million activity events and includes all recorded activity from the beginning of the SourceForge.net website up to June 2005.

Activity related specifically to source code changes is not recorded within the SourceForge.net database but within the CVS source code repositories. After [5], the history records in the project CVS repositories were downloaded and correlated with the SourceForge.net data dump to provide eight additional activity types related specifically to source code operations. Downloading the CVS history took almost three months and generated over 120 million activity event records.

The activity event data contains an activity type identifier, a project identifier, a user identifier, and a timestamp; a detailed description of the activity event data and how it was collected from the numerous database tables and CVS repositories is described in a technical report [6]. Table I provides a summary of the activity types.

TABLE I
LIST OF ACTIVITY TYPES.

Activity Type	Activity Description
submit bug(1)	Person submits a new bug report.
assign bug(2)	Bug report is assigned to person.
submit support request(3)	Person submits a new support request.
assign support request(4)	Support request is assigned to person.
submit patch(5)	Person submits a new patch.
assign patch(6)	Patch is assigned to person.
submit feature request(7)	Person submits a new feature request.
assign feature request(8)	Feature request is assigned to person.
submit todo(9)	Person submits a new to-do item.
assign todo(10)	To-do item is assigned to person.
submit other artifact(11)	Person submits an artifact that is not one of the predefined categories of bug report, support request, patch, feature request, or to-do item.
assign other artifact(12)	Uncategorized artifact is assigned to person.
new forum message(13)	Person posts a new forum message.
followup forum message(14)	Person posts a forum message that is a followup to an existing forum message.
modify project(15)	Person makes an administrative modification to the project; the modification is uncategorized, but they are typically tasks like adding/removing members, changing permissions, updating project settings, etc.
file release(16)	Person posts a new file release; this is typically associated with releasing a new version of the software to the public.
new project task(17)	Person creates a new project task.
assigned project task(18)	A project task is assigned to person.
modify project task(19)	Person modifies an existing project task.
create document(20)	Person creates a new document.
create people job(21)	Person posts a new job; these are similar to help-wanted ads where a project is looking for somebody with particular skills.
checkout source code(22)	Person checks out source code from CVS repository.
export source code(23)	Person exports source code from CVS repository.
release source code(24)	Person releases check out of source code from CVS repository.
tag source code(25)	Person tags source code in the CVS repository with a label.
add source code file(26)	Person adds a new source code file to the CVS repository.
remove source code file(27)	Person removes a source code file from the CVS repository.
modify source code file(28)	Person commits a source code modification to the CVS repository.
update source code(29)	Person updates local checked out source code with any changes in CVS repository.

A. Related Data Sets

The data used in this study can be obtained from the authors and will be made available as part of the OSS Research Portal. Besides the OSS Research Portal [7] at Notre Dame, other large open source data collection projects include FLOSSmole and CVSanaly. FLOSSmole [8] was originally designed to gather information about SourceForge.net projects from the web pages, but it has since expanded to include data from Freshmeat [9], Rubyforge [10], and Objectweb [11]. FLOSSmole also provides data donated by other researchers. Because the data set at the OSS Research Portal is an archive of the back-end database running the SourceForge.net website; it contains some additional information which is not directly accessible from the website or is difficult to extract. Such information includes auditing data regarding when and who made changes, and linkage identifiers between separate sections of the project. CVSanaly [12], [13] gathers detailed data from the CVS repositories at SourceForge.net and archives it into an SQL database. It archives more information regarding CVS activity than we collected for this study and structures the data making for easier analysis than using the raw CVS repository data. All of these data collection projects provide a web-based query mechanism that allows researchers to obtain subsets of the data.

Despite the efforts to collect data about open source projects, there is considerably more data that could be collected. New websites that provide services similar to SourceForge.net continue to appear. Some have been around for a long time like the one for GNU software [14][15], others appeal to specific cultures like GNA! in France [16], while some attempt to centralize efforts around a common theme like bioinformatics [17] or the Ruby programming language [10]. Well established software programs like Apache [18] and Mozilla [19] host additional related projects, and there are archives like CTAN [20] for TeX and CPAN [21] for Perl. Each of these communities provide varying services for developers and users, so designing an inclusive data collection scheme is a daunting task.

III. ANALYSIS OF ACTIVITY

The accumulation of activity data of multiple types into a single data set allows for analyses that would be considerably more complicated if the data was spread out among multiple sources. The type of analyses performed and the knowledge gleaned from them varies depending upon the viewpoint taken. In this section, we describe various types of analyses that can be performed with the activity event data and possible insights those analyses can provide; while in the following sections, we perform a few of these analyses and discuss the results.

