THESIS

APPLICATION OF SOCIAL NETWORKING ALGORITHMS IN PROGRAM ANALYSIS: UNDERSTANDING EXECUTION FREQUENCIES

Submitted by

Minhazur Rahman

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2011

Master's Committee:

Advisor: James Bieman

Robert France
Daniel Turk

ABSTRACT

APPLICATION OF SOCIAL NETWORKING ALGORITHMS IN PROGRAM ANALYSIS: UNDERSTANDING EXECUTION FREQUENCIES

There may be some parts of a program that are more commonly used at runtime, whereas there may be other parts that are less commonly used or not used at all. In this exploratory study, we propose an approach to predict how frequently or rarely different parts of a program will get used at runtime without actually running the program. Knowledge of the most frequently executed parts can help identify the most critical and the most testable parts of a program. The portions predicted to be the less commonly executed tend to be hard to test parts of a program. Knowing the hard to test parts of a program can aid the early development of test cases.

In our approach we statically analyse code or static models of code (like UML class diagrams), using quantified social networking measures and web structure mining measures. These measures assign ranks to different portions of code for use in predictions of the relative frequency that a section of code will be used.

We validated these rank ordering of predictions by running the program with a common set of use cases and identifying the actual rank ordering. We compared the predictions with other measures that use direct coupling or lines of code. We found that our predictions fared better as they were statistically more correlated to the actual rank ordering than the other measures.

We present a prototype tool written as an eclipse plugin, that implements and validates our approach. Given the source code of a Java program, our tool computes

the values of the metrics required by our approach to present ranks of all classes in order of how frequently they are expected to get used. Our tool can also instrument the source code to log all the necessary information at runtime that is required to validate our predictions.

ACKNOWLEDGEMENTS

I would cordially like to thank Dr. James Bieman for his immense help and support as my research advisor and mentor. I highly appreciate his help in influencing, cultivating and fine tuning my research ideas, and his detailed and prompt feedback on my reports and findings.

I also thank my committee members, Dr. Robert France and Dr. Daniel E. Turk, for their careful study of this thesis and constructive feedback.

I am greatful to my professors Dr. Shrideep Pallickara, Dr. Sudipto Ghosh, Dr. Charles Anderson, and Dr. Dan Cooley for teaching me courses that greatly helped me in my research.

Finally, I thank Sharon and all the other staff members of our department for helping me in the official matters.

This thesis is dedicated to my mother Mrs Rabeya Rahman, my father Mr Abdur Rahman, my sister Dr. Rumana Rahman, and my nephew Raihan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Good Software quality in terms of number of faults has always been strongly desired by software users and customers. This has necessitated the research on measurement and prediction of software quality. Computer Scientists in the field of software engineering have emphasized the need to be able to measure, estimate or assess various aspects of software quality and today a huge section of research in software engineering focuses in this area [23], [13], [8], [10]. One such aspect of software quality is frequency of code usage that pertains to the relative frequency that different sections of the code will be used at run time. We refer to the number of times a certain section(statement, method or class) of code gets executed in a given run of the program as execution frequency. Execution frequency can affect software quality in terms of testability and presence of defects. Sections of code that have high execution frequency tend to be more testable [9] and are probably the most critical parts of the program. A fault in these sections can potentially have far reaching consequences, as these sections may be used directly or indirectly by several other parts of the software. A modification in frequently executed code may cause cascading changes in behavior all throughout the software. On the other hand, sections of code that are rarely executed tend to be hard to reach, or less testable [9], and as a consequence faults in this code can be hard to find. Knowledge of execution frequencies even before the software is fully implemented can be used to make decisions about its testing strategy and maintenance

cycle. Knowledge of hard to test parts can aid building test suites. Knowledge of the most critical parts can benefit decisions on maintenance by prioritizing which parts should undergo changes.

1.1 Problem

Before we can predict execution frequencies, we need to have a way to directly measure execution frequencies for different portions of the software to address the above concerns. A distribution of the execution frequencies can be obtained by running the "most common" use cases that the software was developed to support. For instance, profiling tools that report execution frequencies, are often used in fault localization techniques [22]. Such processes employ dynamic analysis which can be a relatively slower process, and pertains to particular inputs or execution traces [5]. Arisholm et al [10] showed how precise measurements can be obtained from dynamic analysis. However their analysis can commence only after the source code and test cases are all available.

1.2 Approach

A social network is a structure that comprises a set of actors, who are related to one another by different kinds of relationships or inter-dependencies such as friendship or common interest. Scientists in the field of social sciences like Freeman [15] or Faust [1] used or came up with Social Networking algorithms to analyze such social networks. Their algorithms can assign numerical values to an actor within the network that convey some notion of 'importance' of that actor based on the summary of relations this actor may have to other actors. Our approach relies on computing the values of some of these social networking metrics on graphs representing code or static code structures and use them to predict run time execution frequencies. We analyze the Compilation Unit Graphs [17] that can be derived from software design

artifacts or reverse engineered from source code. The analysis involves computation of measures of importance of a vertex based on summarizing its relations with other vertices within the Compilation Unit graph [1]. We feel execution frequencies could be correlated to these measures of importance, as both attributes are affected by structural relationship among vertices within the graph. We restricted our research to object oriented software and selected three systems of varying sizes, to compute and study the distribution of the values of these measures. We performed experiments to study the correlation between these values measured by an entirely static procedure with the actual value of execution frequencies obtained by running use cases. Our experiments also included relevant statistical methods to determine to what extent these measures can predict execution frequencies. We developed an eclipse plugin INSAT to automate the entire process of computing the values of the social networking metrics and instrumenting the dynamic analysis meant to compare our predictions with the actual results.

1.3 Contributions

The research contributions of this thesis are as follows:

1. By applying social networking algorithms on static representations of source code, we could predict the most or least used parts of a program

2. We identified tendencies towards the power law in the distribution of the values of social networking metrics in models of source code.

3. We developed a tool called INSAT that automates the previous two processes.

1.4 Organization of Thesis

The thesis is organized as follows. In chapter 2 we present the existing concepts that we have applied in our research and other related works. In chapter 3 we describe our

approach and explain the design of our experiment. The experiment design includes the hypotheses, independent and dependent variables in our experiment, systems used, and strategy to validate our findings. In chapter 4, we show preliminary results and their visualizations to identify trends in them. We employ statistical methods to test our hypotheses from the results that we obtained in chapter 5. Chapter 6 talks about different types of threats to validity of our experiment and inferences. We present our conclusions and talk about possible future extensions to our research in chapter 7.

Chapter 2

Related Work

A considerable body of research has focused on the prediction of dynamic quality attributes from static quality attributes [13], [5], [11]. However, studies involving application of social networking metrics to analyze software are a relatively new concept. In this chapter we review the existing body of research related to our study, and the concepts that we borrowed.

2.1 Prior work on Execution Frequencies

Execution Frequency of a section of code is an internal software attribute whose actual value can be measured at run time. Profiling tools [35] report execution frequencies of program units at different levels of granularity like statement, branch or method. Information provided by such run time profiling tools are widely used for measuring quality of test cases, performance optimizations of programs, or detecting memory leaks. Harrold et al [21] studied various forms of program spectra and related Branch Count Spectra, which is the execution frequency of a conditional branch of a program, as a heuristic to analyze and compare program behavior. Renieris et al [22] used execution frequencies of different lines of code in a program as a form of program spectra that helps localize faults.

2.2 Social Networking Measures

A social network is a structure that comprises a set of actors, who are related to one another by different kinds of relationships such as friendship, common interest etc. We have analyzed social networks by computing measures of 'prestige' for each actor in the network. Prestige can be loosely defined as a measure of 'importance' of an actor in the network based on a summary of the structural relationships that this actor might have with other actors inside the network. The formal definition of prestige varies as different algorithms that focus on different aspects of relationships between actors are used to compute prestige. Prestige computation of a vertex in a graph takes into account different aspects pertaining to the location of the vertex within the graph. Various attributes of a vertex, like the number of incoming or outgoing edges may lower or raise the prestige of several vertices. Other aspects like the likelihood of a vertex falling in the path between two other vertices can also vary prestige.

In the next few sections we outline the different algorithms that will be used for defining and computing the prestige. We apply these algorithms to the graphs shown in Figure 2.1. Tables 2.1 and 2.2 consists of the values of measures of prestige as computed by different algorithms. We henceforth refer the graph to the left of Figure 2.1 as the star-graph and the graph to the right as the line-graph.

2.2.1 Degree Centrality

This is the simplest measure for prestige of a vertex among the algorithms used in this study. Degree Centrality is often used as a simple measure to find the 'importance' of vertices in a network, under the assumption that important vertices have more ties with other vertices within the network [1]. Perhaps the most relevant application of the degree centrality measure is the Coupling Between Objects(CBO) metric used in our research. The CBO metric was proposed by Chidamber et al [13] as a coupling

