CS680: Distributed Software Development

Project Report - Spring Term 2010

Team Members:

1. First Name: Jordan Last Name: Osecki E-mail: Jordan.Osecki@gmail.com
2. First Name: Sammie Last Name: Stahlback E-mail: Sws28@drexel.edu
3. First Name: Michael Last Name: Kim E-mail: Mikim303@yahoo.com

Project Name:

- Analysis Team – Core/Periphery Analysis of MySQL from Data Mining of an Open Source Project

Abstract

The communication, bug tracking, and source code repositories of a project can shed light on to the communication and other functions which happen to a development team throughout the course of a project. These repositories are blueprints which enshrine all characteristics of the project, the development team, and all other aspects that typically go into software production.

Studies on the communication repositories in particular can highlight the actual communication patterns within the software development team itself. For example, is the work being done by just one person, by a small group, or by everyone equally? Are their individuals on the outside which all go to for advice? All of these types of patterns and more can be discovered through the analysis of a project's communications repository.

This paper mines data from the mailing list repositories of an open source project, MySQL, in an attempt to learn the "blueprint" of the software development team throughout different points in time (namely, their software's release dates). The mailing list data is put into an organized database and then re-organized and sent through third-party software that specializes in software project community structure analysis, in an attempt to see the results of this product.

Open source projects, such as MySQL, typically have the blueprint of a core/periphery structure, meaning that there is a small core which does most of the work and a large periphery which does very little (but meaningful) work. This paper studies the mailing lists to determine if MySQL follows this core/periphery structure and to see how it changes over time.

This study has serious implications not just for open source software, but also for any kind of distributed software development. Open source and distributed software development (DSD) has a lot in common, with open source being a form of DSD which is pretty successful. Knowing what makes open source projects successful, one of those things being the core/periphery structure of users, would be a valuable thing that could be applied to DSD projects of all shapes and sizes in order to improve the overall quality of the software that is produced.

Overview

As mentioned in the abstract, there are many distributed software development (DSD) issues and implications that are embraced by this study of the core/periphery structure of MySQL and other open source projects. Some of these issues include communications and communication blueprints, repositories and tools used to communicate remotely, mining repositories, the study of open source versus DSD, and the core/periphery structure embodied by most open source projects which seems to be so successful.

Communications and their blueprints are important. One of the biggest challenges of DSD is to be able to effectively communicate with your fellow co-workers while working in remote locations. This causes issues because software projects are full of dependencies which need to be addressed, and this is much harder to do when the two or more people responsible are not geographically located together.

Because of the communication gaps which are evident and can be seen when a blueprint of the organization structure is taken, repositories and tools that are used to communicate remotely become of the utmost importance. With face-to-face communication and formal meetings lacking in DSD, most communication occurs over computers in recorded media such as e-mail, message boards, mailing lists, bug tracking, source control, etc. These repositories become the main way to interact with people with whom you have to discuss coding dependencies with on a daily basis.

Since all of these tools and repositories become so important, mining their contents can be very valuable to observe trends in projects in general, in projects at the company, and to steer that specific project if there are problems with it. One such trend which can be observed is what the communication structure of the DSD development team is like and if it is hindering things. There is no perfect formula to create a perfectly efficient DSD team, however, so this task is hard.

Therefore, the comparison is made between open source and DSD. This is because open source has many things in common with DSD, such as workers working remotely with no real communication other than the same used by DSD workers. In addition, they use the same types of repositories and are still able to perform great work with high quality software resulting.

Studies have shown that open source software typically produces high quality software with its developers organized in a core/periphery structure, described in the "Abstract" section. This structure allows for gatekeepers to the code who write most of the code and approve the rest of it, particularly to their expertise sections. In addition, the entire periphery acts as bug reporters to aid in making the software the best it can be.

The basis of this report was to explore whether MySQL, a very successful open source project, has a core/periphery structure like it is expected to have. If it does, this can be studied by software producers of all kinds, public and private, open source and commercial, to model it for DSD situations. In addition, the communication blueprint is going to be taken over the course of many releases, which will allow researchers to see the changes to the blueprint over time, correlated to how the software was performing at that time. Finally, this report will explain how the mailing list repositories of MySQL were parsed and then re-stored, which will shed light on the best practices of mining repositories and also the movement for common formats of repositories.

