“Power Laws” in Software Architecture and “OO Programming”

Avani Gade
University of Windsor
gadea@uwindsor.ca

ABSTRACT

A single statistical framework, comprising power law distributions and scale-free networks, seems to fit a wide variety of phenomena. “Power-laws” is a word that has become significant in recent years. Software systems now are large, complex, and ubiquitous, however little is known about the internal structures of practical software system. The software of a Software system must be both functional and evolvable, and unlike biological systems it is to a large extent designed in advance. But “Is it good?" is the real question. There is evidence that Power laws appear in Software at the class and function level. This survey describes research showing the importance of Power laws in Software Architecture as the Software systems play an important role in this context, as they are characterized by a high level of complexity.

Keywords: **Software systems, Power Laws, Small Talk Graphs.**

CONTENTS

1. INTRODUCTION

The survey is about research that is concerned with Power Laws that appear in software at the class and function level and investigations of some relationships that follow power-laws, while others do not. The sources used in this survey are Google scholar, ACM digital Library, IEEE Explorer etc.

After deciding of the topic for the survey, various synonyms for the keywords in the topic where used to search in Google Scholar. This approach helped to identify only relevant papers. Advanced Scholar Search, also helped in concentrating only in the articles published in the computer science. The University of Windsor Leddy library's web portal access to various digital libraries such as IEEE Xplore, ACM, Springer link, etc. further helped in accumulating appropriate papers. The papers accumulated from the above mentioned sources, were the first round of the paper collection. The next and more refined search for the papers was made possible by reading through collected papers and recognizing relevant research work that the authors of the papers referred. The whole process resulted in 20 papers among which 10 were designated as "most important papers" amid which 6 are the journal papers and 4 are books.

The first paper used in the survey research was written in 2003. There were 8 additional papers written in the 2000s including the last paper that was written in 2009.

Some authors have already found significant Power-laws in Software systems. Valverde et al. [2003] studied the emergence of scaling in Software Architecture graphs belonging to large Java and C/C++ open source software systems. The graphs they studied had classes as nodes and had interclass relationships as links. They found significant power-laws in the graph vertices input and output edge distribution in all the software systems they analyzed. Marchesi et al. [2004] measured the degrees of component graphs and CK metrics of Smalltalk programs and Java programs. They observed the power-law or the lognormal distributions as their results. Potanin et al. [2005] in coalition with Valverde et al. [2003] studied the graphs formed by various runtime object-oriented programs, where objects are nodes and references are links. They analyzed many different systems, written in Java, C++, and Smalltalk languages, showing them to be scale-free networks without exception.

The next sets of papers explore Software metrics in a different perspective. Jing et al [2006] was the first paper to investigate Object Oriented software metrics. Baxter et al. [2006] found the power-law or the lognormal distributions in the class relationships in Java programs. The author states that the degree distribution of the component graphs is constructed based on statical analysis of source files are explored. However, Ichii et al. [2008] states that their work was similar to Baxter et al. [2006] but from a different perspective as results in their experiments may differ from ones of the related works since their definition of component graph and software metrics is different from any of the related works.

The next group presents papers dealing with coalitions formed among multiple partners .Yule et al. [2006] proposed a mechanism which is most convincing and widely applicable mechanisms for generating power-laws. Concas et al. [2007] was formed in coalition with Yule et al. as it depends on how the Yule process is able to stochastically model the generation of several of the power-laws found, identifying the process parameters and comparing theoretical and empirical tail indexes. Chen et al. [2008] put forward a module-based evolution model and it described the evolution of large- scale software systems, and proves that the software systems keep the complex network features. The fat- tail distributions are present almost everywhere in software measurements, and that the kinds of distribution found in the studied systems are likely to be found in many other systems. Louridas et al. [2008] in coalition with Cai et al. [2009] state how the software follows a Power law not only at the fine grained level but ranging from microscopic to macroscopic.

All the papers found derive concepts from the work in the first paper on Valverde et al. [2003]. There are 3 papers that explicitly deal with systems, written in Java, C++, and Smalltalk languages. There are 3 papers that describe the coalition of Jing et al [2006], Baxter et al. [2006] and Ichii et al [2008], and how the authors dealt with the Software metrics in different perspectives. An attempt to incorporate Power laws in software and examine the level of constructs is made by 4 papers culminating in the integration of Yule process. The last paper in this survey research is module based evolution model that describes and proves that the Software systems keep the complex features.

Very lately, some studies have been performed on software systems showing that run-time objects and static class structures of object oriented systems are in fact governed by scale free Power law distributions. A large variety of structures, ranging from linguistic word nets to biological metabolic pathways, can be modeled by networks of nodes that communicate via links. In many of these structures, the distribution of links to nodes follows a power law distribution and such networks follow fat tailed distributions and the underlying reason of the presence of these fat tails illuminates several facets of software engineering research is still unknown.

2. SMALL TALK GRAPH

The Smalltalk language provides a powerful environment to easily build and analyze graphs representing its own classes. Moreover, the dynamically typed structure of the language adds more attention as it represents a fundamental difference with respect to other related studied languages. A software system is made of modules and relationships between them. Moreover a software application precisely developed according to the object-oriented paradigm is made of classes and well defined relationships between these classes. This section deals with analyzing the Small talk Classes, Association graphs, Component Graphs and the Power Laws.

i. Class-Method Association Graphs

Class graphs represent an important information space of OO software systems. The signature of complex software organization is a heterogeneous and hierarchical network. It is difficult to find a clear, nice decomposition of software systems and this pattern partly explains why. Class graphs are highly heterogeneous networks, where a very few classes participate in many relationships and the majority of classes have one or two relationships. The software designs are remarkably similar to many other complex networks, like the WWW, the Internet and many biological networks.