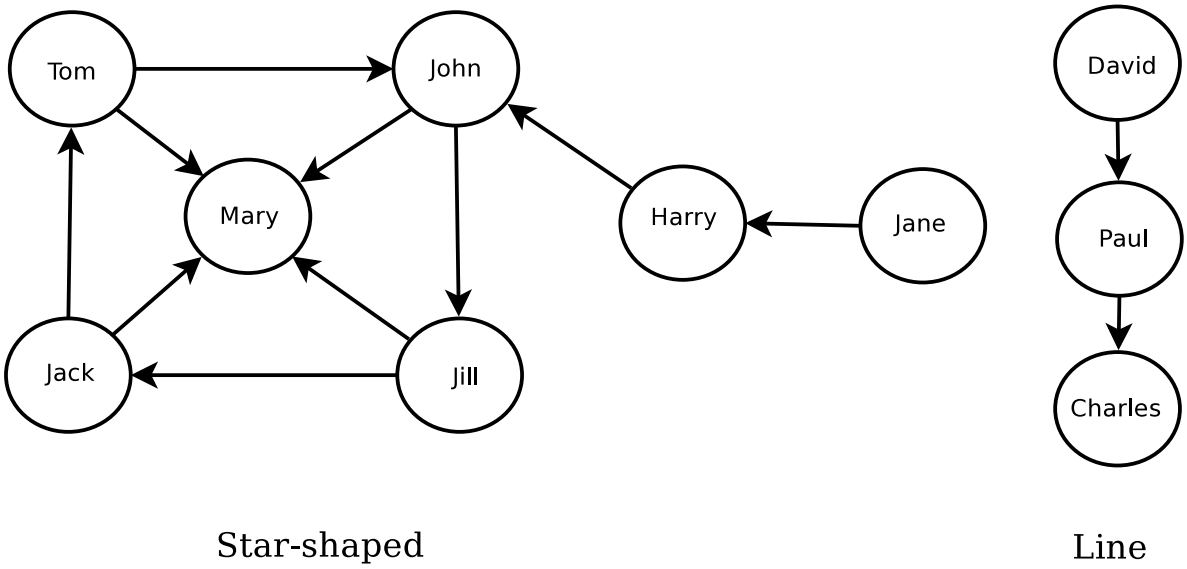

Figure 2.1: Graphs used to study the measures. Star Graph(left) and Line Graph(right)

measure and been used to study object oriented design [13], and ripple effects in changes [14]. Vertex A is said to have higher prestige than vertex B if vertex A has a higher number of "connections" than vertex B. The connections of a vertex can be measured by counting the number of the edges emanating from or directed at this vertex. This measure does not distinguish between incoming and outgoing edges. In the context of object oriented software, Degree Centrality can also be defined as the sum of the number of the afferent and efferent coupling [24] of the class this vertex represents. This is also the only measure among those used in the study that ignores indirect coupling. We have referred to degree centrality metric as 'CBO' in later chapters, as it is identical to the CBO metric.

In the star-graph Mary has four connections and hence has the highest prestige as per degree centrality whereas, Harry and Jane have the lowest. In the line-graph Charles and David have the same prestige which is lower than that of Paul. Given a

Graph $G = (V, E)$ where $v \, \epsilon \, V$, the degree centrality $C_d(v)$ of v is given by [1]:

$$C_d(v) \; = d(v) \; where \; d \; is \; the \; degree \; of \; the \; vertex.$$

2.2.2 Betweenness Centrality

The Betweenness Centrality(BC) measure, introduced by Freeman [15], is based on the tendency of a vertex falling on the shortest path between two other vertices. BC has been used to study and infer information about outbreak of epidemic diseases [16]. Unlike CBO, the betweenness centrality measure does not ignore relationships between non-adjacent vertices. Two vertices are said to be non-adjacent if there is no direct edge between the two of them. The BC of vertex u is the count of the number of times u is visited by the shortest paths in between two other vertices say v, w such that $v \neq u$, $w \neq u \; and \; v \neq w$. Mathematically, $BC(v)$ of a vertex v can be expressed as [2]:

$$BC(v) = \sum_{\substack{u,w \epsilon V \\ u \neq v \neq w}} \frac{\sigma_{uw}(v)}{\sigma_{uw}}$$

where σ_{uw} is the number of shortest paths between vertices u and w, and $\sigma_{uw}(v)$ is the number of shortest paths between vertices u and w that visit v. Computation of this measure for a vertex u may require the following steps:

- List all pairs of vertices (v, w) other than vertex u and $v \neq w$.
- List all shortest paths connecting v and w.
- Count all these paths that visit vertex u.

John has the highest BC in the star-graph in Figure 2.1 as several shortest paths (for e.g. $Jane \rightarrow Harry \rightarrow John$ or $Tom \rightarrow John \rightarrow Mary$) visit John. Mary has zero BC as Mary is a dead end vertex in the graph and does not have a single outgoing edge. In the line graph, the only path involving more than two vertices is $Paul \rightarrow David \rightarrow Charles$ and hence, $BC(David) = 1$.

Table 2.1: Values of different measures of prestige for Star Graph

Vertices	PR	DC	$HITS$	BC
Mary	0.28985125	4.0	0.057700157	0
John	0.19052342	4.0	0.4799442	11.0
Jill	0.13759725	3.0	0.4799442	7
Jack	0.11510362	3.0	0.4799442	5
Tom	0.10554383	3	0.5166229	3
Harry	0.10475586	2	0.18124534	5
Jane	0.056624793	1	0.07669604	0

2.2.3 HITS

Hypertext Induced Topic Search abbreviated to HITS is a search algorithm that was proposed by Kleinberg [3] to rate a web page based on the importance of its content and links to other web pages that it points to. The importance of the content pertains to relevance of the topic that is being searched for. The algorithm assigns two different scores to each node, the hub score and the authority score. The hub score is a measure of the values of its links to other nodes whereas, the authority score is a measure of the value of its contents. Since we are interested in the different relations between vertices within a graph we assign the hub score to the prestige of a node. The motivation behind creation of HITS was to search through web pages, but it owes its origin to social network analysis.

The hub and authority scores are defined recursively in terms of one another. "Good authorities are pointed to by good hubs and good hubs point to good authorities." Given a graph $G(V, E)$ where $v \, \epsilon \, V$ the hub and authority scores can be expressed as [12]:

$$x_v = \; authority \; score \; for \; v$$

$$y_v = \; hub \; score \; for \; v$$

$$x_v = \sum_{pages \; that \; point \; to \; v} y_v, \qquad y_v = \sum_{pages \; that \; v \; point \; to} x_v$$

$$x^{(k)} = L^T y^{(k-1)}, \qquad y^{(k)} = L x^k$$

Table 2.2: Values of different measures of prestige for Line Graph

Vertices	PR	DC	$HITS$	BC
Charles	0.47441217	1	0.20415911	0.0
Paul	0.34117106	2	0.6922135	1.0
David	0.18441679	1	0.6922135	0.0

$$where\ L(u,v) = \begin{cases} 1 & if\ there\ is\ a\ link\ from\ i\ to\ j \\ 0 & otherwise \end{cases}$$

2.2.4 Page Rank

The Page Rank(PR) algorithm was conceived by Brin and Page [6] and is used by the Google search engine to weight hyperlinked documents. Just like HITS PR also has its roots in social network analysis. The weight of a web page represent the probability of a random surfer randomly clicking on links, to arrive at this web page. The results of a search query are presented to the user in order of their weights assigned by the Page Rank algorithm. In the context of a graph, the vertices are represented as web pages and the edges are represented as hyperlinks in between web pages. Given a graph $G = (V, E)$ the page rank of a vertex $v\,\epsilon V$ is recursively defined as a normalized sum of the page ranks of the vertices that have an outgoing edge directly on v. Mathematically page rank of vertex v can be expressed as [6]:

$$PR(v) = \sum_{u \epsilon B_v} \frac{PR(u)}{L(u)}$$

where $L(u)$ is the number of outgoing edges of vertex u, and B_v is the set of all vertices that have an outgoing edge pointing at v. There can be situations where a graph might have a vertex that does not have outgoing edges. It is assumed that when the surfer hits a dead end web page(pages with no outgoing hyper links), the probabilities of all other pages to be the next visited page are equal. The page rank algorithm also introduces a damping factor d which is the probability that the random

surfer would continue clicking and not just stop. The page rank of a vertex v with the damping factor included can be expressed as:

$$PR(v) = \frac{1-d}{N} + d \sum_{u \epsilon B_v} \frac{PR(u)}{L(u)}$$

Mary and John have the highest PR in the star-graph in figure 2.1. This means, among all paths in the star graph, among all vertices Mary is the vertex that is visited the most. Jane has the lowest page rank as Jane is visited by only those paths that start at vertex Jane. In the line graph Charles has the highest page rank as Charles is visited by two of the three possible paths in the entire graph.

2.3 Social Networking Measures in Software Engineering

Zimmerman et al [4] computed social network measures (some of them used in this research) for dependency graphs. They showed these measures can predict defect density on various program units. Their empirical results were based on the structure and bug reports of just one proprietary software. Concas et al [17] studied the distribution of the values of several social networking metrics for two large systems. Their research mainly consisted of identifying trends in the metrics they used, observing changes to those metrics in other versions of the same software. They also examined the correlation between the metrics and fault proneness of different sections of the program. They extended their own study in a follow-up paper [19] by including more social networking metrics and examining their correlation or anti correlation with fault proneness. Li et al [20] applied PR [6] metric used in our research, to compute the "complexity of relations" between classes in object oriented software. However, it was not clear what external software attribute was complexity representing or predicting.

2.4 Our approach compared to existing work

Our research uses established social networking metrics to compute the 'prestige' of a vertex in the graph representing program structure. Concas et al [17] has already studied the distribution of some of these metrics, and how they undergo variations with evolution of software. However, their work correlated the metrics they studied with Bugs metrics. We are yet to come across studies that have correlated prestige values and execution frequencies. We believe prestige of a class can be an indication of how often the code residing in a class gets executed at runtime. In chapter 5 we discuss the relationship between prestige of a vertex and the frequency of execution of code residing in that vertex at run time.

Chapter 3

Our Approach

In chapter 2 we discussed the ways of computing the 'prestige' of a vertex in a directed graph. The graph that we build to compute the social networking measures is the same as the compilation unit graph used by Concas et al [17] and we refer to it as the CU graph. Our research explored the possibility of using prestige of the vertices of a CU graph computed by four different social networking algorithms to predict how frequently a class would get used at runtime. The classes forming the software represent the vertices of the CU graph. The edges of the CU graph are the relations at a design level in between these classes. Given an algorithm, we generated a set of percentile rankings for each class based on its computed prestige. We conducted empirical studies to answer the following questions:

- Does a correlation exists between run time execution frequency of a class with the prestige of the class?
- Can the prestige of a class serve as a prediction of the execution frequency of that class?

Within the scope of our study the runtime execution frequency of a class is defined as the number of times any method or constructor in that class gets executed. Our study is based on testing the following hypotheses.