The main objectives that the project was supposed to achieve include the following: re-creating the entire MySQL community and social network for each major release in the last few years; compiling and studying metrics on the core/periphery structure of this community over the same time frame; and a production of charts showing trends for these two features over the same set of releases.

To pursue these objectives, the researchers will first mine the mailing list of the MySQL project, put it into a common format in a database, and then use this database to construct whatever interface formats are necessary to feed third-party tools which will be used to analyze and re-create the MySQL social network and also obtain core/periphery metrics at each release date the team is interested in. More information on the pursuit of these objectives can be found in the "Approach" section of this paper.

Background

The project we worked with is MySQL, an open source database application. However, the team was not concerned with MySQL other than analyzing the communications that were produced during the development of the product, so that the team could use this to try to reproduce MySQL's structure. The project's main communication took place in a mailing list. The website has many categories of their mailing list, including the following: Servers, Connectors, Internals, Eventum, Community, Non-English, and User Groups. The developer's main communications, however, took place in the Internals category in the "Internals" and "Bugs" mailing lists. The mailing list is in a forum/thread structure, as opposed to an e-mail one-on-one structure.

The technique used within this process included mining these mailing lists in Perl, writing these lists to a database, pulling them back out using Java, and writing formatted files for analysis tools. Perl was chosen to parse the MySQL mailing lists because it is a great program for parsing and allowed for easy manipulation of the fields. In a dataset such as a mailing list, there are many things that change over time, such as the format of names, etc. Perl was used to account for all of these subtle changes and still allow for successful database entries. The information was written to a PostgreSQL database simply to have it in a more accessible format than the mailing lists. A separate position paper analysis by this team discusses getting repositories into "common" formats, and this is an example of taking something very unique like a mailing list and putting into a database with a set schema to make it something researchers can use more easily. Finally, Java was used to pull the information back out of the database. This was chosen because with a set schema in the database, an object-oriented language allows to easily convert the schema to objects. Finally, the data was rewritten in formats that the analysis tools would accept.

The tools that were considered and used to analyze the communication data were NodeXL and UCINET. NodeXL is a template for Excel 207 that can let the user display and analyze a network graph from a simple network edge list. It also allows for customization of the graph and calculation of statistics on the network. UCINET is a social network analysis tool as well, which was developed by Steve Borgatti, Martin Everett, and Lin Freeman. It works with a separate program, called NETDRAW, for help with visualizing networks. It also produces statistics. In the decision of which of these two programs to work with, ultimately UCINET was chosen. While NodeXL had many more options for customization of the social network picture, it only had polar layouts and centrality statistics. Meanwhile, UCINET has actual core/periphery statistics which can be easily computed, based on the analysis from Borgatti's paper.

The work of Borgatti and Everett is very important to this paper, especially in the "Models of Core/Periphery Structures" paper they wrote together. This paper gives a formal notion of a core/periphery structure and also gives algorithms for detecting this structures and statistical tests for testing whether or not a network exhibits core/periphery tendencies. The result of these studies is the actual statistics within the UCINET program, which is also made by these two researchers along with Lin Freeman. These statistics from this tool are vital in calculating core/periphery statistics for the mailing list communications of the MySQL project.

Approach

The following section describes in detail the work that the team of researchers has done and the methods that were employed in order to achieve the intended goal of core/periphery analysis for MySQL. Below are the steps that were followed, to be described in more detail immediately after. The timeline for this project was approximately six weeks, split with three weeks dedicated to data collection and three weeks dedicated to data analysis.

1. Analyze MySQL Mailing Lists – Determine Correct Ones and Best Methods to Mine Them
2. Write Script to Collect Data from MySQL Lists – Use News Groups Archives
3. Create PostgreSQL Database Using Common Mailing List Schema
4. Populate the Database with the MySQL Mailing List Data
5. Choose a Tool to Analyze the Data
6. Choose the Date Ranges / Releases to Analyze the Data
7. Create a Program to Put the PostgreSQL Information in the Correct Format for the Tool
8. Analyze the Communication Data in the Tool and Produce Final Results

The first step was to analyze the MySQL Mailing lists, which are located at http://lists.mysql.com. The ones that were to be used needed to be determined and also the best methods to mine them needed to be determined. The two that ended up being chosen were "internals" and "bugs" because the "internals" list was described as "A list for people who work on the MySQL code." In addition, the "bugs" list seemed important for a core periphery structure because many of the periphery's main contributions are through bug report and repair. Once the correct lists were chosen, the next step was to determine how to mine them. It was determined that they also existed on a News Group, and so this is how they were obtained and mined.