Valverde et al.[2003] state that the signature of complex software organization is a heterogeneous and hierarchical network and this pattern partly explains why it is difficult to find a clear, nice decomposition of a software system and the role of broad distributions in software measurements have been largely dismissed. The authors state that a number of software engineering topics may benefit from their approach, including empirical software measurement and program comprehension.

The authors do not refer to any previous work on this topic.

Valverde et al. [2003] introduce a new approach to software engineering based on recent advances in complex networks. They studied the graph abstractions of software designs, where nodes represent software entities (i.e., classes and/or methods) and edges represent static relationships between them (i.e., inheritance and association). The author's measure graph attributes of software designs in order to find universal patterns of software organization.

Valverde et al. [2003] conducted experiments to find if the possibility of the small-world can spontaneously emerge in an evolving OO software system and they also made comparisons between class graphs and class projections.

The author's claim that based on their analyses:

1. Partitioning an OO software system into classes and methods is very likely to result in a highly clustered software structure with small average path lengths.
2. The authors have found very good agreement between local and global measures of class graphs and class projections.

The authors claim that they have found that some graph measurements of software structures are (statistically) predictable and within a definite range of values and these patterns are almost independent of functionality and other external features. They also claim that they can obtain useful information by comparing different network representations of the same software system.

ii. Analyzing the small talk classes

Full advantage of the introspection of the language can be taken with Smalltalk. Smalltalk is endowed of powerful query methods able to return useful information, like for instance the set of all the classes of the system, the set of classes implementing a given selector, the set of messages sent within a given method, and so on. In the Smalltalk system, there are methods with the same name implemented in many classes, while most methods are implemented in one or few classes. This means that, when defining a new method, the probability that its name is the same of a method already present in the system is roughly proportional to the number of times that method is implemented in different classes.

Marchesi et al.[2004] state that there is increasing evidence that several real networks behave as small worlds, simultaneously showing a short average minimum length path and a high clustering behavior and it is hard to statically resolve all class relationships Moreover, many real networks show interesting laws in the distribution of the number of links connected to a node. The tails of such distributions follow a power law, that is a significant deviation from the Gaussian behavior that would be expected if links were randomly added to the network and they state that their study of software systems from the new perspective of statistical graph properties may be useful in suggesting novel insights into collective biological function.

The authors refer to the work of Noble et al. [2002] and Valverde et al. [2003]

Marchesi et al. [2004] claim the previous work of Noble et al. [2002] and Valverde et al. [2003] has drawbacks as they state that the graphs formed by run time objects, and by the references between them in object- oriented applications are characterized by a power law tail in the distribution of node degrees.

The authors introduce a procedure for building the class graphs of a group of Smalltalk programs, and present the related results. The authors state that Smalltalk language provides a powerful environment to easily build and analyze graphs representing its own classes. Moreover, the dynamically typed structure of the language adds more interest as it represents a fundamental difference with respect to other related studied languages.

Marchesi et al. [2004] conducted experiments on two classes A and B just to find two kinds of relationships: inheritance and dependence and also to check if class A depends on class B when one of the following conditions holds:

- Class A has an instance variable whose type is B (composition);
- A method of class A has a variable (parameter or local variable) whose type is B;
- An object of class B is obtained by a method call, or created, inside a method of class A.

The authors claim to have obtained the following result based on their experiments

- Class A depends on class B if a method of class B is called from within a method of class A.
- If the considered method has only one implementer, class B, the dependence link is clearly unambiguous.
- If the method has more than one implementer class, say n, it is not possible to ascertain with a static analysis which one is the right one.

The authors state that their approach could be an important starting point to better understand the nature and evolution of large software systems. They also claim that they suggest a new point of view for reviewing well known properties of software systems and for leading to new results that are not reachable with the classic software engineering measurement approaches.

The work was cited by Marchesi et al. [2006] and Louridas et al. [2008]

iii. Component Graphs and the Power Law

A software component graph (a component graph), which is a graph where a node represents a software component (a component) and an edge represents a use-relation between components, is widely used for the methods which support the various phases of software development. Component graphs constructed by static analysis are widely, used in various software engineering methods such as software component retrieval ,software measurement ,design recovery ,and software modularization .It is important to know the characteristic of component graphs for effective and efficient analysis.

Ichii et al. [2008] states that the modern software systems are rarely developed from scratch, that is, developed reusing various software components. They also state how a software component graph, where a node represents a component and an edge represents a use-relation between components, is widely used for analysis methods of software engineering.

The authors refer to the previous work by Valverde et al. [2002], Potanin et al. [2005], Baxter et al. [2006] and Concas et al. [2007].

The authors claim the previous work by Valverde et al. [2002], Potanin et al. [2005], Baxter et al. [2006] and Concas et al. [2007] has drawbacks and the use-relations as edges of a component graph are only "significant" types of use-relation, such as inheritance or field declaration, and ignore some types of use-relation such as method call or local variable declaration, which sometimes dominates use-relation among components. Second, some work acquires use-relations only based on lexical analysis of source code and text matching of component name, by which precise relationships are hard to be acquired.