Hypothesis H_0:

There is no relationship between the social networking measures of a class and how frequently the code residing in a particular class gets executed at runtime.

Hypothesis H_1:

The methods that get executed most often at run time tend to be in classes that are considered to have high prestige by at least one social networking measure.

Hypothesis H_2:

The classes holding methods that do not get executed at all, tend to be in classes that are considered to have low prestige by at least one social networking measure.

Hypothesis H_3:

The classes that are considered to have high prestige by some of the social networking measures tend to have methods that are among the most frequently invoked at run time.

Hypothesis H_4:

The classes that are considered to have low prestige by some of the social networking measures tend to have methods that are rarely invoked at run time.

H_0 is the only null hypothesis. Failure to reject H_0, would render testing the other hypotheses useless. The basis for using separate hypotheses for high and low prestige classes or high and low execution frequency classes was that due to the skewed distribution of prestige values(see Chapter 4) and execution frequencies(see figure 5.1). Hypotheses H_3 and H_4 may be considered particularly important from the context of the significance of our study. Conclusions drawn from testing H_3 and H_4 could be used for predicting a run time attribute like execution frequency by static analysis.

3.1 Rationale

The 'Prestige' of a vertex in a graph computed by a social networking algorithm quantifies the notion of prominence of that vertex based on strategic locations within

the graph and direct and indirect ties. Both prestige and the likelihood of code residing in a class(say A) getting used may be affected by:

- Number of direct connections of class A.
- Transitive closure of all classes pointing at A.
- Transitive closure of all classes pointing to A.
- Likelihood of A falling on the path between two other classes.

This intuitively leads us to think that a research into the existence of a relationship between social networking measures and prestige is warranted.

3.2 Generating CU graphs

In our experiment we created the required graph by reverse engineering the CU graph from the source code. A CU graph is similar to a 'detailed' UML class diagram which includes all forms of dependencies among classes normally excluded to avoid clutter. However, unlike UML class diagrams, we do not treat inheritance, association or dependency relations as different. Our experiment has no notion of the strength of coupling based on the type of relation whether inheritance, association or dependency. We first identified the classes that form the vertices of the graph. This involved separating the program's classes from the classes belonging to the framework or library that the program uses. In order to find the edges of the graph we examined the relationships in between classes. We add an edge between vertex A and vertex B if classes represented by these vertices have either an inheritance, association or use dependency relation between them. There can only be zero or one edge in between two vertices in a given direction.

3.3 Validation Strategy

We perform dynamic analysis to verify that the values of the prestige measures are legitimate predictors of execution frequencies. The validation strategy involves running use cases of the program to capture necessary information to identify the most and least commonly used classes that are in the program. This run time information can be compared to the prestige values produced by the social networking algorithms to evaluate the hypotheses.

For the dynamic analysis, we instrumented the source code of the program to add statements that log the qualified name of the class whose method is currently under execution. We ran common use cases that the program supports on the instrumented code so that the inserted statements keep track of the methods and the class in which any executed method reside. We analyzed the log generated by running the instrumented source code to obtain the execution frequency of a class. We then generated percentile ranks for each class, based on the execution frequency of the class. We awarded a zero percentile to classes that never get executed while running the chosen set of use cases. These percentile ranks are compared with the percentile ranks predicted by the social networking algorithms using relevant visualizations and statistical methods explained in chapter 5

3.4 Architecture of INSAT

INSAT stands for **I**NSAT is **N**ot a **S**tatic **A**nalysis **T**ool. INSAT is a scalable prototype tool that we developed to conduct experiments that test our hypotheses. INSAT is an Eclipse plugin, and is capable of reverse engineering and visualizing the CU graphs of Java projects in Eclipse [25]. A high level plugin architecture for INSAT is shown in figure 3.1. INSAT uses the Java Development Tools plugin [26] to find out which classes are a part of the software rather than related libraries. INSAT uses the Byte

Code Engineering Library [27] to parse the byte code to determine the relationships among these classes. Treating the classes as vertices and relationships among the classes as edges, INSAT uses the JUNG [28] library to generate an instance of a graph, and compute the social networking measures for each vertex in the generated graph. INSAT uses R graphics [29] by making Java to R calls to generate plots that visualize distribution of values of the social networking measures across the classes forming the software.

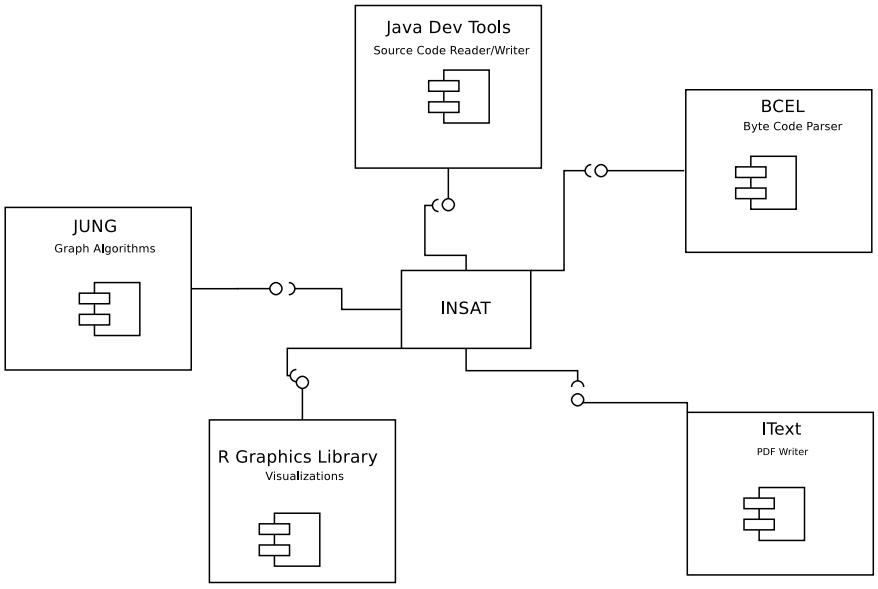

Figure 3.1: High level plugin architecture of INSAT

INSAT also has code instrumentation features that can track the extent to which code residing in a class is used while running use cases. The log files generated by INSAT during dynamic analysis contain information on frequency of method use required to validate the results obtained from static analysis. INSAT makes a separate instrumented copy of the source code of the software, by using the JDT plugin [26] to insert log statements at appropriate places, and then generates an ant script [30] to compile and run the instrumented code.

Table 3.1: Properties of systems used in this experiment

System	Size(KLOC)	Classes
JFreeChart 1.0.13	91.4	607
JHotDraw 7.5	79.9	222
Java Swing 1.4	181.7	1901
JTopas 0.8	4.38	60

3.5 Systems used in the experiment

Table 3.1 enlists the systems and their sizes that were used in this experiment. JFreeChart [33] is a Java library based on swing for creating charts. JHotDraw [31] is a library that can serve as a framework for building graphics based applications in Java. Java Swing [32] is a library shipped within Java for creating GUI based applications and is the largest system used in this experiment. JTopas [34], the smallest system used in the experiment, is a Java library for parsing text data. Our selection enables us observe the distribution of prestige values in systems of varying sizes. All four systems are frameworks that service clients, and are not standalone applications. Hence, the dynamic analysis (as explained in section 3.3) does not start at any entry point within the system, but at clients outside the system. This may reduce the bias on what all classes within the system should get executed, as the system gets used by the interfaces it exposes to clients. All these systems are well known and clients for these systems are readily available.

3.6 Experimental Procedure

The primary objective of the experiment was to study the correlation between prestige measures and execution frequencies for some of the systems listed in Table 3.1. Our experimental procedure can be summarized as follows:

1. The source code of the systems in Table 3.1 were fed into our tool Insat. Insat reverse engineered the CU graphs, and computed the prestige values of the four

metrics PR, $HITS$, BC, CBO. INSAT also visualized the distribution of these values.

2. We studied the distribution of the prestige values for each of the systems and identified trends in them based on their visualizations.

3. We performed dynamic analyses on JHotDraw and JFreeChart using a set of common use cases. This was done on instrumented versions of the two systems, so that we could obtain the values of execution frequencies for each class in the systems.

4. We used relevant statistical methods to study the correlation between the prestige values and the execution frequencies.

Chapter 4

Demonstration of Prestige Ranking

In this chapter we study the distribution of the prestige measures of JFreeChart, JHotDraw, Java Swing, and JTopas. For each of the systems we used our tool INSAT to build the CU graph and computed the values of all the social networking metrics discussed in Chapter 2. In addition to these metrics we also measure the *LOC*(lines of code) metric for each class that is a vertex in the CU graph. The social networking measures would be compared to the LOC metric in our empirical validation procedure.

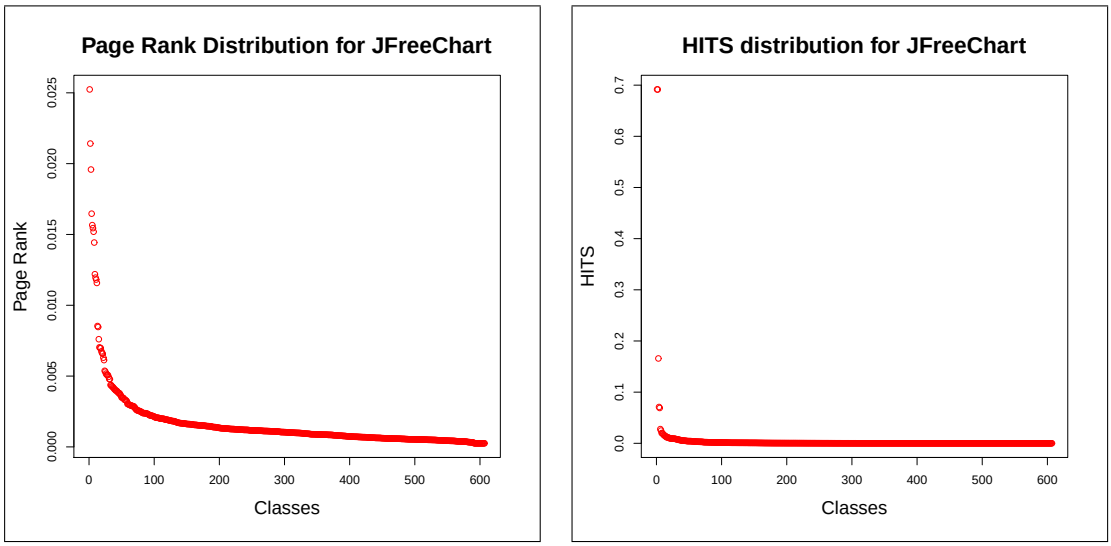

Figure 4.1: Page Rank(PR) Distribution(left) and HITS Rank Distribution(right) for *JFreeChart*

We graphically display visualized distribution of the values of the prestige measures in several plots included in this chapter. Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, show

the trends that these measures follow in the Systems studied in our experiment. The classes on the y-axis of these plots are hence, in descending order of their values for the prestige measures. We thought visualizing the sorted values would help identify the trends these measures may follow.