A. Characteristics of Individuals and Projects

The data provided in the SourceForge.net data dump has actually very little information regarding the individual users in the community. Registered users only need to provide their name and an email address, so demographic information like age, gender, ethnicity and geographic location are not

available. SourceForge.net does offer facilities for users to record what skills they have; things like years of experience in programming languages and operating systems, but only a fraction of a percent of users actually provide that information. For this reason, most insights about users need to be inferred, and the activity event data can provide a wealth of insights based upon the activities that users perform.

Some projects allow anonymous users to perform activities related to the project; these may be activities like creating artifact records or posting forum messages. In the collection of activity data, we maintain the activity records performed by anonymous users; therefore in the analysis of this data, one has to be aware of whether anonymous activity should be used or not. Anonymous activity has the primary issue of lacking identity; it may be that a project member or registered user performed the activity but forgot to login to SourceForge, and it is unknown if multiple activities are performed by one, a few, or many people. Anonymous users are given user id 100 in the activity records, so that activity can be easily excluded if desired.

The SourceForge.net data dump has considerably information about projects; this is not surprising as the purpose of SourceForge.net is to provide services for people to create and to manage software projects. Projects are placed within a categorical hierarchy that broadly distinguishes the purpose of the software; for example, is the software a game, a scientific application, a developer library, etc; these have historically been called trove categories. Projects also have attributes like the programming languages used in the software, what operating systems the software supports, the open source license for the software, and various statistics compiled by SourceForge.net; these attributes are all available in the data dump.

B. Aggregation

Different aggregations of the activity data can answer slightly different questions. If the activity of a user is aggregated across all of the projects that user contributes to, then analysis of the activity provides a viewpoint of the user within the entire SourceForge.net community. The projects are essentially factored out leaving a global view of the users' activity. A similar aggregation can be performed for the projects whereby the activity for all users on a project are aggregated; this factors out the users and provides a global view of the projects' activity. However, aggregation at the user/project level allows analysis such as comparing whether the activity a user performs on one project is different from the activity that same user performs on another project; this can answer questions about whether users have different relationships, like social positions discussed below, with different projects. Likewise from the project perspective, the user/project aggregate level can answer questions about what type of relationships and the number of people with those relationships that exist for projects; maybe there are certain types of relationships in certain quantities that are necessary for a project to be successful?

C. Social Network Analysis

The activity data can be used to construct a multi-relational, weighted, bipartite (two-mode or affiliation) social network; the nodes are individuals and projects, and the edges link individuals to projects. Each edge between an individual and a project represents a relationship based upon an activity that individual performs for that project, and the weight on the edge corresponds to the quantity of that activity performed by the individual. Multiple edges can exist between an individual and a project where each edge represents a different activity type. Topological analysis can be performed on the resulting social network to learn about global properties of the community [22].

Social network analysis can be applied to infer properties about the individuals; one example property is the individual's social position. Positional analysis [23] seeks to group individuals into disjoint subsets according to their social position in the network. A social position can be considered a pattern of embeddedness in the social network, so individuals who are similarly embedded occupy similar social positions. The social network analysis concept behind social positions is that people in the same position are indistinguishable from each other, and those people can be interchanged without altering the structure of the social network. Positional analysis is another technique that can be used to distinguish core and periphery users [24]. Determining whether two individuals occupy the same social position requires some equivalence measure; for social networks, structural equivalence is the defining measure; that is, do both individuals hold the exact same relations to other people in the network. Generally, individuals are rarely exactly structurally equivalent, so approximate measures need to be used in order to obtain worthwhile results; clustering techniques from the data mining literature [25], [26] can be used to extract social positions.

Another social network analysis technique is role analysis. Once the social positions in the network have been discovered, then role analysis refers to the relationship between two social positions. For example, social positions like developer and tester have the relationship that the developer produces source code modifications, communicates those changes to the tester, and the tester tries out those changes and provides feedback to the developer. Analysis of roles can answer questions about whether open source projects have the same relationships between social positions as proprietary software development.

D. Temporal Analysis

Because the activity data has timestamps, we can consider a temporal analysis of the social network. This can answer questions about whether an individual stays in the same social position or changes social position. Does an individual desire to maintain the same social positions and thus moves from project to project over time? Do the number of people occupying social positions and the type of social positions on a project change over time to reflect the growing maturity of the software project? With new users entering the community, are they filling the same social positions as previous new user

have in the past? Do the roles or relationships between social positions change over time?