The second step involved writing a script to collect the data from these lists. As has been mentioned, a Perl script was written which contacts the appropriate News Groups and then parses them and organizes the information, eventually writing it to the database.

The third step involves creating a PostgreSQL database using a common schema. The database was created with the detailed schema listed on the graphic on the very next page. The main table of the schema contains information on each piece of mail, including the "to" and "from" data. These are linked via foreign key to an "alias", which can basically be one of the names for a person. There is a "person" table which is used to keep track of the one or more aliases. Finally, there is a thread key to help to group the messages into their forum or message board like threads. Otherwise, they are organized as e-mails, which isn't true.

The fourth step is to simply populate this database that was created in the previous step, using the script that was created in Step 2.

The fifth step is to choose a tool to analyze the data. As was mentioned previously in the Background section and will be mentioned more at the end of this section as an

issue the team encountered, the team originally chose NodeXL but eventually chose UCINET.

The sixth step is to choose the date ranges or releases in which to measure the core/periphery structure of the data. The snapshots were chosen by creating a list of some of the more recent or interesting MySQL releases, which includes 3-5 different versions from MySQL Version 3.*, 4.*, 5.*, and 6.*.

The seventh step was to create a program which would read from the database and put the data into the correct format for the tool. The previously mentioned Java program was created to do this. It reads from the database, populated objects similar to the database schema, and then creates output for UCINET. A different output file is created for each date/release of interest, and the format is one of simply specifying the edges.

The eighth and final step was to analyze this data in the analysis tool. The output from Step 7 was used to feed UCINET, which would then analyze the data and produce core/periphery metrics. The results of the study of mailing list data from MySQL in UCINET are presented in the next section.

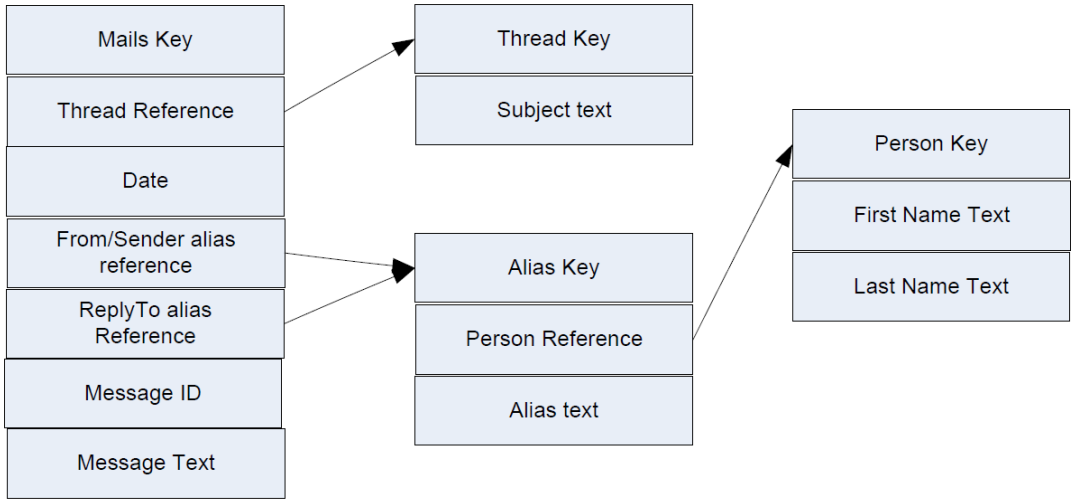

Figure 1: PostgreSQL Database Schema

As far as what has actually been built, the final result is a Perl script, Java program, and PostgreSQL data. The Perl script is one that can be re-used in the future to mine either the current MySQL mailing lists that it currently does on the site or any of the others quite easily. It also currently fills the PostgreSQL database, but can be quickly changed to fill any database. The Java program simply reads from a database and populates this common mailing list / forum database schema into Java objects which can then be manipulated. In its current form, the Java code produces output which is readable by UCINET. It can be easily configured to read from a different database and to write mailing list files for UCINET from different dates. Finally, the PostgreSQL is simply a database populated with all of the internal and bug forum posts from MySQL from the last ten years. It is an example of using the common schema for mailing lists and it can be mined by anyone for any purpose. The repository containing the code and the documentation for all of these pieces is open source and is located at the following website: http://code.google.com/p/mysqldatabasecoreperiphery/.