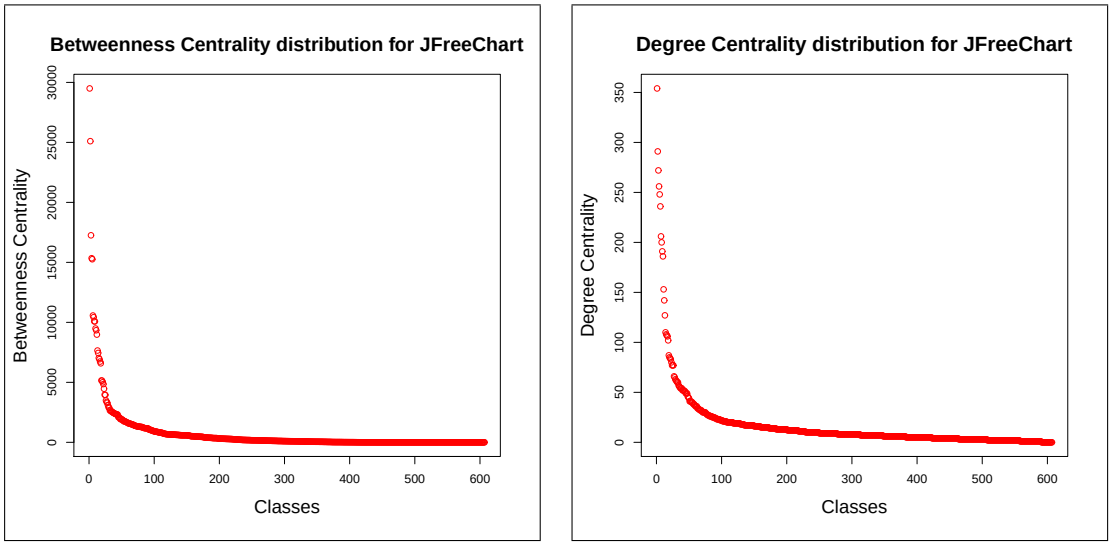

Figure 4.2: Degree Centrality(CBO) distribution(left) and Betweenness Centrality(BC) distribution(right) for *JFreeChart*

4.1 Observations

The distribution of values for all prestige measures in all the systems studied show a power law behavior. The power law behavior underlies the Pareto principle where a large number of the classes have very low values and few of the classes have very high values. The difference in the values of the classes that are more towards the tail are low. Consequently, the number of tied values for the metrics significantly increase towards the tail of the curves.

The power law behavior appears to become more conspicuous with increases in the size of the system in terms of the number of classes. The strongest power law behavior for metrics *PR, BC, CBO* is observed in the Java Swing Library, whereas it is perhaps the least strong for the same measures in JHotDraw which is the smallest

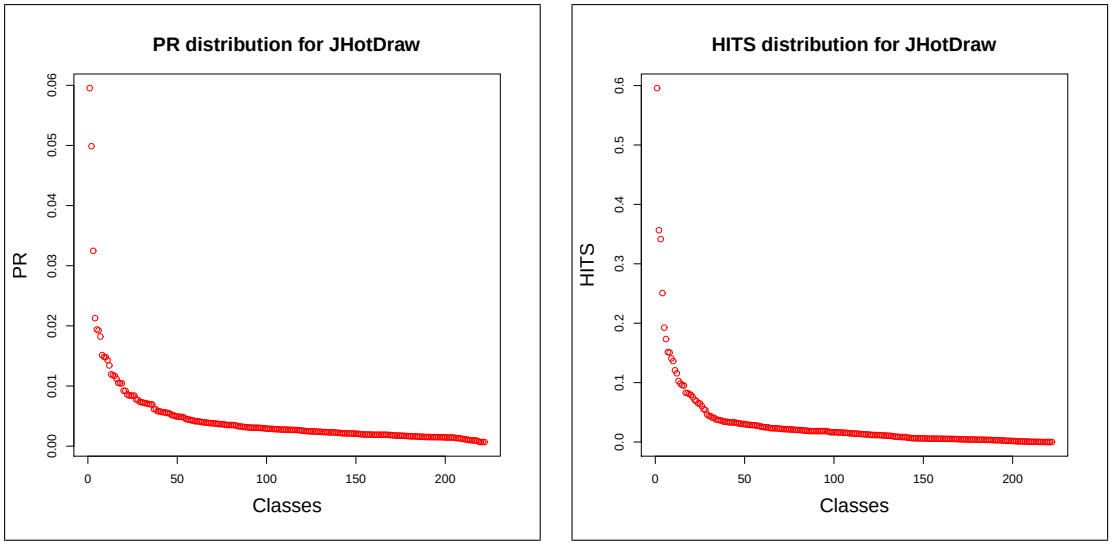

Figure 4.3: PageRank(PR) distribution(left) and HITS distribution(right) for *JHotDraw*

system studied. The tendency towards a power law became even weaker for a smaller system like JTopas [34](Figures 4.7, 4.8) which consists of just 60 classes.

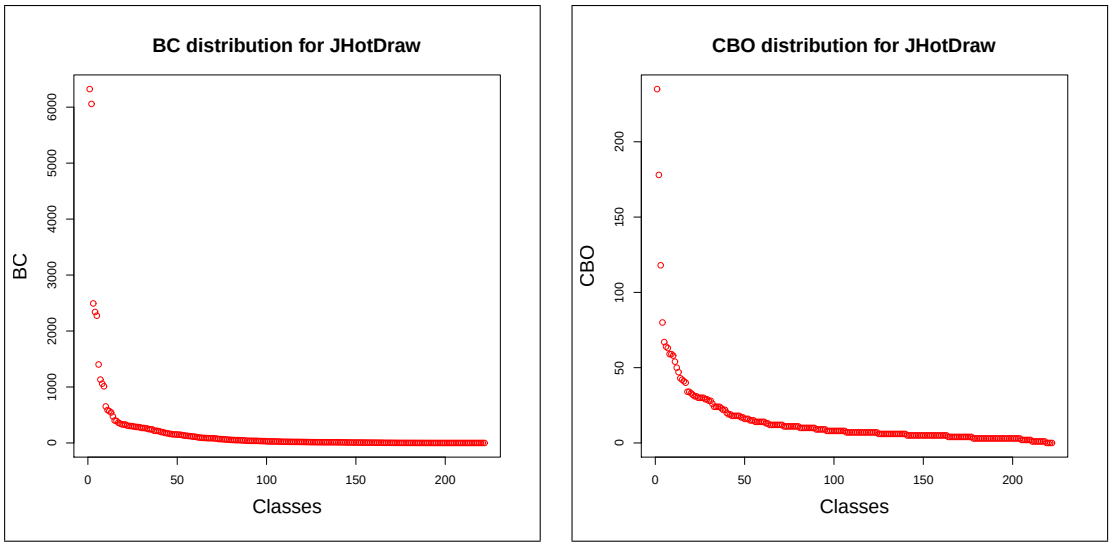

Figure 4.4: Betweenness Centrality(BC) distribution(left) and CBO(Degree Centrality) distribution(right) for *JHotDraw*

The distribution for the *HITS* metric as evident from the right-side plots of figures 4.1, and 4.5 appears to be the most skewed particularly in JFreeChart, and Swing. The *HITS* metric in Swing shows are very long tail with perhaps a high

number of tied scores. The *BC* metric appears to have the highest number of tied values in towards the tail for both JHotDraw and JFreeChart. An observation of the CU graph reveals that a fair number of classes do not fall in the path between any two other classes, thus making the value of the *BC* metric zero.

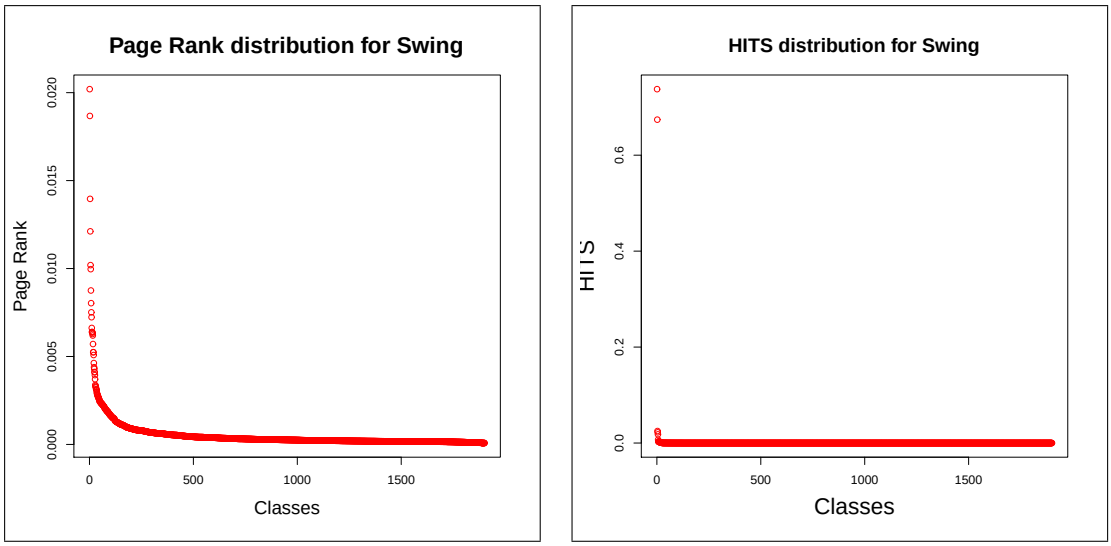

Figure 4.5: Page Rank Distribution(left) and HITS Rank Distribution(right) for *Java Swing*

The highest number of tied values overall(both towards the tail and the peak), is found in the *CBO* metric. The *CBO* metric is always an integer and its values can only be within a certain range in between zero and the total number of compilation units. As a consequence of the number of ties, the values of the *CBO* metric would have the least number of distinct ranks that can be assigned to the distribution. The JHotDraw system could be assigned only 51 distinct ranks to its *CBO* values.