Performing temporal analysis requires that we consider more closely our conceptualization of time in relation to the social network as it determines how we chunk the data for analysis. One perspective is global (absolute, wall-clock) time which looks at the social network as a whole from time period to time period, so one talks about the state of the network in 2003 compared to the state of the network in 2004 or what changes occurred in February versus what occurred in January. Nodes and edges are added or removed or their attributes changed; global network measures are used to describe the changes, and contributions at the individual level tend to get washed out in the aggregate. The other perspective is local (relative) time that takes an egocentric viewpoint by looking at each person in the social network; here each person's first appearance in the social network becomes time 0, and structural and attribute changes are recorded from time period to time period. With the local perspective, the data does not generally correspond to any actual social network because individuals join the network at different global times; instead, its focus on the individual person allows egocentric measures to be analyzed in a time relative way to the overall social network.

E. Software Development Processes

Presumably certain software development processes are being exercised by contributors to a software project, and the activity event data can be considered a trace of those processes. An important question is whether those software development processes can be discovered through analysis of the activity data. Process discovery or process mining involves extracting information about the underlying process models based upon a time series of event data. Existing research defines the problem in different ways depending upon the type of process models extracted, the information provided in the data, and assumptions about the processes producing the events. Types of process models include finite state machines [27], Petri nets [28], and directed graphs [29], [30]. The assumptions about the processes plays an important role in how the algorithms attempt to extract the process models. Some research [28], [29], [27] assumes that there is a single process model, and the event data are traces of multiple executions of that single process. Other research [30] allows for multiple process models and assumes that frequently occurring similar temporal patterns are produced from the same process. Aalst and Weijters [28] provide a survey of the existing literature and pose a number of open research issues.

F. Limitations

One might think that with a large data set just about any analysis can be extracted but this is not necessarily true. There are limitations with analyzing large data set; some which are specific to the actual data source, like SourceForge.net in our case, and some which are for large data sets in general.

Large data sets often have a significant amount of noise within them; this noise can manifest in different ways. Some noise resides within the data values themselves, so called data errors, which are introduced by the user entering the data. These data errors may be incorrect names, identifiers, dates, codes, etc., and can be difficult to catch as the original context for the data may be lost and the correct values might not be inferable from other data. This noise can lead to incorrect analysis if decisions are being made upon those data values; though the general assumption is that such data errors are rare and strong correlations in the data will show up regardless.

Another way noise can appear is in the results of the analysis; for example, it is well known that many open source projects both on SourceForge.net and elsewhere are "dead" projects. These are projects that have been abandoned by their developers. The sheer quantity of these "dead" projects can drown out any signal from the healthy, active projects. It then becomes important to selectively focus on subsets of the data; though selection can be tricky to quantify exactly because there are also many projects which look dead but in fact are stable, production-quality software projects with small user communities, and these projects may be important to include in analysis.

Large data sets often are collected automatically so they can suffer from automation bias; that is a systematic bias whereby data which is easy to collect is collected but difficult to collect data is ignored. Auxiliary to automation is that the data being collected is for a purpose different from research purposes, so research is being performed on the data because it is readily available; this is an issue with SourceForge.net data as described in more detail below. Lastly, some traditional analysis techniques do not scale up to large data sets. While this provides opportunity for new algorithms to be developed, in some cases those techniques are just not tractable, for example finding cliques in a social network, so new or approximate techniques must be devised. Researchers are required to substantiate their analysis techniques besides their results as the significant body of research in the traditional analysis techniques cannot be drawn upon.

The SourceForge.net data dump suffers from a number of limitations beyond the generic limitations described above. Most notable is the lack of information about users. User demographics such as gender, race, age, geographic location, marital status, education, employment, salary, economic status, work experience, skills, interests, etc. are completely non-existent. This makes it essentially unrealistic to make any general observations about the people involved in the open source software community without obtaining additional data from other sources. The primary reason for this limitation is that SourceForge.net was not designed with the goal of capturing research data; its purpose is to allow for the easy management of software projects. The public nature of SourceForge.net also tends to discourage systematic collection of user information; thus providing users greater control over their anonymity and access to private information. Contrast this with a company that is required to maintain certain forms of user information

for governmental reporting and tax purposes.

The open nature of SourceForge.net’s provision of services also means that software projects are free to use or to ignore each service on an individual project basis. For example, some projects do not use the CVS source repository and only provide file releases. Some projects prefer to use mailing lists for communication instead of message forums or vice-versa while some projects use both. Some projects deal with bug reports and feature requests through email while other projects utilize the artifact tracking services. The issue is that proper analysis needs to consider many of the possible combinations that are important to the analysis; excluding a specific data source may introduce a bias in the results and assuming that SourceForge.net contains all of the appropriate data may be incorrect. One particular example is communication among users. The SourceForge.net data dump only contains messages posted on the forums; email messages sent to mailing lists are handled by a separate program where the archives are not readily available, and direct user-to-user email is essentially unrecordable.