There were two main issues that occurred during the process of creating and building this application. The first issue was the issues with mining the mailing list. The mailing list was hard to parse because it contains over ten years worth of data and many things have changed over time, like the format of e-mails, format of names, etc. Since it's not stored in a common or an easy to parse format, all of these become tough obstacles and it requires significant human investment to ensure the data is of high quality. The second issue was the choice of tool to use to analyze the mailing list data. The team pursued NodeXL, discussed previously in the Background section, for a long time before realizing that the centrality measurements that it can give would not be enough to make a convincing case for a core/periphery structure. Therefore, the team switched to UCINET, which was harder to use but guaranteed to have the statistics needed. However, the harder user interface of UCINET, combined with the late switch in the timeline of our project, did create an issue that was hard to overcome.

Evaluation

This section presents all of the team's results and is followed by an in-depth discussion. Below is a table of displaying the main results. In the first column is the name of each release. The second column is the month and year of that particular release. The third column displays the results of a categorical analysis of the core and periphery structure of the MySQL communication structure as a snapshot at that date associated with each release. The fourth column displays the results of a continuous analysis of the core and periphery structure of the MySQL communication structure as a snapshot at that date associated with each release. Following the table and its discussion is a picture of the social network of the MySQL development team based on the communication repository. This is also followed by a discussion of it. Finally there will be discussions of both what these results mean and also strengths and weaknesses of the process used to go about obtaining them.

Table of Results:

Release Name	Release Date	Categorical C/P Metric	Continuous C/P Metric
mysql-3.20.32a.tar.gz	N/A	No Data	No Data
mysql-3.21.33b.tar.gz	6/1998	No Data	No Data
mysql-3.22.32.tar.gz	2/2000	No Data	No Data
mysql-3.23.57.tar.gz	6/2003	Jammed	Link
mysql-3.23.58.tar.gz	9/2003	*Link*	Link
mysql-4.0.26.tar.gz	9/2005	*Link*	Link
mysql-4.0.27.tar.gz	5/2006	Link	Link
mysql-4.1.21.tar.gz	7/2006	Link	Link
mysql-4.1.22.tar.gz	11/2006	Link	Link
mysql-5.2.0-falcon-alpha.tar.gz	1/2007	Link	Link
mysql-5.2.3-falcon-alpha.tar.gz	2/2007	Link	Link
mysql-6.0.0-alpha.tar.gz	4/2007	Link	Link

mysql-5.0.45.tar.gz	7/2007	Link	Link
mysql-5.1.22-rc.tar.gz	9/2007	Link	Link
mysql-6.0.2-alpha.tar.gz	9/2007	Link	Link
mysql-6.0.3-alpha.tar.gz	11/2007	Link	Link
mysql-5.1.23-rc.tar.gz	1/2008	Link	Link
mysql-6.0.4-alpha.tar.gz	3/2008	Link	Link

The table above has links to the log files for both the categorical and continuous core/periphery metrics for each release. The calculations are done on all communications that occurred from the previous release date up to that one. The first few have no data because the MySQL mailing list archive only starts in 2/2000. One of the links says "Jammed" because the UCINET program was unable to process this data and failed. For the most part, the log file links do show a core/periphery structure in the MySQL project. In some cases the structure does not resemble this, but in these cases it is usually when two releases are really close to one another, meaning perhaps a special structure has been created to respond to a few bug fixes. In addition, if it is bugs that are being fixed, some non-core members who are experts in these bugs might be brought in to analyze and fix the problem. Two examples which show the core/periphery structure at its best in this project are the two links that are starred in the table above.