4.2 Interpretations

Concas et al [17], observed similar power law behavior in his study of distribution of social networking measures. A study on power distributions in software systems [18] reported that several internal system attributes show a power law trend. The study reported power law distributions in class size(in LOC or number of methods), method

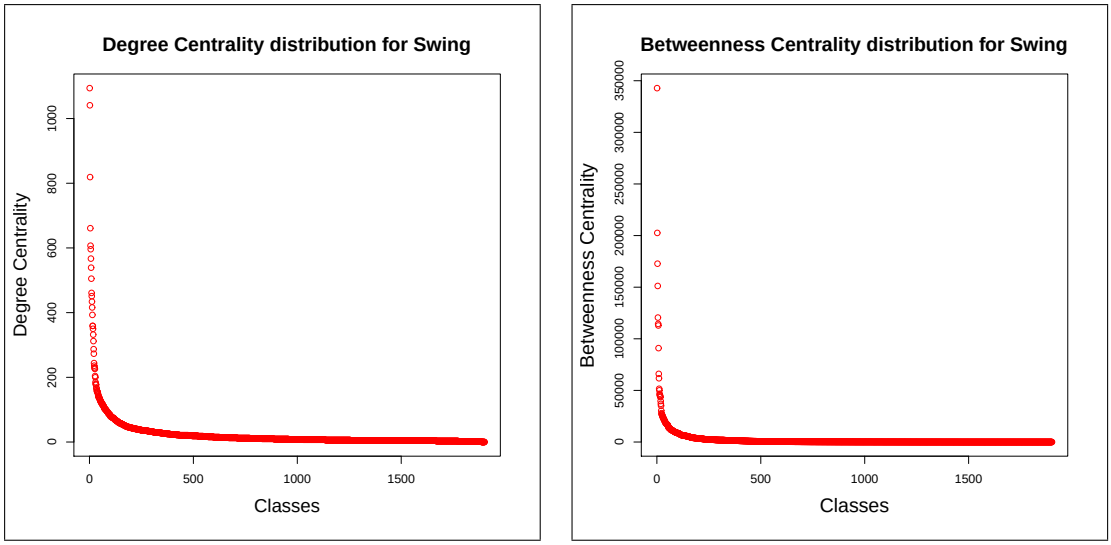

Figure 4.6: Degree Centrality distribution(left) and Betweenness Centrality distribution(right) for *Java Swing*

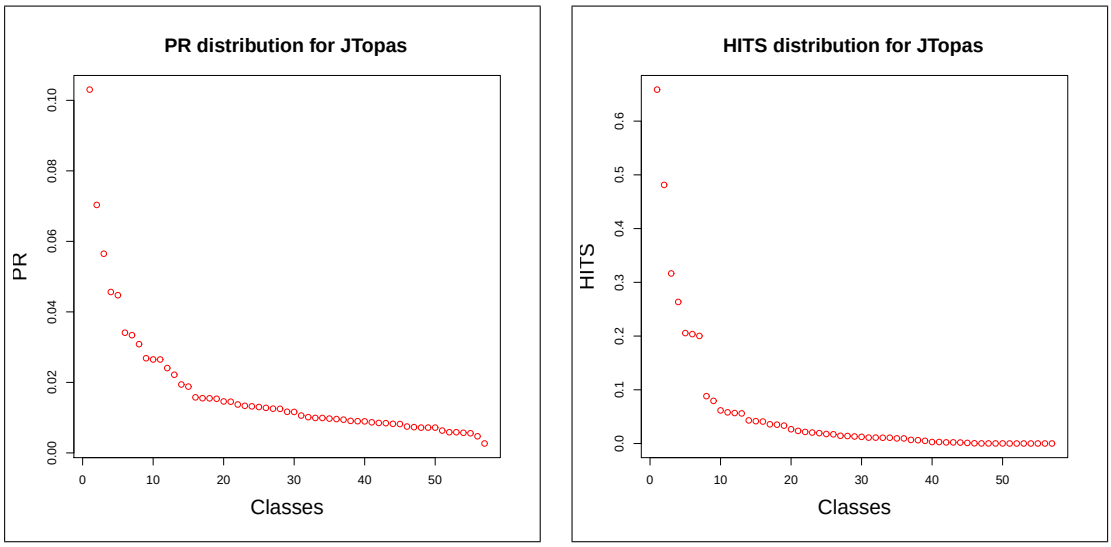

Figure 4.7: Page Rank Distribution(left) and HITS Rank Distribution(right) for *JTopas*

size, length of inheritance hierarchies, frequency of method or variable names, number of calls to specific method names, number of instance variables per class etc. The values of the social networking measures used in our research depend on some of these system attributes. Intuitively social networking metrics may follow a power law based distribution too. Moreover, we found similar power law based distributions in actual values of the run time execution frequency shown in figure 5.1.

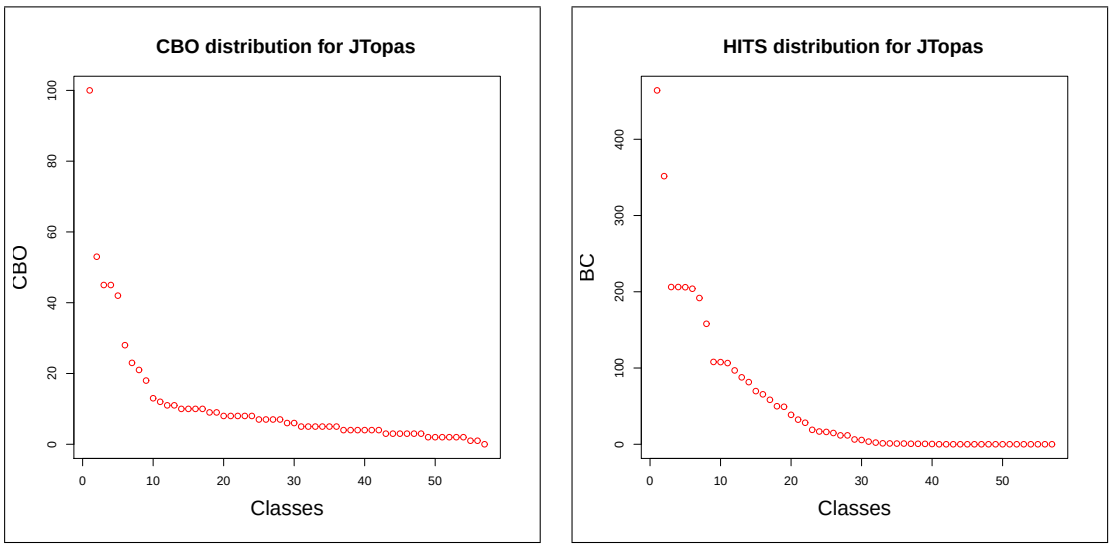

Figure 4.8: Degree Centrality distribution(left) and Betweenness Centrality distribution(right) for *JTopas*

Chapter 5

Evaluation of Hypotheses

Our objective is to evaluate the hypothesis concerning the relationship between the values of the social networking measures and the execution frequencies of classes. We needed the values of execution frequency of the classes in the systems from Table 3.1 to test the hypotheses. The distribution of the run time execution frequencies of the classes when the system is in operation is expected to vary based on the use cases that are executed. We obtained a 'probable' distribution of execution frequencies by building an operational profile that consists of the most common use cases the system is expected to support. For example, we built the operational profile for the JFreeChart framework by executing clients that create the different forms of plots that JFreeChart supports. Figure 5.1 shows the distribution of run time execution frequencies for JFreeChart and JHotDraw. The execution frequency distribution shows a power law behavior similar to trends observed for the social networking measures.