Ichii et al. [2008] constructed component graphs using all types of use relations which are analyzable statically for the component graph based on light semantic analysis and tried to find out if the software component graphs composed of Java classes, that seek the degree distribution follows so-called the power-law, which is a fundamental characteristic of various kinds of graphs in different fields.

Ichii et al. [2008] performed experiments to see the following

1. If the in- and out-degree distributions of a component graph of a software system follow the power-law?
2. If the in- and out-degree distributions of a component graph for multiple software systems follow the power-law?
3. If the in- and out-degree distributions of sub graph of a component graph for software systems follow the power-law?
4. What aspects of components contribute to the power-law (or non-power-law) at the in- and/or out degree distribution of a component graph?

Ichii et al. [2008] claim to have obtained the following results

1. Based on the experiment 1, the in degree distribution follows the power-law almost ideally and the out degree distribution does not follow the power-law, where the distribution has a peak in the range of small values while a straight line is observed in the range of larger values.

2. Based on experiment 2, with the component graphs for multiple software systems: the in-degree distribution follows the power-law and the out-degree distribution does not follow the power-law.

3. Based on experiment 3, the subsets whose components are picked out based on a keyword has similar characteristic with the superset: The in-degree distribution follows the power-law with the similar parameters and the out-degree distribution partially follows the power-law.

4. Based on experiment 4, the in-degree relates to the roles of the components but it has low correlation with the size, the complexity, and the cohesion. They also found that the out-degree has higher correlation with the size and the complexity of the component.

The authors claim that they found interesting results from their experiments. There is assymetricity between the in- and out-degree of the component graphs constructed with various use-relations, where the in-degree follows the power-law meanwhile the out degree does not follow the power-law.

Year	Author	Title	Papers Referred to	Major Contribution
2003	Valverde et al	Hierarchical Small Worlds in Software Architecture	None	The authors introduce a new approach to software engineering based on recent advances in complex networks. They also found that small-worlds can spontaneously emerge in an evolving OO software system.
2004	Marchesi et al.	Power Laws in small Talk	Nobel et al.[2002] and Valverde et al. [2003]	The first paper to introduce a procedure for building the class graphs of a group of Small talk programs.

2008	Ichii et al	An exploration of power-law in use relation of java software systems	Valverde et al. [2003], Potanin et al. [2005], Baxter et al. [2006] and Concas et al. [2007].	This paper states that the software component graphs composed of Java classes that seek the degree distribution follows so-called the power law, which is a fundamental characteristic of various kinds of graphs in different fields.

3. SHAPE OF SOFTWARE

Understanding the shape of existing software is a crucial first step to understanding what good software looks like. It is important to select good software that has quality attributes such as high modifiability, high reusability, high testability, or low maintenance costs. The structure of software has now become so complex that it is difficult to know what the initials are held in the minds of software engineers. The Software Coupling Network is a kind of coupling between classes, it is counted when calculating Coupling Between Object classes (CBO) values of CK suite of OO metrics to get a more complete and meaningful view of the software.

i. Selecting Software

Software engineering has focused on how software could or should be written, but there is little understanding of what actual software really looks like. Quantitative models of software have been designed to predict the effort required to produce a system, measure the development rates of software over time (process metrics) or measure the volume of software in a system and its quality. But very little was known about the large-scale structures of software that exists in the real world. The Baxter et al.[2006] found that while there were distributions for which there was good evidence for a power-law, there are a number for which there was little evidence that a power-law exists. The relationship between large-scale structures found in software, and quality attributes such as understandability, modifiability, testability, and reusability should be understood and important step towards the goal.

Baxter et al. [2006] addressed how some relationships follow power-laws, while others do not and variations that seem related to some characteristic of the application itself. Much of software engineering has focused on how software could or should be written, but there is little understanding of what actual software should really look like. They also state how their study provides important information for researchers who can investigate how and why the structural relationships they found may have originated and what they portend, and how they can be managed as little is known about the structure of Java programs in the wild: about the way methods are grouped into classes and then into packages, the way packages relate to each other, or the way inheritance and composition are used to put these programs together.

The authors refer to the work of Valverde et al [2003] and Potanin et al [2005]

The authors put forward the idea to account for the origin of scale-free network structure in other domains: growth with preferential attachment and in which existing nodes link to new nodes with probability proportional to the number of links they already have, and hierarchical growth in which networks grow in an explicitly self-similar fashion.

Baxter et al. [2006] considered a large sample size corpus than has been considered by past similar studies, applied analysis techniques to characterize how closely each distribution obeyed a power law. The authors also invented a corpus of Java software consisting of 56 applications of varying sizes, and measured a number of different attributes of these applications.

The authors claim that hypothesizes that measures a relationship that the programmer is inherently aware of will tend to have a ‘truncated’ curve and not a power-law.

The work was cited by Louridas et al. [2008] and Concas et al. [2010]

ii. Software Coupling Networks

Software has become a complex piece of work by the collective efforts of many. Software networks are extracted from source code with nodes representing software components and edges representing their interactions, namely, the inheritances and collaborations between classes. It is still not clear how these relationships between the metrics come into being. However, this first approach fails to take the direction of edge into consideration. The discovery of the relationship between WMC (Weighted Methods per Class) and out degree of the coupling links stimulates the thought that there might also be such kind of correlations between other OO metrics.