IV. METHODS

A. Discovery of Social Positions

In data mining, the general problem of grouping elements into disjoint subsets is called clustering, and popular algorithms include K-means, K-mediod, and Expectation-Maximization(EM). K-means clustering can be used when approximate structural equivalence is desired with Euclidean distance or correlation; however, it requires the number of clusters, K, to be specified a priori. K-mediod is identical to K-means except it uses the median value to describe the cluster instead of the mean; while, EM is an iterative application of K-means to find the appropriate number of clusters.

An underlying assumption for many of these techniques is the existent of an appropriate equivalence metric, and some go further in assuming the underlying distribution for clusters is a Gaussian (normal) distribution. These are tenuous assumptions as it is unclear whether those distributions hold for structural patterns found in typical social networks. For clustering, we devise a novel algorithm that clusters individuals into social positions. By utilizing a non-parametric statistical test, the algorithm makes no assumptions on the underlying distribution of the social network data; likewise, it actually determines structural non equivalence of people thus avoiding issues with similarity metrics.

Unlike a distance metric that determines how close sample data is in an N-dimensional space, a statistical test compares sample data to determine if the samples come from the same underlying distribution. For positional analysis, we assume that each social position has a different underlying distribution that defines some sample data; however, this distribution is not known a priori. Therefore, the task of determining the social positions is best done by making as few assumptions as possible on the underlying distributions for each social position while still being able to determine if a sample data set does or does not come from the same distribution as

another sample data set. The non-parametric test we use for analyzing social positions in Open Source Software is Fishers contingency-table test for variables with more than two categories [31]; however, our algorithm is independent of the actual statistical test so other more appropriate non-parametric tests may be used for different data sets.

A significant result from the statistical test is interpreted as the null hypothesis is rejected and the alternative hypothesis is accepted; while a non-significant result indicates we fail to reject the null hypothesis. The null hypothesis states that the activity distribution for two individuals is the same distribution; while the alternative hypothesis states that the distributions are different. This leads very naturally into an algorithm for clustering that is based upon non equivalence of data instead of equivalence; in particular, if the null hypothesis is rejected then two samples cannot be from the same underlying distribution (with some level of confidence) therefore those samples cannot be the same social position so they are put into separate clusters. By doing a pair-wise statistical test between each sample, we separate all of the samples which are different, and we are left with a set of samples where we could not reject the null hypothesis and that set then becomes one cluster. We continue this pair-wise comparison process with the remaining unclustered samples, i.e. the samples that got rejected or excluded from a cluster, until all of the samples have been placed into a cluster. Algorithm 1 shows how a set of samples are clustered.

Algorithm 1 cluster(S: set of samples)

```

UC = S; i = 1;
while UC ≠ ∅ do
    Ci = UC; UC = ∅;
    while some samples not yet pairwise compared do
        A = Pick unmarked sample from cluster, Ci
        for each other sample, B, in cluster Ci do
            Run statistical test on A and B
            if significant result then
                Ci = Ci/B;
                UC = UC ∪ B;
            end if
        end for
        Mark A as being pairwise compared
    end while
    i = i + 1;
end while
return C1...i

```

Our algorithm is not without issues however; it suffers from one of the key problems that is also problematic for other clustering mining algorithms. Specifically because this statistical test does not satisfy the mathematical property of transitivity, it is not an equivalence relation, so the clusters it produces are not unique. The clusters produced are highly dependent upon the order in which the samples are compared; likewise, finding the optimal set of clusters is at least NP-hard so no easy algorithmic solution is available. To alleviate

this problem, we perform an EM-like clustering where we iteratively re-evaluate the clusters until they settle down to a stable state; that is, the clusters do not change from iteration to iteration.

B. Temporal Social Positions

We consider a simple extension to our clustering algorithm in order to provide temporal analysis of social positions. The essential idea is to chunk the data into time intervals and run the clustering algorithm on each data chunk; thus, a time series of social positions is obtained which represents the evolution of social positions in the social network. In a stable network, one where the process that governs the underlying mechanism for the formation of the social network stays constant, we would expect the social positions to change little over time; though the individuals who hold a specific social position may change as well as the number of people. However, in a self-organizing or adaptive network, one where the underlying process is changing or possibly has a lifecycle associated with it, we would expect the social positions to change; furthermore, the changing social positions should correlate in some manner to the underlying process governing the social network.

As mentioned above, either a global or local time perspective can be taken when considering how to aggregate data for temporal analysis. We argue that the local time perspective is more appropriate for the temporal analysis of social positions as it is an egocentric measure, so that is what we use for analysis of the SourceForge.net activity data.

