**Introduction:**

As software projects continue to increase in size and complexity, many companies desire methods that can predict defects in software components prior to release. Early defect prediction has the potential for a more focused quality assurance process and for early identification of defect-prone components. For instance, if defects are predicted to be prominent in several networking binaries, quality assurance efforts can allocate more funding and effort on testing and reviewing those predicted binaries. If used properly, these failure prediction models have the potential in saving a software company money and allowing for fewer bug-prone binaries. Since this area has much promise, much research has been conducted that aims to determine software metrics that can be used to discover components that are likely to possess defects. Much work in this field has explored the use of various types of metrics including organizational metrics [1], social networks [2], and distributed development [3]. These studies use metrics such as code churn, code complexity, socio-technical networks, and code ownership.

While these studies are encouraging, they share several commonalities that can make their results slightly problematic. First, many of these studies are published by researchers at Microsoft Research and possess the common software domain of Windows Vista. While the use of Windows Vista as the domain to test these prediction models is not flawed in itself, there are As such, while these metrics may be valid predictors of failure-prone binaries for Windows Vista, many of these results cannot be extrapolated and used as predictors in other software domains. Second, since many of these studies have been published, there has been an emergence in ultra-large mining projects that pull data from code repositories such as GitHub and SurgeForge. Using techniques such as MapReduce and NoSQL, projects such as Boa and GHTorrent are able to efficiently store and process vast quantities of repository data [4]. To illustrate the scale of the data that these projects collect, GHTorrent’s dataset for the Mining Software Repository (MSR) challenge is over \_\_\_\_ GB uncompressed that is intended to be used locally. Though it was possible before, researchers now have an easier time being able to analyze and collect metrics from a larger array of software projects.

One study performed by Pingzer et al. [5] looks at the use of a particular social network called developer-module networks as a failure-predictor in Windows Vista binaries.

**Methodology:**

To achieve the research questions stated above, this project will evaluate several large scale GitHub repositories and compare a social network of contributors and components to the repository’s GitHub Issues data. This project will largely follow the methodology presented in the paper *Can Developer-Module Networks Predict Failures?* except with on a dataset which includes a variety of open-source projects. This section describes three major areas that will be investigated and researched through the duration of the project. These are the repository dataset that will be used, the experimental evaluation, and the software tool. This section describes each of these three areas in detail.

**Dataset:**

With memory, storage, and processing power increasing at a dizzying pace and with new data processing frameworks including MapReduce and Spark, researchers in the field of software engineering are able to mine larger repositories of code data. Despite this advantage, developing mining programs using frameworks such as MapReduce tends to be far too low level and time-consuming for the needs of researchers. Since much research in software evolution tends to require the use of large quantities of data, many different solutions have been developed to aid researchers in mining code repository data. One such solution, developed by researchers at Iowa State University, is a domain specific language (DSL) called Boa. By using a well-defined schema, users can write queries that can pull data from different version control systems in a uniform way [4]. Based on their schema, queries can be written that include data pertaining to software projects, repository contributors and code changes.

What makes this attractive is that, in addition to the Boa DSL, researchers at Iowa State developed a repository of GitHub data from September, 2013 that contains over 70,000 repositories [5]. While this is a large number of repositories, the data is stored on Iowa State's servers and Boa queries targeting the data are run on a Hadoop cluster. Despite the drawback of not being local, there are advantages such as being able to offload computational power and being able to submit Boa queries through a rich web interface, Eclipse plugin or Java API. As such, for this project, we propose to use Boa and its associated GitHub repository data. Based on this, this project will use the Boa DSL and the accompanying GitHub data from September 2013 to achieve the stated goals.

While Boa has many advantages that make it preferable to other GitHub data dumps, one limitation is Boa’s lack of GitHub Issue data in its GitHub dataset. Since our project relies on this issue data, we will need to overcome this problem. To do this, the project will build a simple program that uses the GitHub API to mine a repository’s Issue data. The GitHub API is service provided by GitHub that allows clients to receive desired GitHub data in the form of JSON [6]. The service is easy to access and there are many different open-source software libraries that automate parsing this data. Based on these factors, our project will obtain the Issue data using either raw JSON requests or by using an external open-source library.

**Experimental Evaluation:**

**Software Tool:**

An extra component delivered by this project will be a software tool that will attempt to automate part of the methodology described above. It is expected that this tool will be able to connect directly to Boa’s external servers, pull desired GitHub repository data using customized Boa queries, mine GitHub Issue data for that repository and build the developer-module graphs. This tool will have a graphical user interface that will visualize the computed social network for the supplied GitHub repository.

While most of the specifications of the software tool will be determined during the project lifetime, there are several already-decided aspects. First, this tool will be developed using Java because of language limitations for the Boa API and since Java boasts a large number of open-source graph packages including Java Universal Network Graph Framework (JUNG) and Gephi. With much research already conducted into the features of these open-source graphing libraries [7], this project will also determine which is best for use in the software tool.