Jing et al.[2006] state that they deal with two kinds of problems firstly, the software community that is challenged to put forward efficient Software Engineering methods to cope with the complexity and on the other hand, the approach taken in the complex network research is effective in studying complex objects, however, fails to catch the specific characteristic of software and thus it falls short of producing meaningful results for software community.

The authors refer to the previous work of Valverde et al. [2002] and Potanin et al. [2005]

Jing et al. [2006] claim the previous work of Valverde et al. [2002] and Potanin et al. [2005] has drawbacks and they were unable to give a thorough view of software, which impedes the understanding of software developing process as well as the software itself.

The authors state that by employing methods from the study of complex network, they investigate Object Oriented (OO) software metrics from a different perspective. By modifying the previous OO software network model, they in cooperate metrics Weighted Methods per Class (WMC) and Coupling between Object Classes (CBO) from the CK suite into their model and propose the Weighted OO Software Coupling Network to be a more integrate and meaningful characterization of the OO software. The authors also show how the SE community might be useful for exploring the inherent relationship between other software metrics.

Jing et al.[2006] conducted experiments on four open source software's varied in size they also applied the same method and study on another OO metric to see if the Lack Of Cohesion In Methods(LCOM) is more correlated with WMC and out degree or with CBO and in degree.

From their experiments, the authors state that the distributions of both metrics follow the Power law. As they correspond to weight and degree respectively in the network model, and also the same distributions exist for the network characteristic quantities. They also found that LCOM is more correlated with WMC and out degree than with CBO and in degree, and the distribution of average LCOM vs. WMC and out degree is very near to power law also.

Jing et al. [2006] claim that similar distributions exist between average LCOM and WMC as well as out degree. They also claim that their new discoveries improve understanding on the inherent characteristics of software, which will help in identifying the inner mechanisms governing software development. With this understanding, they may be able to choose appropriate SE guidelines for specific applications to get the desirable software properties, such as function distribution and structure characteristic.

Year	Author	Title	Papers referred to	Major Contribution
2006	Baxter et al	Understanding the shape of java software	Valverde et al. [2002] and Potanin et al. [2005]	The first paper to describe about the way methods are grouped into classes and then into packages, the way packages relate to each other, or the way inheritance and composition are used to put these programs together.

2006	Jing et al	Scale free in Software Metrics	Valverde et al. [2002] and Potanin et al. [2005]	The first paper to investigate Object Oriented (OO) software metrics from a different perspective.

4. INTRODUCTION TO POWER LAWS

Power-law distributions have been found in many natural, social and technological phenomena. A Power Law implies that small occurrences are extremely common, whereas large instances are extremely rare. This regularity or 'law' is sometimes also referred to as Zipf's law and sometimes Pareto law. To add to the confusion, the laws alternately refer to ranked and unranked distributions. A graph with the power-law distribution are also called "scale-free network" and it can be seen on various domains such as coauthoring relationship among scientists, hyperlink relationship among WWW pages.

The Yule, or Yule-Simon, process is one of the most convincing and widely applicable mechanisms for generating power-laws. There is evidence that power laws appear in software at the class and function level.

i. The Yule Process

The Yule process was proposed by Yule in 1925 to explain the distribution of the number of species in genus, families or other taxonomic groups of biology. The process deals with a population of entities, each having a property, characterized by an integer numeric value. The Yule process describes a mechanism for generating a population, with successive addition of entities, and with a rule for incrementing the property value of existing entities. The key issue is that, if the entity whose property has to be modified is chosen with probability proportional to the size of this property, the resulting property distribution will tend to a power-law.

Concas et al. [2006] state that they focused on some properties of OO systems neither studied nor modeled before, that show a power-law behavior. They also state how the Yule process is able to stochastically model the programming activities leading to these distributions.

The authors refer to the previous work by Valverde et al. [2003] and Potanin et al. [2005]

Concas et al. [2006] tried to find the Power-law distributions in the properties, code production, the inheritance hierarchies, the naming of variables and methods, and the calls to methods which have never been reported before.

Concas et al. [2006] state that they measured four different properties of code production, the inheritance hierarchies, the naming of variables and methods, and the calls to methods. They also explained the reasons behind the power-law behaviour of the distributions studied modeling the software development aspects yielding these properties using the Yule process.

The authors state that the analysis which they conducted proves the power-law distributions in the properties like code production, the inheritance hierarchies, the naming of variables and methods, and the calls to methods. They also prove that the power-law distributions in these properties, denoting that the programming activity in no way can be simply modeled as a random addition of independent increments, but exhibits strong dependencies on what has been already developed.

Concas et al. [2006] claim that the measurements made on the various software systems, present their statistical distributions, and try to statistically model their generation using Yule process which had never been studied before.

ii. Object Oriented Software Systems as Graphs

Power-laws are found in the distribution of entity-related features. Nowadays, object-oriented (OO) programming (OOP) is the prevalent software development paradigm. The graph for any software system can be enormous but is by no means random. A small random change in the software and in the corresponding graph can impair and crash the program. The presence of fat tails means that classes or other software entities with extreme values of the metrics are likely. These classes are where an important part of the system complexity resides and, hence, are important to evaluate the system quality.