C. Discovery of Temporal Activity Patterns

Lack of detailed understanding about the actual software development processes exercised by open source software projects implies that we should have relaxed assumptions with weak restrictions about those processes. Specifically we assume that projects utilize different processes and that those processes may evolve or change over the lifetime of the project. Also, we assume that multiple processes are occurring simultaneously for each project, that multiple users are involved in a single process, and that users can be involved in multiple processes simultaneously. Based upon these assumptions, we shall search for frequent temporal patterns specifically taking a graph-based representation of the processes and searching for frequent graph structures.

Our intent is to find frequent temporal patterns either within an activity sequence or across multiple activity sequences. For the open source activity data, we make the assumption for our algorithm that a software process occurs within just a single project and does not cross projects, so we naturally partition the activity sequence into a set of sequences for each project.

Finding frequent temporal patterns is a matter of counting up the number of equal temporal patterns across the set of activity sequences of interest. If we are counting up temporal patterns within a single activity sequence then we need to be careful not to double count events for multiple temporal patterns. If we are counting up temporal patterns across a

partition of an activity sequence then by definition the events are disjoint. For the open source activity data, only the activity type is used for determining if two events are equal.

We devised an algorithm that functions along the same lines as the Apriori algorithm [32] whereby an initial set of candidates of size one is created; then multiple passes over the data is performed, counting the support for each candidate, pruning candidates which do not meet a minimum support, and generating a new set of candidates of size one larger. This is repeated until no candidate sets meet the minimum support.

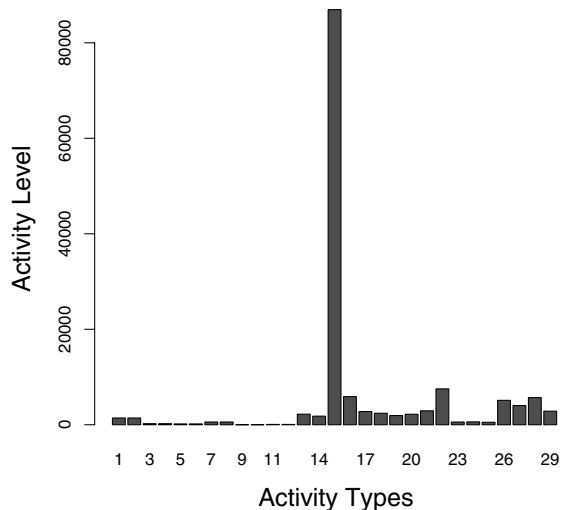
Algorithm 2 countCandidates(S : activity sequence, C : set of candidates)

```

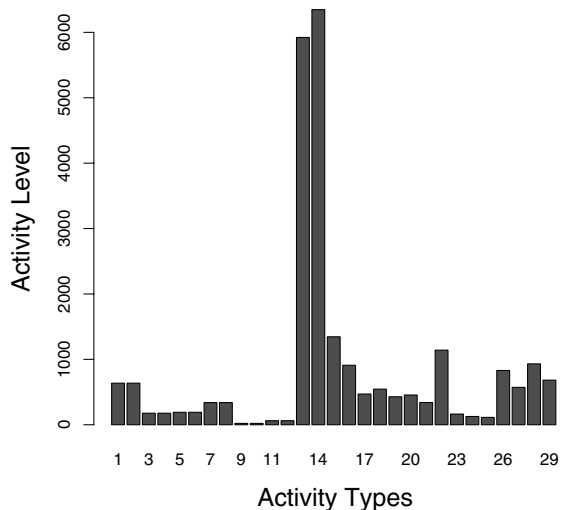
int counts[length of C] = 0;
for each candidate  $c \in C$  do
    boolean flag[length of S] = false;
    repeat
        found = false;
        k = 0; // event index for candidate
        // Scan through activity sequence
        for j = 0; j < length of S; ++j do
            if flag[j] then
                continue; // already matched
            end if
            // Does candidate event match sequence event
            if c[k] == S[j] then
                ++k;
                flag[j] = true;
            end if
            // Found match for all events in candidate
            if k == length of candidate c then
                found = true;
            end if
        end for
        if found then
            ++counts[c]; // increment count
        end if
    until found == false;
end for
return counts;