The categorical core/periphery metrics are defined using the following settings in UCINET: The correlation algorithm with 50 iterations and a population size of 100 genetic algorithms. The resulting log file contains first a listing of which nodes would belong in the core and which in the periphery. It also contains a block matrix, which has four quadrants, one signifying the core and three signifying different types of peripheries. The final fitness density is shown, which is how well this network was able to be matched up to a traditional core/periphery structure. At the end of the file a density matrix is available, showing the density of each of the four quadrants. The continuous core/periphery metrics are unfortunately currently unavailable due to a floating point operation bug in UCINET.

Social Network of MySQL:

Release Name	Release Date	Social Network
mysql-3.20.32a.tar.gz	N/A	No Data
mysql-3.21.33b.tar.gz	6/1998	No Data
mysql-3.22.32.tar.gz	2/2000	No Data
mysql-3.23.57.tar.gz	6/2003	Link
mysql-3.23.58.tar.gz	9/2003	*Link*
mysql-4.0.26.tar.gz	9/2005	Link
mysql-4.0.27.tar.gz	5/2006	*Link*
mysql-4.1.21.tar.gz	7/2006	Link
mysql-4.1.22.tar.gz	11/2006	Link
mysql-5.2.0-falcon-alpha.tar.gz	1/2007	Link
mysql-5.2.3-falcon-alpha.tar.gz	2/2007	Link

mysql-6.0.0-alpha.tar.gz	4/2007	Link
mysql-5.0.45.tar.gz	7/2007	Link
mysql-5.1.22-rc.tar.gz	9/2007	Link
mysql-6.0.2-alpha.tar.gz	9/2007	Link
mysql-6.0.3-alpha.tar.gz	11/2007	Link
mysql-5.1.23-rc.tar.gz	1/2008	Link
mysql-6.0.4-alpha.tar.gz	3/2008	Link

The table above has links to the social network picture diagrams for each of the releases that are listed above. The social network diagrams were built by analyzing all communications that occurred from the previous release date up to that one. The first few have no diagram because the MySQL mailing list archive only starts in 2/2000. For the most part once again, the diagram links do show a core/periphery structure in the MySQL project. In some cases the structure diagram does not resemble this, but in these cases it is usually when two releases are really close to one another, meaning perhaps a special structure has been created to respond to a few bug fixes. In addition, if it is bugs that are being fixed, some non-core members who are experts in these bugs might be brought in to analyze and fix the problem. Two examples which show the core/periphery structure at its best in this project are the two links that are starred in the table above.

The social network diagrams are defined using the following settings in UCINET: UCINET data source (*.##H or *.##d) and 1-mode network. The graphs show the nodes as each programmer in the communication structure and a directional arrow the direction of communication between them. The thickness/darkness of the arrow is used to represent its magnitude.

The results just discussed above for both the table and the social network graph mean with respect to open source software and how it relates to DSD that MySQL, yet another open source project, has a core/periphery structure for the social make-up of their development team. Because of this, it reaffirms that this is indeed the structure that all open source projects seem to take on and is something that can be transferred to the DSD field, since they are so similar, and hopefully make a great impact there. DSD suffers from many of the same problems as open source, so if it were to develop this core/periphery structure, it may have the same successes as open source. This idea will be discussed more in the “Conclusion” section.

There are both some great strengths and also a few weaknesses in the team’s approach. Some of the great strengths of the research are that the data was put into a common format first, that the results are easily reproducible, and that proven products were used in part of the experiment’s pipeline. The data was first put into a common format, which is great for future researchers to build from in their experiments. It was very tedious to mine the MySQL mailing list, and this is something that took much longer than the team expected and had many nuances. By essentially converting the data into a common format by putting it into a database with a very common sense schema, it is more accessible for future researchers. The experiment is very reproducible because it is well documented earlier in the report and the common format and the use of UCINET, the process used to achieve results is pretty straightforward. Finally, another positive is

that proven products were used in the pipeline to obtain results. By using UCINET, this guarantees that proven core/periphery metrics that are touted by Borgatti and others are used instead of inventing new metrics. This is important to compare the results of this paper to other researcher's experiments as well.