Concas et al.[2007] address the problem of programming activity, even when modeled from a statistical perspective, can in no way be simply modeled as a random addition of independent increments with finite variance, but exhibits strong organic dependencies on what has been already developed. They also state how the Yule process is able to stochastically model the generation of several of the power-laws found, identifying the process parameters and comparing theoretical and empirical tail indexes

The authors listed the previous works of Valverde, et al. [2003], Potanin et al. [2005] and Concas et al. [2006]

Concas et al. [2007] claim that they have invented a comprehensive statistical study of an implementation of the Smalltalk object-oriented system. They also invented the Pareto or, sometimes, lognormal distributions in the properties and tried to explain the reasons underlying the power-law or lognormal behavior of the distributions studied, discussing how a program is built and which dependencies may arise.

Concas et al. [2007] conducted an experiment involving 10 system properties, including the distributions of variable and method names, inheritance hierarchies, class and method sizes, and system architecture graph and measuring the values of various properties related to OO system design and programming entities.

The authors claim to have achieved the following result

1. By normalizing the out links number on the class size, the resulting out-links distribution closely resembles a Gamma distribution.
2. The distributions found are related to existing object oriented metrics, like Chidamber et al. [1998] and how they could provide a starting point for measuring the quality of a whole system, versus that of single classes.

Concas et al. [2007] claim that they found various kinds of statistical distributions, and developed an approach to check whether the generalized Yule process could be a sensible explanation of an observed Pareto distribution identifying the process parameters and comparing theoretical and empirical tail indexes.

The work was cited by Turnu et al. [2011]

iii. Software Power Laws

The notion of power laws as a descriptive device has been around for more than a century. Power laws have cropped up in different guises in various contexts. A study using a Java source code analyzer found that twelve different metrics follow power laws. The metrics concerned the distribution of class references, methods, constructors, fields, and interfaces in classes, and the distribution of method parameters and return types. A complex piece of software is, indeed, best regarded as a *web* that has been delicately pieced together from simple materials. A more detailed follow-up study examined seventeen metrics at that abstraction level and identified fat tails, some of which would be best described by a power law, while some of them would be best described by other distributions.

Louridas et al. [2008] state that distributions with long, fat tails in software are much more pervasive than already established, appearing at various levels of abstraction, in diverse systems and languages. The authors state that the findings parallel to those studies of the static structure of software: there is no typical object size or scale and also there are no significant numbers of popular objects that are heavily referenced. They also state how the apparent ubiquity of power laws inspires intriguing images of common underlying structures, patterns, and laws governing all sorts of complex systems.

The authors refer to the previous work of Marchesi et al. [2004], Baxter et al. [2006] and Potanin et al. [2005]

Louridas et al.[2008]state the previous work of Valverde et al. [2003], Potanin et al [2005] and Baxter et al [2006] had an evidence of power laws in software but only at microscopic level, i.e.; at the level of method calls or class references. The authors state how their contribution is far more pervasive than that has been established. They also state that software follows power laws not only at the level of fine-grained constructs, but at various levels of abstraction, from the microscopic to the macroscopic

The authors state that examine power laws in software from a software engineering point of view; if software forms structures with predictable characteristics, they may be able to explain earlier empirical findings in software engineering research, exploit the structures in current practice, and also provide directions for further research.

The authors conducted experiments and chose modules of varying size and functionality, ranging from simple Java classes to systems using self-contained libraries written in C, Perl, and Ruby to see whether incoming and outgoing links in different levels of abstraction show similar patterns.

Louridas et al. [2008] claim that class graphs were generated indirectly, using the Doxygen documentation tool; the call graphs were obtained from the CodeViz package. Refactoring, the restructuring of existing code in order to optimize its design is offered as a plausible explanation for the emergence of scale-free networks in software and a study using a Java source code analyzer found that twelve different metrics follow power laws. A more detailed follow-up study examined seventeen metrics at that abstraction level and identified fat tails, some of which would be best described by a power law, while some of them would be best described by other distributions

Louridas et al. [2008] claim that power laws emerge both in products developed and released by software organizations, and as a result of ad hoc contributions from around the world and, predictably, the pattern remains the same when they examine both source code and compiled code.

The work was cited by Concas et al. [2006] and Giulio et al. [2006].

iv. Software Mirror Graphs

The concept of software mirror graph is introduced as a new model of complex networks to incorporate the dynamic information of software behavior. Only a few software laws that are physics-like have been exposed in the literature, and the software law problem is poorly investigated in general. The concept of software mirror graphs is introduced to overcome the weaknesses of the directed topological graph which ignores dynamic information of interest. The software mirror graphs adapt the directed topological graphs by taking account of various attributes that are assigned to nodes and edges to convey the dynamic information of interest. Topological measures as well as temporal measures are defined for software mirror graph. The graph describes a snapshot of software state in the course of software execution. The software execution processes are treated as an evolving directed topological graph as well as an evolving software mirror graph.

Cai et al. [2009] state that only a few software laws that are physics-like have been exposed in the literature and the software law problem is poorly investigated in general. They state that the software law problem, are indeed pertinent to the software law problem and give positive evidence that physics-like laws may obey various distinct software systems The authors state that this paper treats software execution processes as an evolving complex network and examines if there exist invariant patterns in the dynamic behavior of software systems. This inspired by the discovery of small-world effects and scale-free distributions by treating software systems as a complex network.