**Challenges:**

As with any project, there are a variety of challenges that will need to be overcome for our project to achieve its stated goals. These challenges can be split into several categories: *data challenges* and *methodology challenges.*

Data challenges refer to problems encountered when dealing with the data itself. Since we will be dealing with GitHub repositories, there are certain considerations that need to take place when conducting the experiment. Previous research has found several challenges that can arise when mining GitHub for data. Since GitHub hosts professional open-source projects and hobbyist projects alike, most repositories tend to have very little commit activity [8]. Luckily, since our project will focus on larger, more active projects, this should not be a problem. One challenge associated with our project may come from the fact that GitHub is a pull-based development system. For instance, the Rails project has over 8,000 project forks and over 50,000 commits [8]. However, of those 50,000 commits, GitHub notes that there are only 34,000 commits to the core Rails repository, there are only 34,000 commits present. As such, this project must consider forked repositories as well to get a bigger picture of commit and contributor information.

Methodology challenges refer to issues that are likely to be encountered when conducting the experiment. One major challenge that may occur will be linking the mined Issue data to the data collected from GitHub. For instance, it is likely that the two datasets will arrive in differing formats. This means that some sort of process that swizzles the data will have to occur so that the two collected datasets are compatible with each other. Figure 1 illustrates this problem. The data swizzling will need to take place before we can create a combined table that contains all this data.

**Timeline:**

To achieve our objectives as stated here, this project has been split into several small milestones. Importantly, while the submission deadline for this project is due on the 29th of March, we aim to continue our work to further work on the software tool. This section highlights our proposed timeline and describes each of the milestones in detail.

*Milestone #1 (March 7th, 2016) –*

This milestone is concerned with understanding the Boa DSL and GitHub API in greater detail. In particular, we would like to work on developing queries that allow us to pull relevant data from GitHub repositories. By the end of this milestone, our project will have the Boa queries completed and will have started developing a tool that allows us to pull Issues from the GitHub API.

*Milestone #2 (March 14th, 2016)* –

In this milestone, there are two major goals that will be achieved. First, we will identify a collection of GitHub repositories that would be ideal to analyze. Repositories that utilize GitHub’s Issue tracker are ideal since this project revolves around pulling defect information from this data. Secondly, at this milestone, it is expected that the tool that mines GitHub issue data is complete and able to properly store archive this data in some data store.

*Milestone #3 (March 29th, 2016)* –

There are multiple goals that must be achieved by this milestone. Mostly, this milestone is concerned with the completion of research that attempts to answer the research questions presented in this proposal. Further, at this milestone this project will deliver a paper summarizing our findings and a presentation. Additionally, if time permits, the development of the software tool will be underway at this state and a usable demo will be provided.

*Milestone #4 (End of April, 2016) –*

While outside of the project deadline, this milestone is concerned with the completion of extra features including the software tool. By this point, the tool should be able to automatically pull data from Boa’s Java interface, visualize GitHub contributions and contributors, and develop social network information from said Boa data. While some of these features are expected to be completed in milestone 3, this milestone is focused on the completion of all features.

**Conclusion:**

In this proposal, we present a study that examines developer-module networks and failure prediction across a wide variety of open-source projects. This project aims to extend upon Pingzer, et al.’s research [9] by building developer-module networks for several large scale projects and through the development of a tool that can automate the building of the developer-module network.

# References

|  |  |
| --- | --- |
| [1] | N. Nagappan, B. Murphy and V. Basili, "The Influence of Organizational Structure on Software Quality: An Empirical Study," in *International Conference on Software Engineering (ICSE)*, New York, NY, 2008. |
| [2] | C. Bird, N. Nagappan, P. Devanbu, H. Gall and B. Murphy, "Putting It All Together: Using Socio-technical Networks to Predict Failures," *Proceedings of the 20th IEEE international conference on software reliability engineering,* pp. 109-119, 2009. |
| [3] | C. Bird, N. Nagappan, P. Devanbu, H. Gall and B. Murphy, "Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista," in *International Conference on Software Engineering (ICSE)*, Vancouver, 2009. |
| [4] | R. Dyer, H. A. Nguyen, H. Rajan and T. N. Nguyen, "Boa: Ultra-Large-Scale Software Repository and Source Code Mining," *ACM Transactions on Software Engineering and Methodology,* vol. 1, no. 1, pp. 1-33, 2015. |
| [5] | Laboratory of Software Design, "The Boa Programming Guide," Iowa State University, 27 July 2015. [Online]. Available: http://boa.cs.iastate.edu/docs/index.php. [Accessed February 2016]. |
| [6] | G. Gousios and D. Spinellis, "GHTorrent: Github’s Data from a Firehose," in *Mining Software Repositories (MSR)*, Zurich, Switzerland, 2012. |
| [7] | J. R. Harger and P. . J. Crossno, "Comparison of Open Source Visual Analytics Toolkits," in *SPIE Conference on Visualization and Data Analysis*, 2012. |
| [8] | E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German and D. Damian, "The Promises and Perils of Mining GitHub," in *Mining Software Repositories (MSR)*, Hyderabad, India, 2014. |
| [9] | M. Pingzer, N. Nagappan and B. Murphy, "Can Developer-Module Networks Predict Failures?," in *Special Interest Group on Software Engineering (SIGSOFT)*, Atlanta, Georgia, 2008. |