In our validation strategy, we tested the hypotheses outlined in chapter 3 by comparing the values of the social networking measures to existing static code metrics that may be related to frequency of code usage. These static measures included direct coupling (measured by the CBO metric) and the LOC metric. The CBO metric is itself analogous to the social networking metric Degree Centrality. We show that the CBO metric in conjunction with the other metrics are better predictors than just

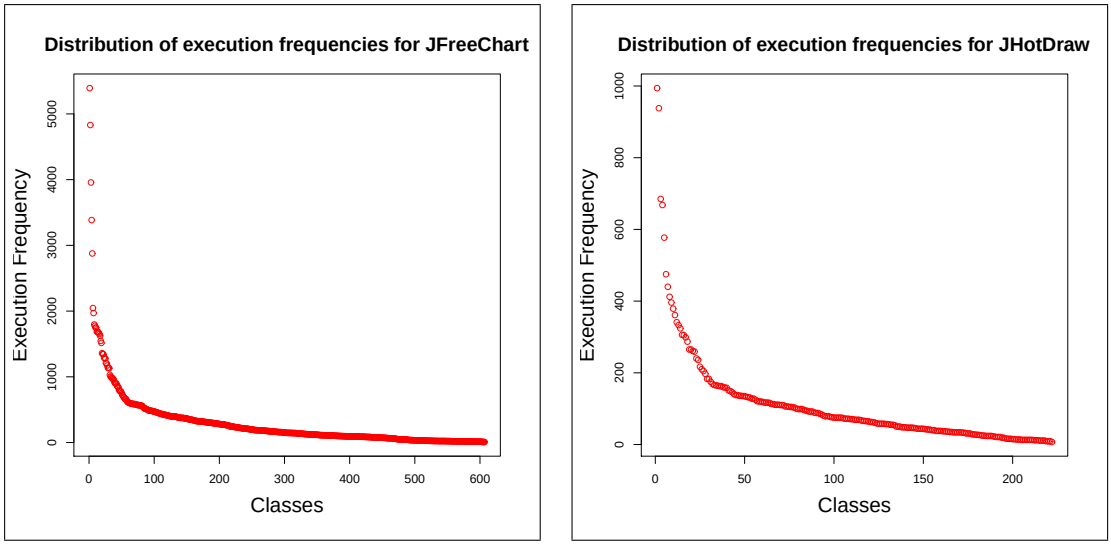

Figure 5.1: Distributions of execution frequency for classes in *JFreeChart*(left) and *JHotDraw*(right)

the CBO metric or the LOC metric. We define the LOC metric of a class as the number of lines of code in a class minus the comments and blank lines. If we consider a scenario where all methods in the system are equally likely to get executed, then a class with higher LOC will have a greater probability of having a higher execution frequency than a class with lower LOC. This may justify the LOC metric being used as a standard to compare our results.

In order to test hypotheses H_1, H_2, H_3, and H_4 we converted the values of the measures to their corresponding rank percentiles. This was done because we are more interested in relative values of prestige rather than the absolute value. Transforming the values to rank percentiles allowed us to use the Wilcoxon test to evaluate statistical significance. The hypotheses tests required us to quantify the notion of 'high' or 'low' for values of execution frequencies and the social networking measures. We chose the top 10% or the bottom 10% (except for testing H_2) to denote a high or low for values of both execution frequencies and social networking measures respectively. The basis behind selecting 10% was to essentially make a trade off between covering as many classes in the system, and maintaining the significance of the statistical tests. The

Table 5.1: Spearman correlations and their significances of all measures with execution frequency for JFreeChart

Measure	**Spearman Corr**	**p-value**
PR	0.455	$2.19 \times 10^{-10} < .05$
HITS	0.302	$4.45 \times 10^{-5} < .05$
BC	0.462	$1.06 \times 10^{-10} < .05$
CBO	0.385	$1.31 \times 10^{-7} < .05$
LOC	0.358	$1.00 \times 10^{-06} < .05$

Table 5.2: Spearman correlations and their significances of all measures with execution frequency for JHotDraw

Measure	**Spearman Corr**	**p-value**
PR	0.463	$2.90 \times 10^{-8} < .05$
HITS	0.325	$1.54 \times 10^{-5} < .05$
BC	0.522	$1.76 \times 10^{-10} < .05$
CBO	0.305	$4.01 \times 10^{-5} < .05$
LOC	0.451	$6.99 \times 10^{-8} < .05$

top 10% was also a safe choice given the skewed distributions of the social networking measures as shown in Chapter 4, as the prestige values start to fall abruptly after that. Detailed results of our tests are given in the following sections. We have used a confidence level of $\alpha = 0.05$ in all of our hypothesis tests.

5.1 Test H_0

We tested hypothesis H_0 to analyze if there was any relationship between the social networking measures and run time execution frequencies. We computed Spearman's correlation coefficient between all four of our social networking measures and LOC, and we further tested if the correlation was significant. The Spearman's correlation coefficient was an obvious choice as all measures apart from LOC, showed a skewed distribution. Spearman's ρ statistic was used to test the significance of the correlations.

Tables 5.1, and 5.2 shows the values of Spearman's correlations and how significant

they are. The BC measure seems to have a strongest correlation, whereas the HR measure had the weakest. Our results showed significant correlation of all measures with run time execution frequency of a class. Hence, we can reject null hypothesis H_0 at the 0.05 level.

5.2 Test H_1

We tested hypothesis H_1 that states that classes with high execution frequencies tend to be considered prestigious by at least one social networking measure. For the purpose of testing our hypothesis we define a class has a 'high execution frequency' if it features in the top 10% of the classes ranked by execution frequency. Figure 5.2 shows a box plot that visualizes the distribution of the rank percentiles of the measures CBO, LOC, and MAX of the classes with high execution frequency. We define the MAX metric as the numerically highest value of the rank percentiles allotted by the four social networking measures PR, $HITS$, BC, and CBO for a class with execution frequency. The MAX metric takes care of the condition that the class needs to be considered prestigious by at least one of the measures.

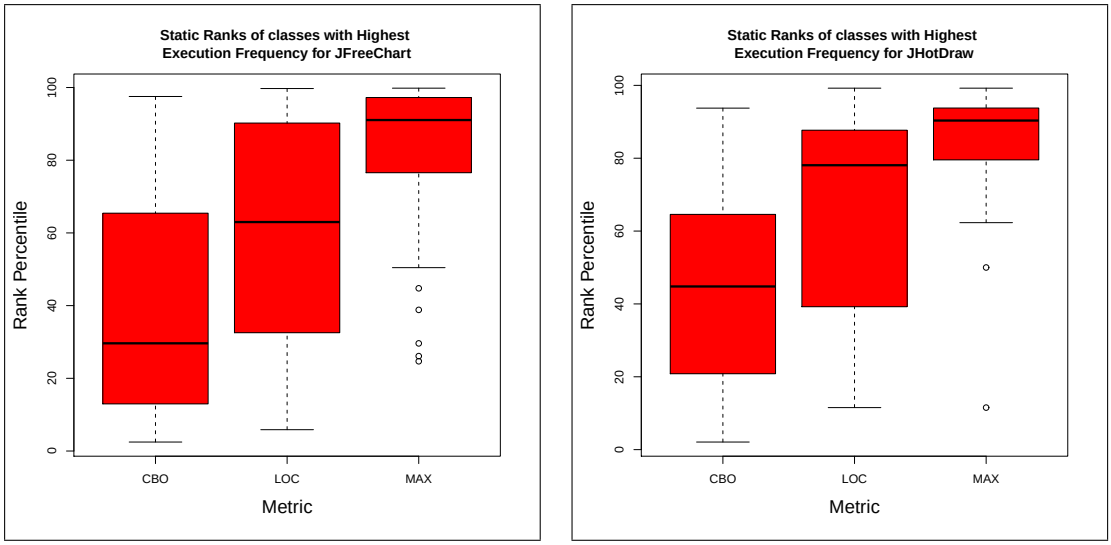

Figure 5.2: Distribution of Rank percentiles obtained by static analysis of the top 10% for JFreeChart(left) and JHotDraw(right)

Table 5.3: p-values of Wilcoxon test for max rank percentiles against CBO or LOC for JFreeChart and JHotDraw

	CBO	LOC
$MAX_{JFreeChart}$	4.17×10^{-12}	1.22×10^{-5}
$MAX_{JHotDraw}$	1.261×10^{-5}	0.02649

From the box plot in Figure 5.2, it appears that for both the systems the rank percentiles for MAX has a narrower concentration, and numerically higher values than the CBO or the LOC metric. Table 5.3 shows the p-values of a Wilcoxon test performed on the MAX metric against LOC and CBO. The p-values are essentially probabilities that LOC or CBO would allot a higher rank percentile than MAX, given a class with a high execution frequency. The p-values are significant at the 0.05 level for both the systems

5.3 Test H_2

Hypothesis H_2 concerns a possible correlation between low prestige and low execution frequency. In our experiment, we regard a zero execution frequency as low. Classes that do not get executed at all at run time are the ones with zero execution frequency. We conjecture that classes with low execution frequency will be mostly classes with low prestige. The box plot in Figure 5.3 shows the distribution of the rank percentiles of the measures CBO, LOC, and MIN of the classes with zero execution frequency. The MIN metric is defined as the numerically lowest value of the rank percentiles allotted by the four social networking measures PR, $HITS$, BC, and CBO to a class with low execution frequency. The MIN metric takes into account our hypothesis that the class should be considered low in prestige by at least one of the measures.

With our data, the CBO metric does a better job in predicting low prestige for unused classes than all other measures apart from MIN. The probability of CBO or LOC giving a rank lower than MIN is expressed as p-values of the Wilcoxon test in Table 5.4. The p-values makes us confident about our conclusion that classes with

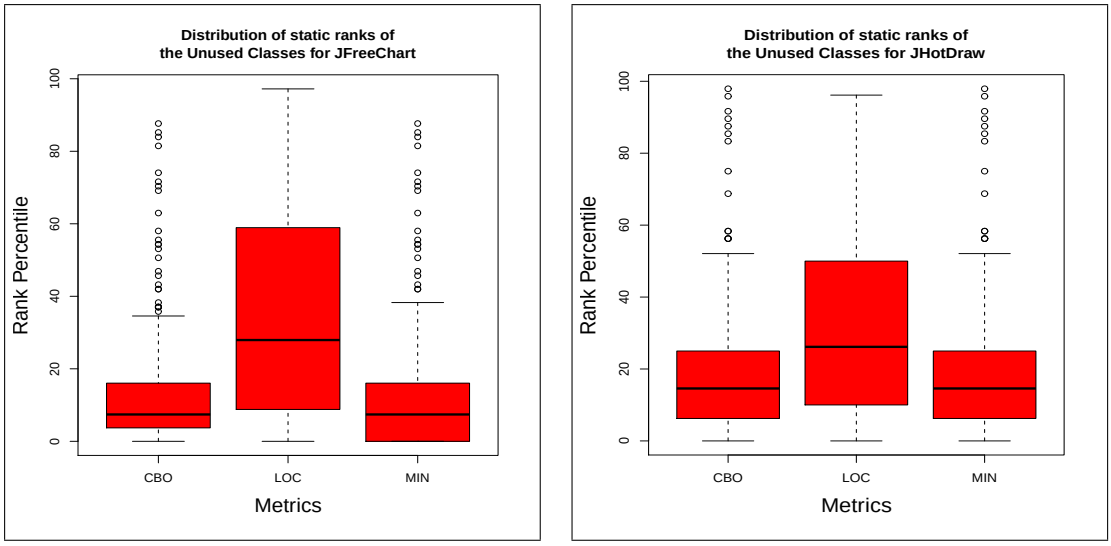

Figure 5.3: Distribution of static rank percentiles for unused classes (zero execution frequency) for JFreeChart(left), and JHotDraw(right)

Table 5.4: probability for MIN rank percentiles to be greater than CBO or LOC for JFreeChart and JHotDraw

	CBO	LOC
$MIN_{JFreeChart}$	0.0422	7.23×10^{-40}
$MIN_{JHotDraw}$	0.1037	1.99×10^{-187}

low execution frequency tend to have a lower value of prestige awarded by the MIN measure than the LOC measure. However, the p-value for JHotDraw does not give us the confidence at the 0.05 level that execution frequency relates better to MIN than CBO.