The authors refer to the work by Valverde et al [2003], Potanin et al [2005] and Concas et al [2007]

Cai et al [2009] state that previous studies that are related to the work, presented static software measurements and reveal that software systems may demonstrate the small-world effects and follow the scale-free degree distributions.

The authors propose software mirror graphs as a new formalism for representing the dynamic states of software in the course of software execution. In comparison with directed topological graphs, software mirror graphs can convey more and richer dynamic information of the software execution process by assigning attributes to nodes and edges. While the software execution processes can be treated as a spatially evolving process by updating the sets of nodes and edges contained in the corresponding directed topological graphs or software mirror graphs, they can also be treated as a temporally evolving process by updating the attributes assigned to the nodes and edges contained in the corresponding software mirror graphs.

To define a trial of software execution as the process of executing a pre-determined number of test cases from the initialization of a subject program the authors conducted experiments to see the following:

1. How many test cases should be executed for in a trial of software execution for a given subject program?
2. How should test cases be selected from the corresponding test suite?
3. How many trials should be conducted for a given subject program?

The authors claim to achieve the following result based on the experiment:

The software experiments for each of the three subject programs were put into operation for 40 trials. Each trial generated a single execution trace, which was a sequence of methods that were executed consecutively in the software execution process and they reveal that software systems may expose the small-world effects and follow scale-free degree distributions from the static perspective. They also observed that the software execution process may follow scale-free degree distributions if it is treated as a directed topological graph. They treated the software execution process as an evolving directed topological graph as well as an evolving software mirror graph and the authors found new findings that are pertinent to the software law problem.

Cai et al. [2009] claim that,

a. They found four distinct research perspectives for the first time.
b. The software systems as a complex network mainly adopted other research perspectives the Dynamic Evolving (DE) research perspective is adopted in this paper for the first time. The software execution processes are treated as an evolving complex network.
c. The concept of software mirror graphs is introduced to overcome the weaknesses of the directed topological graph which ignores dynamic information of interest. The software mirror graphs adapt the directed topological graphs by taking account of various attributes that are assigned to nodes and edges to convey the dynamic information of interest. Topological measures as well as temporal measures are defined for software mirror graph. A software mirror graph describes a snapshot of software state in the course of software execution. The software execution processes are treated as an evolving directed topological graph as well as an evolving software mirror graph.
d. They also claim that the software execution processes may demonstrate as a small-world complex network in the topological sense, they no longer expose the small world effects in the temporal sense. Further, the degree distributions of the software execution processes follow a power law. However, they also follow an exponential function or a piecewise power law.

This paper was cited by Valverde et al. [2003], Potanin et al. [2005] and Concas et al. [2007]

Year	Author	Title	Papers referred to	Major Contribution
2006	Concas et al	On the Suitability of Yule process to stochastically model some properties of the object oriented systems	Valverde et al.[2003] and Potanin et al.[2005]	The first paper to explain the distribution of the number of species in genus, families or other taxonomic groups of biology. It is a mechanism for generating a population, with successive addition of entities, and with a rule for incrementing the property value of existing entities.
2007	Concas et al.	Power Laws in a large object oriented software systems	Valverde et al. [2003],Potanin et al.[2005] and Concas et al.[2006]	It was based on the Yule process they describe how the Yule process is able to stochastically model the generation of several of the power laws found identifying the process parameters and comparing theoretical and empirical tail indexes.

2008	Louridas et al.	Power Laws in Software	Marchesi et al.[2004],Baxter et al. [2006] and Potanin et al.[2005]	The first paper in which the software follows the Power laws not only at the level of fine grained constructs, but at various levels of abstraction, from the microscopic to the macroscopic.
2009	Cai et al.	Software execution process as an evolving complex network	Valverde et al.[2003],Potanin et al.[2005] and Concas et al.[2006]	Only a few software laws that are physics like have been exposed in the literature, and the software law problem is poorly investigated in general. The concept of software mirror graphs is introduced to overcome the weakness of the directed topological graph which ignores dynamic information of interest.

5. MODULE BASED EVOLUTION MODEL

Large-scale software systems usually consist of a huge number of modules, and have a series of releases along with these modules. This can be seen as software evolution. Researchers have put forward several models of software evolution by employing the theory of complex networks. A module based evolution model is a refined model and it describes the evolution of large-scale software systems, and proves that the software systems keep the complex networks features. The theoretical analysis of module deals with the complex networks of software systems that show evidences of scale-free feature.

i. Theoretical Analysis of Module Based Evolution Model and Case Study

Complex networks represent totally different types of systems, they share many common features. One such well known feature is the scale free nature where the degrees of nodes conform to a power law distribution another is the small-world effect where the distance between any two nodes is rather short compared with the size of the network. Such kinds of complex networks are also called the scale-free networks. Large-scale software usually grows with many modules in software evolution. The complex networks of software systems show evidence of scale-free feature. So the network based evolution model very likely follows the power-law property.