```

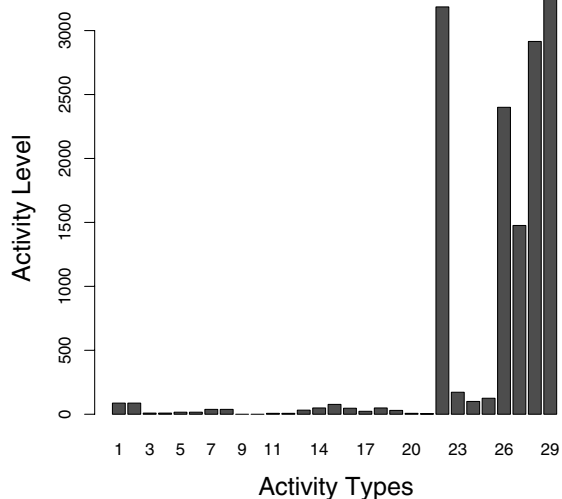
Counting the number of appearances of a candidate, see Algorithm 2, within the activity sequence involves multiple scans over the activity sequence to match activity events in the same temporal sequence as the activity events in the candidate. When a match is found, the matching activity events in the sequence are flagged so that subsequent scans do not match those events again. Pruning the set of candidates is simply performed by removing candidates whose count does not meet a minimum support threshold. Lastly, the pruned set of candidates is used to generate a new set of candidates of size one larger. The generation of the new candidate set follows the Apriori principle that a candidate sequence can only be frequent if all of its subsequences are also frequent.



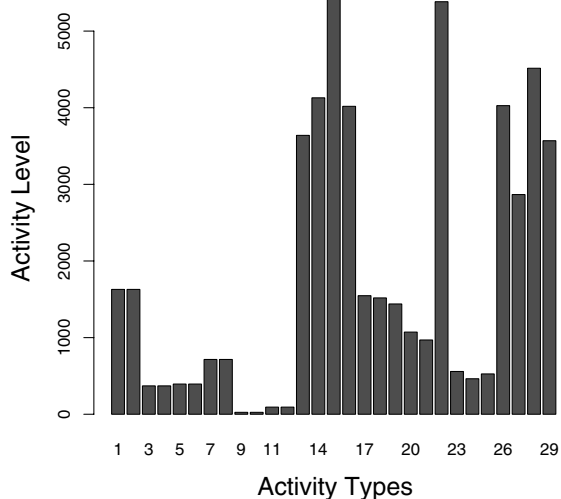
(a) Project Administrator



(b) Message Poster



(c) Software Developer



(d) Handyperson

Fig. 1. Typical activity distribution for a few social positions. The activity types are numerically coded as in Table I, and the activity level is the total number of activity events for each activity type in that social position’s cluster. The Project Administrator (a) has the primary activity of project modification, the Message Poster (b) has mainly posting of new and followup forum messages with a few other activities, the Software Developer (c) has mainly the source code operations as its activities; while the Handyperson (d) has significant activity for many types.

V. RESULTS

A. Social Positions

For our clustering algorithm, each user/project pair and the corresponding activities that user performs for that project becomes a single sample, so the statistical test compares a user/project pair against another user/project; each random variable is a single activity event performed by the user and recorded into one of the twenty-nine activity categories. Our data set has a total of 271307 unique user/project pairs with

activity from the beginning of SourceForge.net to June 2005 which contains over 120 million activity events. The initial result of running our algorithm on this data produced 15941 clusters; however after performing the iterative clustering, only 45 clusters remained with greater than 100 people in a cluster, and the remaining clusters are outliers with only a few people in each cluster. Figure 1 shows the activity distributions for some of the social positions.

The largest cluster contains 111889 user/project pairs while

the second largest cluster contains 93199 user/project pairs, so together the two largest clusters account for 75% of the SourceForge.net community. Of the 45 clusters, manual analysis of the activity distributions indicate about seven different social positions can categorize the activity. Six of the social positions was each a separate cluster while the seventh social position, the software developer, included the remaining 39 clusters. The clusters for the software developer social position all clearly have source code operations as their primary activity, but they are in separate clusters because they have different proportions for the different operations, and some clusters have a small amount of project modification and file release activities. Table II describes the seven social positions extracted.

B. Temporal Social Positions

We chose a time period of one month for aggregating the data; and as of this writing, we have run our algorithm for over 20 time periods. Table III shows the results for the first four time periods which is all that we currently analyzed. Month 0 is the activity performed from local time 0 (time of first activity event for each user/project pair) until one month has passed, Month 1 is the activity performed after the first month until the second month has passed, and so on. The total number of user/project pairs for month 0 does not match the total in Table II because some activity records either had no date or had invalid dates, so that data was discarded.

We ran our clustering algorithm on the data for each time period separately then grouped the largest clusters together into social positions. Notice that for the first period we get results similar to our analysis for activity over all time. By the second period, however, there is a significant shift; the Project Administrator social position is gone completely, and there is a significant drop in the total user/project pairs which are performing activity. Likewise there is a significant drop in the Message Poster social position, presumably people who initially only post forum messages have either left the project or have started performing other activities which has shifted their social position. By the third period, the Handyperson social position has clearly taken a prominent role in the community, and surprising the only social positions left by the fourth period are the Software Developer and Handyperson.

