**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 build a developer-module network for each of these projects. These networks will then be compared to a repository’s GitHub Issues data which will be used to establish defects. This project will largely follow the methodology presented in the paper *Can Developer-Module Networks Predict Failures?* but will examine this phenomenon on multiple projects. To achieve this proposal’s stated goals, there are three main areas that need to be considered in this proposal. These are the repository dataset that will be used to build the developer-module networks, the experimental setup and evaluation, and the software tool. This section describes each of these three areas in detail.

**Dataset:**

As memory, storage, and processing power continuing to increase and with new data processing frameworks available that include MapReduce and Spark, researchers can mine and process lots of data from source-code management (SCM) repositories. However, despite the advantage of having such storage capabilities and frameworks, it can be fairly complicated to manually mine websites like GitHub and efficiently store all gathered data. Further, since some research topics in software evolution tend to require querying repositories for specific data, several different tools and solutions exist that aim to assist developers in querying SCM data efficiently and quickly. One tool, developed by researchers at Iowa State University, is a domain specific language (DSL) called Boa. By using a well-defined schema that is independent of different SCM repositories, users can write queries that pull data from different systems in a uniform way [4]. Based on the Boa schema, queries can be written that can target Git, Mercurial, and Subversion repositories among others. Further data pertaining to software projects, contributors and code can be queried.

While the Boa DSL itself is a powerful DSL for mining large repositories, the Boa project also contains a snapshot of GitHub and SurgeForge repository data. For instance, the GitHub snapshot contains data from September, 2013 and it has over 70,000 repositories stored [5]. What makes this further desirable is that, though these snapshots are very large, the data is completely stored on Boa’s servers. Therefore, when a user submits a Boa query, the query is compiled on Iowa State’s servers and run as a MapReduce program on a remote Hadoop cluster. Therefore, even though this project would not have local access to this data, there is the advantage of being able to offload computational power to a remote cluster and being able to access the Boa interface through a rich web interface, Eclipse plugin or Java API. Based on these factors, this project will use the Boa DSL and the accompanying GitHub data from September 2013 to collect project and contributor data that will allow us to build the developer-module network. Prior to submitting this proposal, we ran several example queries on the full GitHub snapshot and were able to determine that the data is well-suited for the needs of this project.

While Boa has many advantages that make it preferable to other projects that allow users to mine GitHub data, there is one major limitation that has to be addressed. Since Boa attempts to use a standardized schema for all SCM services, there is no way to obtain GitHub Issue data for selected repositories. Since this project relies on the use of Issue data to determine components that are failure-prone, Issue data needs to be pulled from an additional source. As such, we propose building 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]. Further, this program will rely on different open-source software libraries that exist which will allow our program to pull specific data from GitHub and cache it in a database. While the specifics of this program has not been fully determined, it is expected that many of the libraries and classes used in it will be ported over and used in the software tool described in Section <SEC\_NUM>.

As an aside, while GHTorrent was another potential candidate for use as the dataset in this project, the GHTorrent project was suspended as of February 27th and all project data has been removed until further notice due to privacy concerns.

**Experimental Evaluation:**

**Software Tool:**

While not the main purpose of this project, an extra component that will be worked on is a software tool that will attempt to automate part of the methodology described above. Since this tool is not the main focus of this research project, it will likely not be fully completed by the time of the class presentation and, instead, will be worked on afterwards. The functionality of the tool can be divided into several different segments. Once complete, this tool will be able to connect directly to Boa’s servers, execute Boa queries, retrieve their results, and from it build the developer-module network automatically for any specified repository. Further, to better illustrate the size and layout of the network, this tool will have a graphical user interface that will display network of the desired repository. Finally, since this tool is planned to automate the described methodology, the user will not have to write their own Boa queries to pull the required data from Iowa State’s servers.

Though this project may seem fairly ambitious, there are several considerations that have to be made which will assist in production. First, since Boa has an API, having this software tool connect to Boa and run queries is fairly trivial. Prior to writing this proposal, we wrote an example program that carries out much of said functionality. Next, since Boa queries will have to be written during the manual experiment, these queries can be recycled and integrated into this tool. Finally, since Java boasts a large number of open-source graph packages including Java Universal Network Graph Framework (JUNG) and Gephi, building the developer-module network will be simplified. Since many Java graphing libraries have been subject to much research regarding their features [7], this project will also need to determine which is best for use in the software tool.

**Challenges:**

As with any project, there are a variety of challenges that we will need to overcome for our project to achieve its stated goals. For this project, the challenges that have been identified can be split into two 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 this experiment. Previous research has found several issues that can arise when mining GitHub for data. Since GitHub hosts both professional, open-source, and hobbyist projects, the vast majority of repositories tend to have very little commit activity [8]. Luckily, since our project aims to focus on larger, more active projects, this should not be a problem. Another challenge associated with GitHub may come from the fact that it 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, there are actually only 34,000 commits to the core Rails repository. As such, this project must also consider the activity in 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 Issue data for a particular project to the data collected from Boa. Since it is likely that the information pulled from these two sources will likely have different schemas, much consideration must be taken into how we will process this. It is possible that there is some process that will swizzles the data so that the two collected datasets are compatible with each other. Figure <FIG> illustrates this problem. The data swizzling will need to take place before we can extrapolate information from this dataset.

**Timeline:**

To achieve the objectives stated in this proposal, this project has been split into several small milestones. Importantly, although the submission deadline for this project is on the 29th of March, we aim to continue this project to further develop the software tool. This section highlights the 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. Projects that utilize GitHub’s Issue tracker are required since this project determines defects from this data. Secondly, at this milestone, it is expected that the tool that mines GitHub Issue data will be complete and able to properly 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 we will deliver a paper summarizing our findings. 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 the first prototype of 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 the data. While some of these features are expected to be completed in milestone three, 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 by developing 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. |