Chen et al [2008] state that large-scale software systems usually consist of a huge number of modules, and have a series of releases along with these modules. In recent years, researchers have put forward several models of software evolution by employing the theory of complex networks but could not prove that the power-law degree distribution can be held in any model. Researchers have discovered that the complex networks of software systems show evidences of scale-free feature and the network evolved based on their model should be very likely to appear the power-law property.

The authors referred to the previous work of Valverde et al. [2003] and Potanin et al. [2005]

Chen et al .[2008] refer to the previous work of Valverde et al. [2003] and Potanin et al. [2008] and state that the duplicating and rewiring procedures which mean new nodes generated from old ones, and copy the links of them at the same time. They think more of reusing classes, but care less about which node will be duplicated.

Chen et al. [2008] put forward a refined model of software evolution based on the BA model: module-based evolution. They apply the module-based evolution model to these complex networks and simulate the evolution of key network features such as average clustering coefficient and average path length and have found that the relation between the number of outgoing edges for each new node and the times of the number appearance follows the exponential distribution.

The authors conducted experiments to see the following

- To record the number of outgoing edges for each new node, and plot a figure about the relation between Times of Number and Numbers.
- To see if the scales of the network evolved according to their model are very close to the actual scales, and to find the average relative error.
- To show the changes on two important complex networks' features: average clustering coefficient and average path length.
- To show the comparison on γin (the scaling exponent of in-degree distribution).

The authors claim to have achieved the following results based on their experiments

- Based on experiment 1 the relation is exponential by ignoring the tail, so the authors got five exponential distributions from five incremental processes.
- Based on experiment 2 it shows that the scales of the network evolved according this model are very close to the actual scales, and the average relative error is only about 1.34%.
- Based on experiment 3 the average clustering coefficient is still relatively larger, and the average path length is still relatively smaller during the experiment.
- Based on experiment 4 the average relative error on experimental data with α is only 19.3% of that on experimental data without α.

Chen et al. [2008] claim that compared to real networks, their model can precisely describe the evolution of features, and be used to help developers understand the characteristics of large-scale software evolution. The model can be used to help developers understand the characteristics of large-scale evolution.

Year	Author	Title	Papers Referred to	Major Contribution
2008	Chen et al	Module-based large-scale software evolution based on complex networks	Valverde et al. [2003] and Potanin et al. [2005]	The authors put forward a refined model of software evolution based on the BA model: module. It describes the evolution of large-scale software systems and proves that the software systems keep the complex networks features.

6. CONCLUSION

"Power laws do not enjoy a monopoly in having a good fit over a range of physical, economical, or social phenomena". There exist other distributions that also have a good fit with respect to some of them i.e; Poisson distribution and the power law distributions can be easily generated, by a variety of mechanisms.

The software execution processes behaves as a small-world complex network in the topological sense, but they no longer expose the small world effects in the temporal sense. The degree distributions of the software execution processes may follow a power law. However they may also follow an exponential function or a piecewise power law. It is often impossible to distinguish between a power-law and the tail of a log-normal distribution. "Fat tails appear in the relationships between software modules, be they functions, classes, or libraries, open source or closed source, recently developed or vintage code". The difference in scale and implementation between the modules leads to believe that it is not the specific size or technology of those modules that somehow goad a system into displaying these characteristics. The previous works reported for out-degree distribution found, in many OO systems, a power-law behavior and not a log normal one.

Software networks were first studied by Valverde et al [2003]. Software networks are extracted from source code with nodes representing software components and edges representing their interactions, namely, the inheritances and collaborations between classes. However, this first approach fails to take the direction of edge into consideration. Potanin et al [2005] and Baxter et al [2006] had evidence of power laws in software but only at microscopic level, i.e.; at the level of method calls or class references. Louridas et al. [2008] also stated the software follows power laws not only at the level of fine-grained constructs, but at various levels of abstraction, from the microscopic to the macroscopic. Chen et al. [2008] put forward a refined model of software evolution based on the BA model: module. It describes the evolution of large-scale software systems, and proves that the software systems keep the complex networks features studies that are related to the work, presented static software measurements and reveal that software systems may demonstrate the small-world effects and follow the scale-free degree distributions

Of specific interest are recent claims that many important relationships between software artifacts follow a 'power-law' distribution. If this were true, it would have had important implications on the kinds of empirical studies that are possible. Marchesi et al[2004] states clearly that their considerations are still preliminary, and the differences in the distribution metrics proposed should be correlated to externally observed properties, such as how easy it is to add functionality, how many defects a system exhibits, or how much effort is required to fix these bugs, opening up interesting future research directions. Jing et al. [2006] circumstances that it is still not clear how relationships between the metrics come into being. The author declares that the future work includes exploring the cause of the uneven distribution of cost and function as well as the close correlation between the function and the coupling. Ichii et al. [2008] states that their future works are to explore other sort of component graphs such as ones based on other types of use-relation and to discuss about applying to these results for analysis methods of software engineering. Louridas et al. [2008] states that even though they might not know the underlying reason for the presence of fat tails it illuminates several facets of software engineering research, prescribes current practice, and suggests important new research venues.