C. Temporal Activity Patterns

The temporal pattern algorithm was able to extract interesting processes from the temporal activity sequences for a set of open source projects. Some of the processes discovered look like typical software development processes involving communication, submission of bug or features reports, and access to source code. Figures 2, 3, and 4 show some example processes based upon activity sequence patterns extracted by the temporal pattern algorithm.

VI. DISCUSSION

We have shown that utilizing data mining algorithms and techniques can provide insights across a large set of open

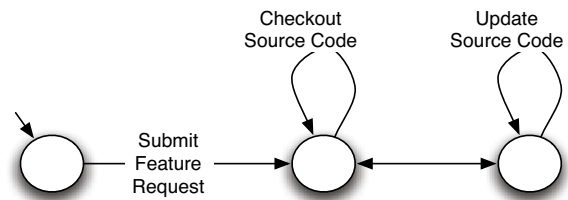


Fig. 2. Typical process which shows an initial submission of a feature request followed by a series of checkouts and updates of the source code.

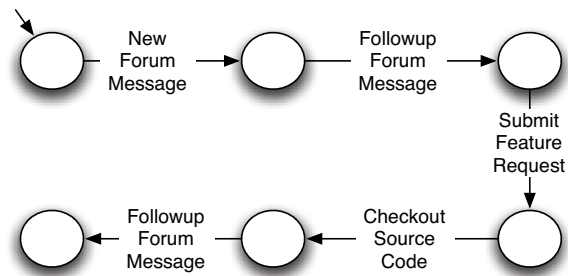


Fig. 3. Possible process for a feature request discussion, submission, and resolution.

source software projects. The extraction of social positions would seem to indicate that some but not all of the typical specialized positions in a software development organization are present in many open source software projects; furthermore, temporal analysis of social positions seems to imply that most specialized positions disappear leaving only software developers and people who do a little bit of everything. Discovery of temporal activity patterns shows some promising initial results with some patterns looking like typical software development processes.

While we have managed to extract twenty-nine activity types from the SourceForge.net data, there is additional detailed activity that can be extracted. For example, the only activity recorded for artifacts is the submission and assignment

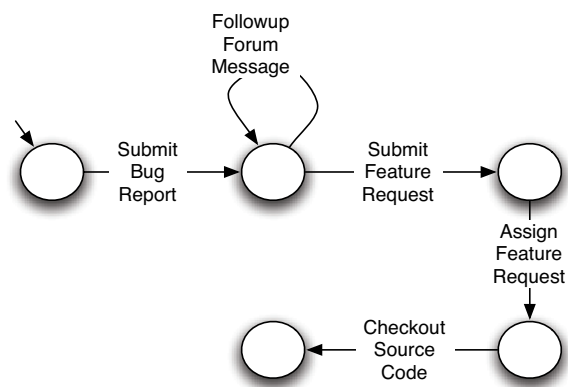


Fig. 4. Possible process for a bug report being turned into a feature request.

of the artifact; researchers interested in how open source software projects manage bug reports and feature requests can dig deeper into the data to extract other changes like closure of the artifact, change of status, followup comments, etc. Similar detail can be pursued for project modifications which includes adding and removing developers, permissions and security settings, and enabling or disabling SourceForge.net provided services.

In this article, we have just scratched the surface of what can be done by analyzing data from a large number of open source software projects. Research in this area holds great potential both for those interested data mining techniques as well as open source researchers. Specialized studies which look at specific subsets of the open source software community along with combinational studies that incorporate surveys and questionnaires with large data sets should provide additional opportunities for new insights.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829. Dr. Nitesh Chawla for fruitful data mining discussions and SourceForge.net for providing data. Thanks to the reviewers for their comments.