Some of the weaknesses of the approach include that there may be some errors in the mining process and that it is now dependent on others. The mining process was very tedious. Over ten years and several versions of their mail servers, the format of people's names, e-mail addresses, etc. changed. In addition, social aspects such as etiquettes and other things changed, making it hard to parse efficiently. The process took a lot of time and several iterations, as much of the data required special cases of parsing. In addition, due to the time frame, the team cannot guarantee that every single thing was properly parsed from the mailing list as it was input into the database common format. In addition, the team's experiment is dependent on the work of Borgatti and on UCINET. While this is a strength as previously stated, it is also a weakness because if there end up being any issues with their research, this team's research would also be affected by it. In addition, to reproduce the work, their software is required – another dependency.

Conclusions

This experiment yielded some very important "lessons learned" from the entire project. The first big lesson the team learned was that mining repositories and mailing lists in particular can be very cumbersome and that there is a worthy argument for common formats, which the team discusses in a separate position paper which can be found listed in the "References" section. While the planned for approximately 50% of the time given for the project to data mining/collection, it ended up taking much closer to 70-80% of the total project duration, due to some of the nuances and just simple changes over a ten year span in how MySQL developers communicate, as described in the previous section. A second big lesson, learned throughout the course of this project is that open source and DSD are very similar. This topic has been discussed extensively throughout this report.

The most significant contribution of this paper is simply a full core/periphery analysis of the MySQL product at plenty of different release points. This analysis as well as the actual social network graph produced by the research team allows other researchers to see trends in the structure of the development team throughout the different releases. Not only does this let the team justify that there is indeed a core/periphery structure, but how the structure of programmers changes from one release to another can be correlated with actual changes and issues with each release to perhaps explain why the changes indeed occurred in the first place.

If the research team were to redo or restart this experiment and were given more time, there are a few changes that would be made, mainly based on hindsight of how the experiment concluded. If redone, the team would devote more time to the mining process and also would have settled on UCINET sooner. The mining process ended up running well over what anyone expected, which did affect the quality of the results that were able to be gathered in the analysis phase of the project. While the extended mining was important to ensure that the data taken from the mailing list was accurate, it did reduce the amount of time the results could be analyzed with core/periphery metrics. The team

also had settled on working with NodeXL for a while before ultimately switching to UCINET. This also took away some from the analysis phase of the project, as some analysis was done with a product no longer used.

However, the results yielded do show much promise and are accurate in showing the core/periphery structure of MySQL's development team over an 18 release period of their software product. This definitely reaffirms the notion that open source is core/periphery, provides a great source for other researchers to use to correlate the development team structure at each release with characteristics of the release itself, and also provides DSD projects with products or other characteristics similar to MySQL an opportunity to learn from the core/periphery structure and try to use it to improve their software development process.

Other than using the team's research, researchers could successfully extend this work by studying even more releases of MySQL, mining the repository more extensively, and taking more core/periphery statistics than just the ones described by Borgatti. The team only studied 18 pertinent releases of MySQL, but releases could easily study the many more out there. In addition, the mailing list could be mined more extensively, or rather checked more for the nuances and inconsistencies described in previous sections. Researchers could start with this team's common format database and just check it for accuracy, creating a few more special cases this team might have missed in its short timeline. Finally, this project relies heavily on Borgatti core/periphery metrics which is a positive, but also provides room for other researchers to analyze the database created here with other core/periphery metrics that are in existence or completely new ones.

References

Borgatti, S. P., & Everett, M. G. (1999). Models of Core/Periphery Structures. *Social Networks 21*, (pp. 375-395).

Borgatti, S. P., & Everett, M. G. (2000). Peripheries of Cohesive Subsets. *Social Networks 21* (pp. 397-407). Elsevier.

Borgatti, S.P., Everett, M.G. and Freeman, L.C. 2002. Ucinet for Windows: Software for Social Network Analysis. Harvard, MA: Analytic Technologies.

Borgatti, S.P. 2002. Netdraw Network Visualization. Analytic Technologies: Harvard MA.

MySQL Lists. (n.d.). Retrieved June 2, 2010, from MySQL: http://lists.mysql.com/

NodeXL: Network Overview, Discovery, and Exploration for Excel. *CodePlex Open Source Community*.

Stahlback, S., Osecki, J., & Kim, M. (2010). Benefits of Separating the Data Mining of Repositories from the Research Analysis Phase. *Distributed Software Development Position Paper*.