Despite of much work done by the authors they were unable to find why some distributions follow the power laws in some applications for some metrics and not others. The hypothesis of previous studies failed to explain why the distribution of a number of metrics on object-oriented software obeys a power-law. In the case of power-law distributions, there is no theory to explain why such scale-free structures in software. Two main hypothetical mechanisms have been put forward

a. to account for the origin of scale-free network structure in other domains growth with preferential attachment
b. Existing nodes link to new nodes with probability proportional to the number of links they already have, and hierarchical growth in which networks grow in an explicitly self-similar fashion. It is still far from clear, however, what (if any) fundamental theory might account for the ubiquity of the phenomenon in software

Ultimately, relationship between large-scale structures found in software, and quality attributes such as understandability, modifiability, testability, and reusability should be clearly understood to know the Power Law in Software Architecture and the OO Programming.

REFERENCES

BAXTER, G., FREAN, M., NOBLE, J., RICKERBY, M., SMITH, H., VISSER, M., MELTON, H., AND TEMPERO, E. 2006.Understanding the shape of java software. *In ACM Sigplan Notices* .41. 397-412.

CAI, K. AND YIN, B. 2009. Software execution processes as an evolving complex network. *Information Sciences* 179, 12, 1903-1928.

CHEN, T., GU, Q., WANG, S., CHEN, X., AND CHEN, D. 2008. Module-based large-scale software evolution based on complex networks. *8th IEEE International Conference on Computer and Information Technology, CIT 2008, IEEE,* 798-803

CONCAS, G., MARCHESI, M., PINNA, S., AND SERRA, N. 2006. On the suitability of yule process to stochastically model some properties of object-oriented systems. *Physica A: Statistical Mechanics and its Applications* 370, 2, 817-831.

CONCAS, G., MARCHESI, M., PINNA, S., AND SERRA, N. 2007. Power-laws in a large object-oriented software system. *IEEE Transactions on Software Engineering*, 33, 10, 687-708.

GIULIO, C., MICHELE, M., ALESSANDRO, M., AND ROBERTO, T. 2010. An empirical study of social networks metrics in object-oriented software. *Advances in Software Engineering, Hindawi Publishing Corporation,* 2010, 1-21.

HEDEFALK, F. 2009. Robustness of spatial databases: Using network analysis on gis data models. *University of Gävle, Department of Technology and Built Environment*.

ICHII, M., MATSUSHITA, M., AND INOUE, K. 2008. An exploration of power-law in use-relation of java software systems. *19th Australian Conference on. Software Engineering, 2008. ASWEC 2008. IEEE,* 422- 431.

JING, L., KEQING, H., YUTAO, M., AND RONG, P. 2006. Scale free in software metrics. *30th Annual International in Computer Software and Applications Conference, 2006. COMPSAC'06. IEEE*, 229-235.

LABELLE, N. AND WALLINGFORD, E. 2004. Inter-package dependency networks in open-source software. *Arxiv preprint* cs/*0411096*, cs.SE/0411, 1-6.

LI, H., HUANG, B., AND LU, J. 2008. Dynamical evolution analysis of the object-oriented software systems. *IEEE Congress on Evolutionary Computation, 2008.CEC 2008 (IEEE World Congress on Computational Intelligence,)IEEE,* 3030-3035.

LIU, J. AND LU, J. AND HE, K. AND LI, B. AND TSE, C.K. 2008. Characterizing the structural quality of general complex software networks. *International journal of bifurcation and chaos in applied sciences and engineering*, world scientific publishing co, 18, 2,605-613.

LOURIDAS, P., SPINELLIS, D., AND VLACHOS, V. 2008. Power laws in software. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 18, 1- 2.

MARCHESI, M., PINNA, S., SERRA, N., AND TUVERI, S. 2004. Power laws in Small talk. *ESUG 2004 Research Track*, 1-27.

POTANIN, A., NOBLE, J., FREAN, M., AND BIDDLE, R. 2005. Scale- free geometry in OO Programs *Communications of the ACM* 48, 5, 99-103.

QIN, Y., WEI, W., YE, J., and Yang, C. 2008. Research of complex networks characteristics in software architecture. *Jisuanji Gongcheng yu Sheji(Computer Engineering and Design)*, China Aerospace Science \& Industry Corporation, P. O. Box 7208-26, Beijing, 100074, China, 29,1- 9

SUN, S. AND XIA, C. AND CHEN, Z. AND SUN, J. AND WANG, L.2009. On structural properties of large-scale software systems: from the perspective of complex networks. *Sixth International Conference on Fuzzy Systems and Knowledge Discovery, 2009. FSKD'09*, IEEE, 7,309-- 313

VALVERDE, S. AND SOL_E, R. 2003. Hierarchical small worlds in software architecture. *Arxiv preprint cond-mat/0307278*. *Special Issue on Software Engineering and Complex Networks Dynamics of Continuous, Discrete and Impulsive Systems Series B (2007)*

WANG, L., WANG, Z., YANG, C., AND ZHANG, L. 2011. Evolution and stability of Linux kernels based on complex networks. *SCIENCE CHINA Information Sciences*, Springer, 1-11.

WEN, L., DROMEY, R., AND KIRK, D. 2009. Software engineering and scale free networks. *IEEE Transactions on Systems Man and Cybernetics, Part B: Cybernetics*, IEEE, 39, 4, 845-854.

ACKNOWLEDGEMENT

I have the honor to acknowledge the knowledge and support I received from Dr. Richard Frost throughout this survey. I convey my deep regards for giving me an opportunity to do this survey.

Submitted to:

Dr. Richard Frost

Professor and Acting Director at University of Windsor

Ontario, Canada