REFERENCES

- [1] A. Hars and S. Ou, "Working for free? – motivations of participating in open source projects," in *Proceedings of the 34th Annual Hawaii International Conference on Systems Sciences*, 2001.
- [2] 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*, vol. 11, no. 3, pp. 309–346, 2002.
- [3] C. Jensen and W. Scacchi, "Collaboration, leadership, control, and conflict negotiation and the *netbeans.org* open source software development community," in *Proceedings of the 38th Annual Hawaii International Conference on Systems Sciences*, 2005.
- [4] Sourceforge, "http://www.sourceforge.net," 2005.
- [5] S. Christley and G. Madey, "An algorithm for temporal analysis of social positions," in *North American Association for Computational Social and Organizational Science (NAACSOS 2005)*, Notre Dame, IN, 2005.
- [6] —, "Collection of activity data for sourceforge projects," University of Notre Dame, Tech. Rep. TR-2005-15, 2005.
- [7] OSS Research Portal, "http://zerlot.cse.nd.edu."
- [8] FLOSSmole, "http://ossmole.sourceforge.net/."
- [9] Freshmeat, "http://freshmeat.net/."
- [10] Rubyforge, "http://rubyforge.org/."
- [11] Objectweb, "http://www.objectweb.org/."
- [12] CVSAAnalY, "http://libresoft.urjc.es/tools/CVSAAnalY."
- [13] G. Robles, S. Koch, and J. M. González-Barahona, "Remote analysis and measurement of libre software systems by means of the CVSAAnalY tool," in *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04). 26th International Conference on Software Engineering*, 2004.
- [14] GNU Software, "http://savannah.gnu.org/."
- [15] non-GNU Software, "http://savannah.nongnu.org/."
- [16] GNA!, "https://gna.org/."
- [17] Bioinformatics, "http://bioinformatics.org/."
- [18] Apache, "http://www.apache.org/."
- [19] Mozilla, "http://www.mozilla.org/."
- [20] The Comprehensive TeX Archive Network, "http://www.ctan.org/."
- [21] The Comprehensive Perl Archive Network, "http://cpan.perl.org/."
- [22] J. Xu, S. Christley, Y. Gao, and G. Madey, "A topological analysis of the open source software development community," in *The 38th Annual Hawaii International Conference on System Sciences*, 2005.
- [23] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge, UK: Cambridge University Press, 1994.
- [24] K. Crowston, K. Wei, Q. Li, and J. Howison, "Core and periphery in free/libre and open source software team communications," in *Proceedings of the 39th Annual Hawaii International Conference on Systems Sciences*, 2006.
- [25] D. Hand, H. Mannila, and P. Smyth, *Principles of Data Mining*. MIT Press, 2001.
- [26] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann, 2005.
- [27] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," *ACM Transactions on Software Engineering and Methodology*, vol. 7, no. 3, pp. 215–249, 1998.
- [28] W. Aalst and A. J. M. M. Weijters, "Process mining: a research agenda," *Computers in Industry*, vol. 53, pp. 231–244, 2004.
- [29] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining process models from workflow logs," in *Lecture Notes in Computer Science*. Springer-Verlag, 1998, vol. 1377, p. 469.
- [30] S.-Y. Hwang, C.-P. Wei, and W.-S. Yang, "Discovery of temporal patterns from process instances," *Computers in Industry*, vol. 53, pp. 345–364, 2004.
- [31] J. Krauth, *Distribution-Free Statistics: An Application-Oriented Approach*. Amsterdam, The Netherlands: Elsevier Science Publishers, 1988.
- [32] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, 1994.

TABLE II
SOCIAL POSITIONS IN THE SOURCEFORGE.NET COMMUNITY

Social Position	Description	Size
Software User	The largest cluster with the primary activities of posting new forum messages, followup forum messages, and checkout out source code.	111889
Project Administrator	The second largest cluster with the primary activity of making project modifications; the project administrator also performs file releases, but most other activities are relatively minor or non-existent.	93199
Software Developer	Primary activities are source code operations like checking out source code, add/remove source code files, modify source code, and update source code. The social position contains 39 clusters all with different relative proportions of the source code operations, and some software developers have significant levels of project modification and file release activities.	47495
Task Management	Significant usage of the project task management provided by SourceForge.net.	2181
Bug Reporter	Significant bug reporting activity with a slight amount of features requests, support requests, and patches.	1138
Feature Requester	Primary activity was submission of feature requests but also has a significant amount of bug reporting.	370
Handyperson	The handyperson has significant activity for many different activity types including source code modifications, bug reporting, project modifications, file releases, and project tasks.	217
Not Categorized	The remaining very small clusters that were not analyzed.	14818
	Total user/project pairs	271307

TABLE III
TEMPORAL SOCIAL POSITIONS IN THE SOURCEFORGE.NET COMMUNITY

Social Position	Description	Month 0	Month 1	Month 2	Month 3
Project Administrator	Primary activity of making project modifications; other activities are relatively minor or non-existent.	86951	0	0	0
Message Poster	Primary activity of posting new or followup forum messages; other activities are relatively minor or non-existent.	96052	7315	0	0
Software Developer	Primary activities are source code operations like checking out source code, add/remove source code files, modify source code, and update source code. Multiple clusters with different relative proportions for the source code operations; some software developers have significant levels of project modification and file release activities.	67488	32126	21054	18239
Release Management	Primary activity of file release but also significant project modification activity.	11700	7227	0	0
Task Management	Significant usage of the project task management provided by SourceForge.net.	1775	0	0	0
Handyperson	The handyperson has significant activity for many different activity types including source code modifications, bug reporting, project modifications, file releases, and project tasks.	1768	120	14050	10709
Not Categorized	The remaining very small clusters that were not analyzed.	3066	1638	1712	1611
	Total user/project pairs	268800	48426	36816	30559