5.4 Test H_3

Hypothesis H_3 is directly related to the usefulness of social networking metrics as predictors of execution frequency. If we can show that prestigious classes tend to get executed frequently, then this can help estimate the run time usage of a class without actually performing any dynamic analysis. First we identified the elements of a set we consider to have "high prestige by the greatest number of measures". In the following listing we describe the steps to construct the MAX set. C and MAX were initially

empty sets and n is integer equal to 10% of the total number of classes in the system under test.

1. We formed a collection C that consisted of classes that feature in the top 10% of the rank list of the four social networking measures. It is possible that a class can feature in the top 10% for multiple rank lists, so C has several duplicates.

2. We go on removing classes that have the greatest number of duplicates from C and put them in set MAX. We started with classes that have four instances in C, and went on adding for three and two instances. This process continued when either of the following conditions are satisfied:

   - The cardinality of MAX was n which means we have a set MAX with the required number of elements.
   - There are no classes with multiple instances left in collection C.

3. If $n(MAX) < n$ the next step is to prune the set C(C is a set now has it does not have any duplicates) so that sum of the cardinalities of C and MAX is n. This was achieved in the following steps:

   (a) Go on removing classes in C randomly with a bias towards higher percentile rank of prestige till the condition $n(MAX) + n(C) = n$ is satisfied.

   (b) $MAX = MAX \cup C$. MAX is now a set of classes considered prestigious of greatest number of SN measures and has the required cardinality.

A pseudo code for the algorithm to build the set MAX is given below:

map = empty hash map
n = 10% of total number of classes
for rank list R in all SN metrics **do**
  for c in top 10% of R **do**

if map has c **then**

$frequency =$ value of c in map

$frequency \leftarrow frequency + 1$

put $(c, frequency)$ in map

else

put $(c, 1)$ in map

end if

end for

end for

$MAX = \phi$

$B = \phi$

for all keys k in map **do**

$frequency =$ value of k in map

if $frequency \geq 2$ **then**

add k to set MAX

else

add k to set B

end if

end for

if $n(A) < n$ **then**

$x \leftarrow n - n(MAX)$

while $n(B) > x$ **do**

$y \leftarrow$ random element ϵ B with bias towards lower percentile

remove y from set B

$MAX = MAX \cup \{y\}$

end while

end if

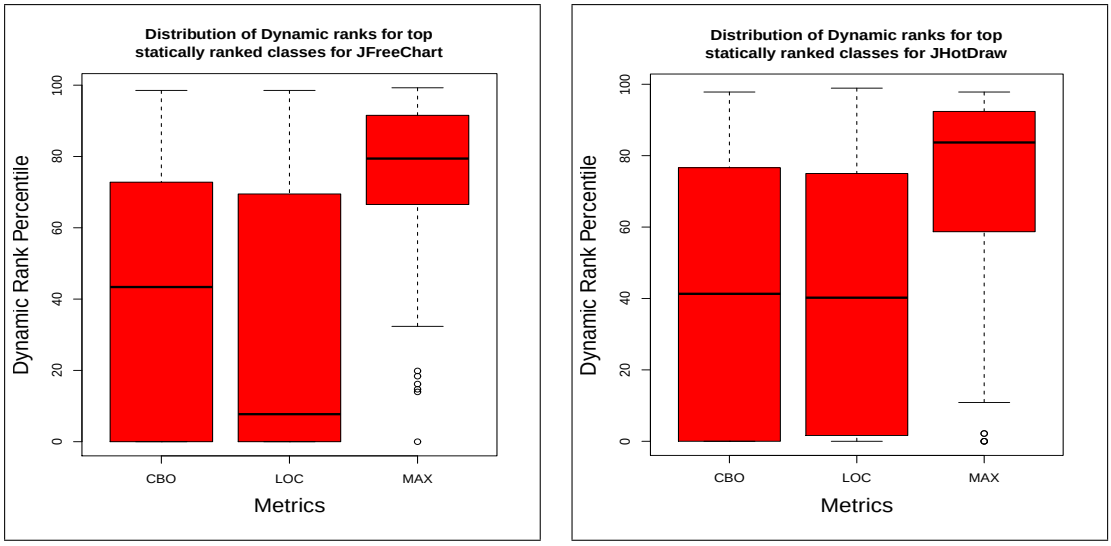

Figure 5.4: Distribution of Dynamic rank percentiles for statically top ranked classes for JFreeChart(left), and JHotDraw(right)

Table 5.5: probability for max rank percentiles to be greater than CBO or LOC for JFreeChart

	CBO	**LOC**
$MAX_{JFreeChart}$	4.83×10^{-7}	7.68×10^{-9}
$MAX_{JHotDraw}$	5.05×10^{-6}	1.57×10^{-5}

The box plots in figure 5.4 visualizes the distribution of execution frequencies (scaled to percentile ranks) of the MAX set compared to execution frequencies of classes that feature in the top 10% of the CBO and LOC metric. The box for MAX shows a much narrower and numerically higher concentration of values for MAX. The p-values from table 5.5 can be interpreted as the probability of getting a class considered among most prestigious by either LOC or CBO, having a execution frequency greater than a class considered among the most prestigious by the MAX metric. All p-values are significant at the 0.05 level which indicates that MAX is a better predictor of execution frequency than CBO or LOC.

5.5 Test H_4

H_4 posits a correlation between low prestige and low execution frequency. We examined the distribution of execution frequency (scaled to rank percentiles) of a set of classes with low relative prestige. We constructed the set of low prestige classes using a similar procedure outlined in section 5.4 for constructing a high prestige set. Instead of starting with the top 10% we selected the bottom 10% for each measure. We changed step 3(a) by randomly removing elements with a bias towards relatively higher percentile ranks. The results of the Wilcoxon test is given in table 5.6

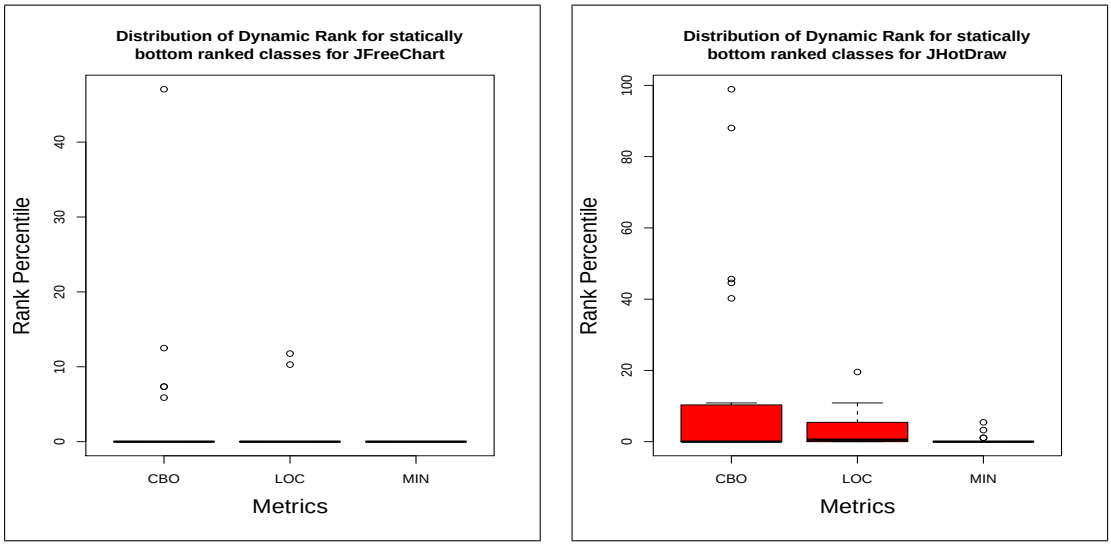

Figure 5.5: Distribution of Dynamic rank percentiles for statically top ranked classes for JFreeChart(left), and JHotDraw(right)

The MIN metric in the box plot in Figure 5.5, shows the distribution of execution frequency rank percentiles of classes that we consider low prestige. Nearly all classes in MIN for both the systems were ranked with zero percentile(that is the classes never got used). However, the difference in the predictions of CBO and LOC from MIN does not look significant, especially in case of JHotDraw. The p-values in table 5.6 can be interpreted as the probability of a class in the CBO or LOC metric set having a execution frequency lower than that of a class belonging to the MIN

Table 5.6: probability for max rank percentiles to be greater than CBO or LOC for JFreeChart

	CBO	**LOC**
$MIN_{JFreeChart}$	0.023	0.16
$MIN_{JHotDraw}$	0.0082	0.033

set. The p-value for JFreeChart that compares MIN and LOC is not significant at the 0.05 level. However, all other p-values are significant which indicates MIN is a better predictor of low prestige than CBO for both systems, and the LOC metric for JHotDraw.

Chapter 6

Threats to Validity

Threats to validity exist for all empirical studies. We did make assumptions and could not control all factors that can affect our results and inferences. In this chapter we describe the threats to validity and how we address them.

