**Empirical Study with Boa:**

To validate the findings of Pingzer et al. and to achieve the research questions stated in the previous section, we propose a study that aims to construct developer-module networks for several large, open-source projects on GitHub. This study aims to answer both research questions posed in the previous section. Predominatly

***Dataset:***

Due to the many inherent limitations in the GitHub API, the study conducted in this project does not directly access project data from GitHub. As a result, this project will use GitHub data collected by the Boa project as the dataset to pull project data from. The reason Boa was selected for use in this study was due to its rigid schema across all projects, flexible Java API, and domain specific language. Further, the current snapshot of the Boa dataset is from September 2015 allowing us access to more recent GitHub projects. While Boa is fairly advantageous, several limitations are discussed in the Future Work section of this paper.

As such, we developed a Boa DSL query called *project search* that allowed us to search its data repository for projects based on the number of contributors, files, and ratio of regular commits to bug fix commits. Further, we developed Boa DSL query called *contribution data* that is able to pull all data required to build the contribution network. Specifically, this query returns a list of all files in the project, a list of contributors, and a list of files changed by each commit. These two scripts allowed us to easily download data that can be used to construct the developer-module networks used in the study.

Due to the sheer size of the Boa dataset and since each Boa query scans all projects regardless of whether a script targets a specific project, we opted to build a local database that is populated with the results of the Boa queries for each project. To illustrate the length of time required for query completion in Boa, the average run time of our contribution query on 10 different projects is around 5 minutes. As such, by developing a local database, pertinent project details are able to be quickly accessed with or without an internet. Further, interesting projects can be cached and used later.

***Methodology:***

To answer both the research questions stated this study has been broken into several sections.

Once representative projects have been selected from Boa, all relevant data ne

***Software Tool:***

To efficiently carry out the methodology as described in Section 4.1, a software tool called *NetworkMine* was developed that automates the majority of the process. Written in Java, the *NetworkMine* tool is comprised in two different parts; a Boa miner which pulls all software repository details from the Boa website, and a social network builder which constructs the developer-module network and computes all relevant metrics. This division is important since, in future iterations of the tool, different GitHub mining repositories can be swapped for Boa. The remainder of this section will explain the two sections of the tool in detail.

*Boa Miner:*

The Boa miner component is responsible for pulling project data from Boa’s servers into a local SQLite3 database. The tool allows for users to search for projects based on the number of contributors and files and can run the specified search of any of Boa’s three different GitHub datasets. Once a collection of projects is found that match a user’s desired search parameters, the user can download any number of the returned projects. Figure 1 shows a screenshot of the tool allowing users to select projects to download.

The Boa miner component is written in Java and requires the SQLite JDBC driver and Boa API developed by Iowa State.

*Social Network Builder:*

The social network builder component carries out the brunt of the methodology. Using the project repository data downloaded from the Boa miner component, the social network builder constructs a developer-module network for a specified project and then computes the betweenness, closeness, and degree centrality metrics for that graph. Once complete, the user is then able to see the Spearman correlations for each of those social network metrics and the results of the logistic regression run on the graph. Unlike the Boa miner component, this part of the tool is accessed via the command line. The output of the program is a comma-separated values (CSV) file that contains the results of the Spearman correlation and logistic regression functions.

While the tool was developed during this study, it relies on several open-source libraries including Java Universal Network/Graph Framework (JUNG) and Apache Spark. JUNG is used to develop the graph and compute centrality metrics while Apache Spark, while normally used as a cluster computing framework, is used to perform the logistic regression.

**Results:**

This section describes the results that were obtained after running

***Selected Projects:***

While the Boa dataset has a large number of GitHub software projects that could be used in this study, we decided to limit our scope by focusing on a subset of these projects. While this project aims to test whether developer-module networks are able to predict failures over a large number of GitHub projects, too many projects would take far too long to compute and also dilute the results presented in this paper. As such, we decided to select fifteen large projects that would form a good representative sample. Fifteen is a good value since it will allow us to test our hypothesis on a variety of different projects to eliminate the possibility of chance while not being too many. Further, while we wanted larger projects with many files and contributors, we decided not to select the top fifteen largest projects from the Boa dataset since many tended to be divergent forks of the Android project. Including projects of a similar nature would result in duplicate results that could spoil the conclusion.

Based on all these considerations, we selected fifteen projects that each had over 150 contributors and 1,000 files and were from different software domains. Table 1 shows several descriptive statistics about the number of files and contributors for each of the project. Of our project dataset, <num> of the projects are in the software development domain, <num> of the projects are Android-based, and <num> are web projects and technologies.

***Spearman Correlation:***

Similar to Pingzer et al., for each project that we tested, we computed the Spearman correlation between each of the metrics to determine whether there was a correlation between each of these metrics. In it, we decided to set values of [-0.7, 0.7] and above to be strongly correlated components. Table 1 shows the results of the Spearman correlation test for each of the five metrics that were computed in the study. To compress the results, we present the mean of each Spearman correlation across all the projects tested in the study. Each of the correlations calculated are significant at the 0.01 level.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **# of Commits** | **# of Bug Fixes** | **Betweeness Centrality** | **Closeness Centrality** | **Degree Centrality** |
| **# of Commits** | 1 |  |  |  |  |
|  | **# of Bug Fixes** | 1 |  |  |  |
|  |  | **Betweeness Centrality** | 1 |  |  |
|  |  |  | **Closeness Centrality** | 1 |  |
|  |  |  |  | **Degree Centrality** | 1 |

Importantly, some of the projects did not show strong correlations between some of the metrics. For instance, Table 1 shows the Spearman correlations for the <project> project. Some of the correlations are close to 0 indicating that the values have no correlation at all. This lack of a positive correlation between bug fixes and the centrality metrics indicates that logistic regression may not work well at predicting