6.1 Construct Validity

Within the context of our study, threats to construct validity would question whether measures of prestige conceptually relates to the relative frequency of code usage at run time. In a more general sense, the threat to construct validity will doubt the possibility of dynamic behavior being predicted entirely by static analysis. Briand et al [7] have reported the decay of quality models that are based on static analysis, with dynamic binding, and inheritance. The presence of 'dead code' often decays program structure and renders coupling measurements made by static analysis imprecise [10]. To alleviate this threat, we performed relevant statistical analyses to test for significance of correlations between the prestige values and execution frequencies. Another threat to construct validity is the fact that we did not distinguish between different kinds of relationships like inheritance, association, or dependencies even though these relations at a static level are associated with different notions of strength.

6.2 Internal Validity

In the scope of our study, threats to internal validity may include factors we missed or could not control, that can affect a casual relationship between prestige and execution frequency. Closeness centrality is a social networking measure that we did not include in our study. Inclusion of other social networking measures to build a prediction model for execution frequencies is left as a possible extension to our study(refer section 7.1). The CU graph for each system included only the relations that were visible at the design level of the system. There is a possibility that other underlying relations may exist, like two classes coupled via a third class which exists in some other library outside the system. These type of relations can change the CU graph, and eventually affect the computations of prestige. We have ignored these relations in our experiment.

6.3 External Validity

Threats to external validity questions whether the findings of our study can be generalized for other software. We observed a power law behavior in the distribution of the prestige measures on all five libraries that we have tested so far. The trends that we observed in these distributions are similar to those observed by Concas et al [17]. Running our tool on the libraries with their clients included, gave us the same type of distributions. We feel that the trend these distributions follow can be generalized for other systems. However, we have studied and successfully validated the correlation between prestige and execution frequency for just two libraries JFreeChart and JHotDraw. It is not guaranteed whether this correlation can be generalized for all types of software. The same experiment could be done with softwares of different sizes to determine if this correlation exists.

6.4 Conclusion Validity

Conclusion validity refers to the legitimacy of the data and the statistical methods used in our experiment to arrive at conclusions. Perhaps the biggest threat to validity of our research is related to the selection of use cases in our empirical validation procedure. We created a run time profile for the systems by executing a set of 'obvious' use cases that these systems were built to support. This threat questions to what extent the created run time profile represents the intended usage of the system. We selected a set of use cases that the programs were originally designed to support. The clients that implemented these use cases were partly downloaded from the product's website and partly written by us. As the distributions of prestige followed the power law, they were not normally distributed. In order to alleviate threats to conclusion validity concerning inappropriate statistical methods, we used statistical methods and tests like Spearman correlations, and Wilcoxon's signed rank test that are relevant to non-parametric data.

Chapter 7

Conclusions and Future Work

In our research, we applied social networking algorithms to static program structures. We created CU graphs, where vertices were formed by the classes in the software, and edges were formed by design level relations in between the classes. We analyzed these graphs by computing the prestige for each vertex using four social networking measures PR, $HITS$, BC, and CBO. For each prestige measure, we visualized and studied the trends that the distribution of these measures follow. We found that values for all of these measures followed a skewed distribution and showed tendency towards the power law. From the visualizations, power law characteristics like for a given measure few of the classes having very high values and all other classes having low values were observed. We found a greater number of tied values towards the tail of the distributions.

We demonstrated the importance of prestige computed by the social networking measures by studying the relationship between the prestige of a class with the run time execution frequency of that class. The study was performed on two systems where we performed dynamic analysis on each system by running common use cases that the systems were built to support. The results of the statistical analysis suggested that prestige and execution frequency of some classes were correlated. Most of the classes with high execution frequency were given a high prestige by at least one measure. Most of the classes that had high prestige awarded by some measures ended up with

a high execution frequency. Classes with low execution frequency were given low prestige by at least one measure. We also showed that prestige of a class can serve as a better predictor of execution frequency of that class than CBO [13] or lines of code, for classes that tend to have very high or very low execution frequencies.

We developed an eclipse plugin that we call INSAT to automate our experiment. The source code of systems can be given as input to INSAT which is then reverse engineered to a CU graph, and computations for prestige values are made. INSAT is also equipped to instrument code to gather run time information like execution frequency of a class.

7.1 Future Work

Future work may include replicating the study in other systems of different sizes to determine if prestige and execution frequency still correlate. We observed power law behavior in the distribution of prestige that is seems more conspicuous with increase in size of the software. This observation may be further explored in future research. There exists other social networking measures like closeness centrality whose distribution of prestige and relation to execution frequency may be studied. We used MAX and MIN metrics in our empirical validation in Chapter 5 which can perhaps be improved by learning a model that predicts execution frequency from the prestige computed by the social networking measures. We could then use the same model for classes with prestige that are in the peak or the tail of the curve, instead of using MAX and MIN. Further investigation can be done to determine whether other measures of testability or coupling can better predict execution frequency or if they are strongly correlated to the social networking measures that we used. Since we now know that the prestige values can help estimate execution frequencies, these prestige values could aid the research in automated generation of test inputs.

REFERENCES

[1] S Wasserman, K Faust. "Social Network Analysis". Cambridge University Press, 1994.

[2] S Narayanan. "The Betweenness Centrality of Biological Networks", MS Thesis. Virginia Polytechnic Institute and State University, 2005.

[3] J M Kleinberg. "Authoritative Sources in a Hyperlinked Environment" in the *ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[4] T Zimmerman, N Nagappan. "Predicting Defects using Network Analysis on Dependency Graphs". *International Conference on Software Engineering*, May 2008.

[5] Y Lin, A Milanova. "Static Analysis for Dynamic Coupling Measures". *The Center for Advanced Studies Conference*, 2006.

[6] L Page, S Brin, R Motwani, T Winograd. "The PageRank Citation Ranking: Bringing Order to the Web". *Stanford InfoLab*, Nov 1999.

[7] L. C Briand, J Wuest "Empirical Studies of Quality Models in Object-Oriented Systems". *Advances in Computers, Vol 59*, 2002.

[8] E Arisholm, L. C Briand, A Føyen. "A Unified Framework for Coupling Measurement in Object-Oriented Systems". *IEEE Transactions on Software Engineering, Vol 30, No 8*, Aug 2004.

[9] P Ammann, J Offut. "Introduction to Software Testing", Cambridge University Press, 2008.

[10] L. C Briand, J. W Daley, J Wüst. "Dynamic Coupling Measurement for Object-Oriented Software". *IEEE Transactions on Software Engineering, Vol 30, No 8*, Aug 2004.

[11] L.C. Briand, J. Wüst, H. Lounis, "Using Coupling Measurement for Impact Analysis in Object-Oriented Systems," *International Conference of Software Maintenance*, 1999.

[12] A Govan, C Meyer. *Ranking Methods* `http://meyer.math.ncsu.edu/Meyer/Talks/RankingMethods.pdf`, May, 2011.

[13] S.R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented design", in *IEEE Transactions on Software Engineering*, Vol 20, June 1994.

[14] F.G. Wilkie, B.A. Kitchenham. "Coupling measures and change ripples in C++ application software", in *The Journal of Systems and Software*, 2000.

[15] L.C. Freeman. "A Set of Centrality Based on Betweenness", *Sociometry*, 1977.

[16] A.S. Klovdahl, E.A. Gravissb, A. Yaganehdoostc, M.W. Rossc, A. Wangerd, G.J. Adamse, J M. Musser. "Networks and tuberculosis: an undetected community outbreak involving public places", in *Social Science and Medicine*, 2001.

[17] G Concas, M Marchesi, A Murgia, R Tonelli. "An Empirical Study of Social Network Metrics in Object-Oriented Software", *Advances in Software Engineering*, Volume 2010.

[18] G Concas, M Marchesi, S Pinna, N Serra. "Power-laws in a large object-oriented software system", *IEEE Transactions on Software Engineering, Vol 33*, 2007.

[19] R Tonelli, G Concas, M Marchesi, A Murgia. "An Analysis of SNA Metrics on the Java Qualitas Corpus", *India Software Engineering Conference*, Feb 2011.

[20] F Li, T Yi. "Apply PageRank Algorithm to Measuring Relationship's Complexity" in *IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, 2008

[21] M. J Harrold, G Rothermel, R Wu, L Yi. "An Empirical Investigation of Program Spectra", *Workshop on Program Analysis for Software Tools and Engineering*, 1998.

[22] M Renieris, S. P Reiss. "Fault Localization With Nearest Neighbor Queries", in *IEEE International Conference on Automated Software Engineering*, 2003.

[23] N Fenton. "Software Measurement: A Necessary Scientific Basis", *IEEE Transactions on Software Engineering, Vol 20*, 1994.

[24] R. Martin, "OO Design Quality MetricsAn Analysis of Dependencies," *Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*, OOPSLA. 1994.

[25] The Eclipse Project. `http://www.eclipse.org`, May, 2011.

[26] Eclipse Java Development Tools. `http://www.eclipse.org/jdt/`, May, 2011.

[27] BCEL Library. `http://jakarta.apache.org/bcel/`, May, 2011.

[28] Java Universal Network/Graph Framework. http://jung.sourceforge.net/, May, 2011.

[29] The R Project For Statistical Computing. http://www.r-project.org/, May, 2011.

[30] Apache Ant. http://ant.apache.org/, May, 2011.

[31] JHotDraw - Java GUI framework for technical and structured Graphics. http://www.jhotdraw.org/, May, 2011.

[32] Java Foundation Classes(JFC). http://java.sun.com/products/jfc/download.html, May, 2011.

[33] JFreeChart - Java Chart Library. http://www.jfree.org/jfreechart/, May, 2011.

[34] JTopas - Java Tokenizer and Parser tools. http://jtopas.sourceforge.net/jtopas/index.html, May, 2011.

[35] Code Cover - An open-source glass-box testing tool. http://www.codecover.org, May 2011.